

Progetto:

- Progetto individuale di una applicazione client-server o P2P in linguaggio C.
- Da discutere in sede d'esame.
- Va consegnato 72h prima.
- Senza progetto → No Verbalizzazione.
- Le specifiche verranno date a Dicembre.

Password Materiale Pistolesi

- Username: *ri*
- Password: *retiInf2020*

Esame:

- L'orale comprende domande teoriche e pratiche.
- Non ci sono limiti alle consegne dell'esame.
- Si userà debian , la stessa che si usa per Sistemi operativi , la trovi sul sito del Prof. Avvenuti.

Premessa:

Questi appunti contengono sia la parte del prof. Anastasi che la parte del Prof.Pistolesi, anche se quest'ultima non è curata quanto la prima.

Tipologie di Reti

- **A Connessione Diretta.**
 - Collegamenti Punto-Punto.
 - Framing.
 - Rilevamento e Correzione dell'Errore.
 - Trasferimento Affidabile dei dati.
 - Controllo di Flusso.
 - Protocollo PPP.
- Reti Locali
 - Accesso Multiplo.
 - Reti Locali Ethernet.
- A Compattezza di Pacchetto
 - Ethernet + Switch.
 - Circuito Virtuale e Datagram.
 - Reti packet-switched (compattezza di pacchetto) a copertura geografica molto ampia (fino a livello globale).

Interconnessione di Reti (Internet):

Collegare varie reti diverse e farle comunicare come un'unica rete.

Ogni rete ha un formato di pacchetto , segnali , indirizzamento propri , che potrebbero essere incompatibili con altre reti.

Ad esempio se entri in una stanza dove tutti parlano arabo e te parli solo italiano , non capisci una mazza.

Protocollo IPv4:

Protocollo IP "Internet Protocol".

Permette di inviare messaggi da un pc sorgente a un pc destinatario.

Assegnazione dinamico degli indirizzi IP (Protocollo DHCP "").
Traduzione degli Indirizzi (Protocollo NAT "").
Risoluzione degli Indirizzi IP (Protocollo ARP "").
Cenni sul protocollo IPv6.

Routing:

Come fare a inviare messaggi tra una rete e un'altra?

Un router è un computer dedicato a fungere la funzione di interprete tra due reti diverse.

Internet

Internet è un sistema che interconnette un certo numero di computer (Hosts / Colui che Accoglie) , ospitano le applicazioni di rete.

Gli host sono collegati tramite la rete di accesso.

I router svolgono la funzione di intermediario , su di loro non girano applicazioni e non generano traffico , ricevono pacchetti di dati originanti sugli host.

Questi pacchetti generati sugli host sono inviati a dei router , i router poi li inoltrano presso ogni router fino al router finale.Alla fine del percorso il pacchetto è recapitato all host destinatario , da cui si estrae il messaggio e verrà inviato all'applicazione destinataria.

Gli host sono collegati al router tramite link di comunicazione , ossia il canale tramite cui viaggia il segnale.Alcuni link di comunicazione sono : Fibra , Doppino , Radio o Satellite.

Dato un link posso stimare il numero di Kb/s , ossia il Transmission Rate.

Tutti questi dispositivi , compongono delle reti , gli host si collegano al router più vicino.
I router poi compongono una rete di router.

Una rete è un insieme di Host e Router collegati mediante un link di comunicazione.

Oggigiorno ad internet sono connessi qualsiasi cosa (pensa alla domotica).

Internet of Things (IoT)

Oramai le reti sono composte più da oggetti connessi che da "puri" calcolatori.

Protocolli

Servono regole per comunicare, e nel mondo dell'informatica le regole sono definite con dei protocolli.

Definiscono come due attori devono comunicare tra loro.

Gli attori devono inviarsi certi messaggi in un certo formato.

E viene definito anche cosa fà una parte quando riceve un messaggio.

Un protocollo è un insieme di regole che definisce formato e ordine dei messaggi che si inviano 2 interlocutori della rete.

Inoltre specifica anche cosa si deve fare a fronte di un certo messaggio.

I protocolli sono definiti da comitati di standardizzazione , tipo IETF (Internet Engineering Task Force).

Gli standard sono fatti per facilitare il lavoro degli sviluppatori e per permettere a parti eterogenee di interfacciarsi.

Visione dall'alto

Internet Visto "dall'alto" , ossia da persone non ingegnerizzate.

- Un bosco visto da un elicottero appare come una macchia verde.
- Un bosco visto da dentro appare come un insieme di alberi.

Internet appare come un sistema di comunicazione universale.

Client & Server

Per client , mi riferisco al processo client (stessa cosa per server).

"Dispositivo Client" = "Dispositivo in cui gira il programma Client".

Un host può essere macchina client o macchina server , dipendentemente dal momento.

Un host può essere collegato ad un router attraverso link di comunicazione (Wireless , Wired etc).

ISP - Internet Service Provider

Le ISP sono Aziende (ISP - Internet Service Provider) che forniscono servizi di connettività , i loro clienti hanno il loro router privato connesso ai loro router centrali.

Un esempio di ISP può essere l'università di Pisa (verso studenti e dipendenti).

Un azienda piccola si rivolge a ISP "veri" , ossia che forniscono a pagamento il servizio di connettività.

Gli ISP si dividono in

- Locali , UniPi è locale , lo fornisce solo ai dipendenti.
- Nazionali , TIM è nazionale , lo fornisce ai suoi clienti sparsi in tutta Italia.
- Globali.

Reti di Accesso

Si dividono in 3 categorie:

- La rete Istituzionale, gli host sono quelli dell'istituzione.
- La rete Residenziali, gli host sono quelli che si possono avere in una casa.
- Le reti pubbliche mediante 4G o 5G, oppure di attività commerciali come un Bar.

Questi reti si differenziano per il fatto che il link di accesso è condiviso o no.

Se un canale è condiviso da 100 utenti è diverso da un canale per 2 utenti.

Il Wi-Fi è un link di accesso condiviso per tutti gli utenti connessi a quell'access point.

DSL - Digital Subscriber Line

Hosts ----- DSL Modem ----- Splitter ----- Linea Telefonica

Una compagnia telefonica (ad es. Vodafone o TIM) funge da ISP.

Velocità di downstream uguale a quella di upstream (simmetrica).

Il link di accesso è la linea telefonica.

ADSL - Asymmetric Digital Subscriber Line

ADSL è l'evoluzione della linea telefonica , infatti il link di comunicazione verso l'esterno è la linea telefonica.

A differenza della DLS il downstream è maggiore del upstream (asimmetrico).

Soggetta ad interferenze che rendono diversi canali non utilizzabili.

Modem

La funzione del modem è infatti quella di modulare (MO) e demodulare (DEM) il segnale in arrivo dal computer e dalla linea telefonica permettendo l'accesso in rete.

Il segnale digitale trasmesso dal computer viene tradotto nel segnale analogico trasmesso sulla linea telefonica e, viceversa, il segnale analogico che passa per la linea telefonica viene tradotto in un segnale digitale leggibile dal computer.

Il Modem ha il compito di modulare i dati digitali ad alte frequenze.

- Basse Frequenze → Telefono
- Alte Frequenze → Dati provenienti dall'end system.

Splitter

Lo Splitter smista il segnale in 2 parti: uno va al Modem, l'altro al telefono.



Cable-based Access

In america , invece la televisione è via cavo.

In europa è poco diffusa.

FTTH - Fiber to the Home :

Fibra ottica dalla centrale fino alle case.

FTTC - Fiber to the Cabinet :

Dalla centrale fino all'armadietto c'è la fibra.

Ma dall'armadietto alle case c'è il doppino telefonico (collegamenti in rame della ADSL).

Meno rame c'è meno interferenze avrò , quindi più veloce risulta la connessione.

Rete di Casa

Nella rete di casa ho il servizio di connettività (ADSL in europa , via cavo in USA).

Ci sono sia dispositivi connessi in maniera wired che wireless.

Più dispositivi wireless ho , più disturbata sarà il link di connessione senza fili.

Spesso nelle case moderne il DSL-modem , Router e Access-point sono condensati in un unico dispositivo.

Rete Istituzionale

"Collegano enti e servizi per lo più pubblici".

Tipicamente con un raggio di 100 m.

Ci sono sia dispositivi connessi in maniera wired che wireless.

Più dispositivi wireless ho , più disturbata sarà il link di connessione.

In questo tipo di reti , di solito i pc fissi sono wired

Rete Pubblica

Solo Wi-Fi, tipicamente con un raggio di 10 Km.

4G → 10 Mbps

Una rete di accesso contiene sempre un Router.

Il router connette la rete all'ISP.

Link di Accesso

Il segnale (analogico) si propaga sul link di accesso.

I link di accesso sono di due categorie:

- **Link di Accesso Guidati**
 - "Guidati" perché il segnale analogico segue una guida fisica.
 - **Doppino Telefonico**
 - Il segnale è di tipo elettrico.
 - Composto da un certo numero di coppie di fili di rame intrecciati secondo un determinato passo.
 - Nota che i fili intrecciati compongono una spira, e a fronte di corrente un campo magnetico.
 - Il cavo infine viene schermato attenuando le interferenze e diminuendo i disturbi.
 - **Cavo Coassiale**
 - Bidirezionale.
 - Formato da un conduttore centrale, una guaina isolante e da una maglia metallica che funge da schermatura e da filo di ritorno.
 - **Fibra Ottica**
 - Il segnale è composto dalla luce (fotoni), ogni impulso è un bit.
 - Molto affidabile e performante, ma anche molto costoso.
 - Immuni al rumore elettromagnetico e consente un *bitrate* molto elevato e con un *bit error rate* molto ridotto rispetto ai fili in rame.
- **Link di Accesso Non Guidati**
 - Il segnale si propaga liberamente, tipo i segnali elettromagnetici:
 - Radio.
 - Satellite.
 - Un problema tangibile è il delay *end-end* (circa 0,3 secondi).
 - Wi-Fi
 - Bluetooth.
 - Infrarossi.

Packet Switching - Commutazione di Pacchetto

Metodo per raggruppare i dati in pacchetti che vengono trasmessi su una rete digitale.

I pacchetti sono costituiti da un'intestazione e un payload.

I dati nell'intestazione vengono utilizzati dall'hardware di rete per indirizzare il pacchetto alla sua destinazione, dove il payload viene estratto e utilizzato da un sistema operativo, software applicativo o protocolli di livello superiore

Permette di "dividere" un messaggio in pezzi più piccoli detti "pacchetti".

Il router ha il compito di smistare questi pacchetti e lo fa secondo una precisa politica.

I router (di solito) utilizzano la politica "*Store-And-Forward*".

1. Riceve il pacchetto.
2. Lo mette in un buffer al suo interno (*store*).
3. Lo inoltra verso la direzione giusta (*forward*).

Supponiamo di avere 2 host (A e B) entrambi connessi al Router K.

1. $t=0$ A inizia a trasmettere il 1° pacchetto.
2. $t=(L/R)$ K ha ricevuto il 1° pacchetto, inizia a trasmetterlo.
3. $t=(2*L/R)$ B ha ricevuto il 2° Pacchetto.

Nota che se il router non aspettasse di ricevere i pacchetti per intero, allora non ci sarebbe questo ritardo di L/R .

Ritardo di Trasmissione.

Il push-out (invio) richiede L/R unità di tempo per trasmettere tutto.

Con L il numero di pacchetti e R la capacità del link di accesso.

- **Packet Queuing (Ritardo di Accodamento)**
 - Se il rateo di arrivo dei pacchetti è maggiore rispetto al rateo di invio, i pacchetti in entrata si accumulano e quindi vengono accodati in attesa del push-out.
- **Packet Loss (Ritardo di Perdita)**
 - Il buffer non è infinito, se il buffer si riempie i pacchetti in entrata potrebbero essere scartati, ovviamente con una certa politica.

Circuit Switching

Alternativa al packet switching, si attiva un circuito che va dal nodo sorgente al destinatario.

Prende ispirazione dal funzionamento della rete telefonica "antica".

L'intera capacità del link di accesso è dedicata interamente al trasmettitore per tutta la durata della sessione di comunicazione.

Eventuali altri host devono mettersi in attesa.

Approccio molto conservativo ma estremamente inefficiente, infatti caduto in disuso.

ISP - Internet Service Provider

Come fanno gli host a collegarsi tra di loro?

Ci sono tante reti di accesso, ognuna implementata da un diverso ISP.

Come si collegano tra di loro?

Ogni ISP ha una sua rete e ognuna di queste reti ha un router verso l'esterno (router di frontiera).

I router esterni non hanno tutti un link di accesso tra loro.

Può esistere un unico ISP nel mondo?

Non può esistere un ISP globale, per problemi di natura politica.

Ci sono svariati ISP in competizione tra loro.

Le reti degli ISP sono interconnesse tramite degli IXP (Internet Exchange Point).

Ci sono anche ISP locali che a loro volta si collegano agli ISP nazionali.

Infine ci sono i Content Provider Network per le grosse aziende che devono distribuire i loro contenuti privati :Tipo Amazon, Microsoft o Apple.

Interfaccia a Socket

(Applicazione A) Host A <----> Internet <----> Host B (Applicazione B)

Impareremo a far comunicare 2 applicazioni tramite l'interfaccia a socket.

L'applicazione A ha a disposizione una astrazione del SO , ossia il socket , che le permette di comunicare con un'altra applicazione in esecuzione su un altro host dall'altra parte della rete.

Anche l'applicazione B avrà un suo socket.

La comunicazione non è solo orizzontale , ma per trasmettere qualcosa dobbiamo usare la "pila".

La pila ha in cima il livello applicazione , dove risiedono le applicazioni A e B e termina con il livello fisico.

Per trasmettere qualcosa bisogna scendere attraverso diversi livelli di astrazione , fino al livello fisico dove esistono solo impulsi elettrici per poi "scalare" di nuovo la pila (nell'host B) per arrivare al livello applicazione e quindi all'applicazione B.

Gli impulsi elettrici sono purtroppo vulnerabili a disturbi e ad altro → Perdita di Informazione.

Pila Protocollare

Per comunicare in rete due macchine creano una astrazione fatta a livelli, quindi quando si scrive una applicazione in C

Quell'applicazione sarà in esecuzione su un Host.

La nostra applicazione, sul nostro Host è a un livello di astrazione molto alto (infatti il C++ è un linguaggio ad alto livello).

La pila protocollare è un insieme di livelli , ognuno con il suo livello di astrazione.

Quando voglio (dall'host A) inviare dati da qualche parte , la nostra applicazione molto spesso deve scambiare dati con un'altra applicazione in esecuzione su un'altra macchina (Host B).

L'informazione , per trasferirla , la devo impacchettare in un messaggio.

Per fare ciò, io programmatore devo interagire con il kernel, e dirgli che devo inviare un messaggio all'host B, ovviamente per dirglielo devo usare le System Call fornite dal sistema operativo.

Il programmatore può usare determinati protocolli (ossia modi per inviare il messaggio) che garantiscono che il messaggio arrivi a destinazione senza nemmeno un bit corrotto.

Il messaggio (detto "Payload" ossia "informazione per cui pago il trasporto") è creato dopo aver impacchettato le informazioni che voglio inviare.

Ogni livello si occuperà di una specifica funzionalità , dove più si scende più il livello di astrazione cala.

Perchè io devo scendere tutti questi livelli?

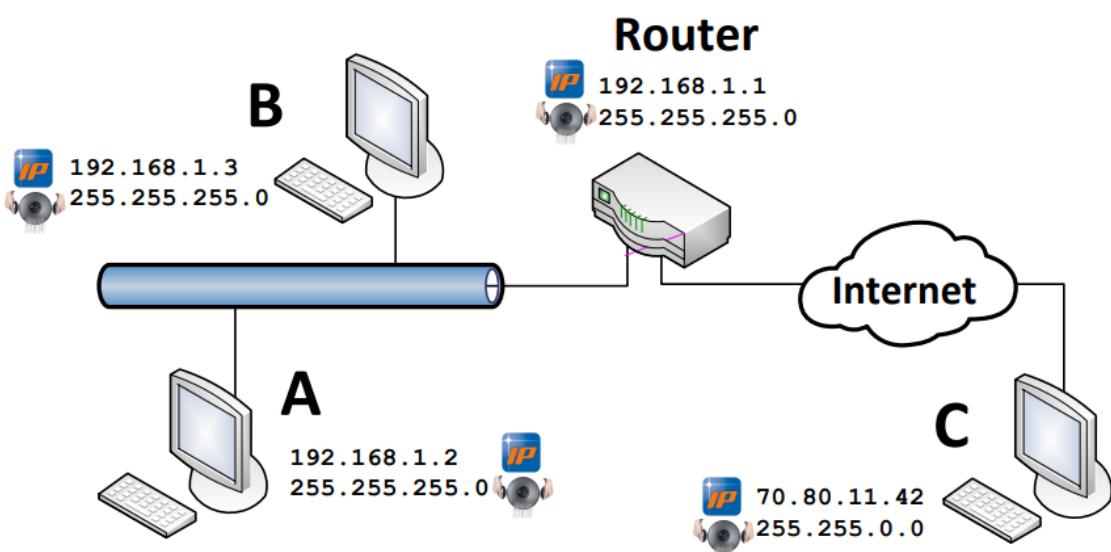
Ogni livello ha una funzione molto specifica , ogni livello si preoccupa ad esempio del trasporto , applicazione , data link , rete etc.

Nella discesa il messaggio subisce dei trattamenti che lo rendono sempre più vicino ad una rappresentazione elettrica (impulsi).

Esempio:

- Caramella : Payload
- Carta : Tutta la roba che aggiungono i livelli.

Una Semplice Rete



Di cosa ha bisogno l'host A per comunicare in rete?

Sia per andare verso l'host B o verso l'host C.

Il gateway (router) decide quale sarà il prossimo router che prenderà a carico la richiesta dell'host A.

Il router applica una specie di algoritmo di Dijkstra per trovare il percorso a costo minimo.

La nuvola in realtà è rappresentabile come un grafo , i cui nodi sono altri router.

Restando in ambito di grafi.

Quali sono i fattori che determinano il peso di un arco (una route che collega 2 router)?

- Distanza del link di connessione.
 - Lunghezza del filo , ciò ha implicazioni nel ritardo di propagazione del segnale e sulla probabilità che esso subisca disturbi.
- Livello di Congestione della Route.
 - Il cammino più corto potrebbe essere intasato (perché tutti lo preferiscono) , quindi come in strada : "*Evito l'Aurelia perché è trafficata , quindi uso le stradine interne , allungo di 1 Km ma ci metto 30 minuti meno*".

Cosa serve ad un Host per comunicare?

- **Indirizzo IP :**
 - Come la targa dell'automobile, l'indirizzo IP è dato dall'ISP nel momento in cui il router viene acceso (la targa è data quando la macchina viene messa in strada all'uscita del concessionario).
- **Maschera di Rete :**
 - Serve a manipolare l'indirizzo IP e dividerlo in due parti:
 - Indirizzo di Rete.
 - Indirizzo dell'Host nella rete.
- **Indirizzo del Gateway (Default Gateway).**
 - Indirizzo IP in cui posso trovare il Router nella rete.
- **Indirizzo del Server DNS :**
 - Il server DNS (Domain Name Service) è un server che permette di tradurre un nome simbolico (ad esempio : www.repubblica.it) in un indirizzo IP raggiungibile dal nostro Host ossia un indirizzo IP.
 - Senza di esso obbligherei l'utente a scrivere manualmente gli indirizzi IP (che ricordiamo, spesso sono dinamici).

Indirizzo IP

Identifica un Host univocamente nella rete.

In IPv4 è una sequenza di 32 bit.

In realtà , l'indirizzo IP identifica l'interfaccia di rete all'interno della rete.

Se un host ha una connessione cablata e una Wi-Fi, allora ho un indirizzo per la scheda di rete cablata e uno per il Wi-Fi, ovviamente in 2 interfacce diverse.

Gli indirizzi IP vengono espressi spesso in Notazione Decimale Puntata: 192 . 168 . 1 . 2

Dove 8 Bit vengono tradotti in un numero tra 0 e 255 , un indirizzo IP ha 4 ottetti di bit , quindi 4 numeri compresi tra 0 e 255.

Un indirizzo IP è scomponibile in 2 parti:

- Indirizzo di Rete
 - Identifica la rete.
 - Composto da k bit.
- Indirizzo di Host
 - Identifica l'host nulla rete.

- Composto da $(32 - k)$ bit.

Prima k era fisso e variava tra { 8 , 16 , 24 } , ora con gli indirizzi IP classless k varia tra [1 , 31].

Maschera di Rete

Sequenza di 32 bit, con tanti bit a 1 quanti sono i bit associati all'indirizzo di rete (ossia k , il numero dei suddetti bit).

Grazie alla maschera di rete e con banali operazioni logiche posso estrarre sia l'indirizzo di rete che quello dell'host.

Spesso si usa una notazione compatta → < *Indirizzo_IP ; k* >

Indirizzo di Broadcast

Indirizzo che permette di inviare un messaggio a tutti gli host della rete (più precisamente, a tutti gli host in un determinato dominio di broadcast).

Si ottiene facendo la OR bit a bit tra l'indirizzo IP e la netmask negata, ossia è l'indirizzo IP della rete con il massimo valore della parte dedicata agli Host

Indirizzo del Gateway

Il mio host ha un indirizzo IP e una mask, per ora posso inoltrare pacchetti solo a host nella mia stessa rete.

Nel file di configurazione della scheda di rete, sotto l'indirizzo di broadcast , ci scrivo :
gateway 192.168.1.1.

Se ho un pacchetto che non è riferito a nessun host della mia rete , allora lo mando al default gateway.

Quello che verrà dopo lo vedremo quando faremo i Router.

Indirizzo del Server DNS

Quando cerco qualcosa su Google vorrei usare nomi simbolici al posto di scomodi indirizzi (e facilmente dimenticabili) IP.

Il server DNS traduce un nome simbolico in un indirizzo IP.

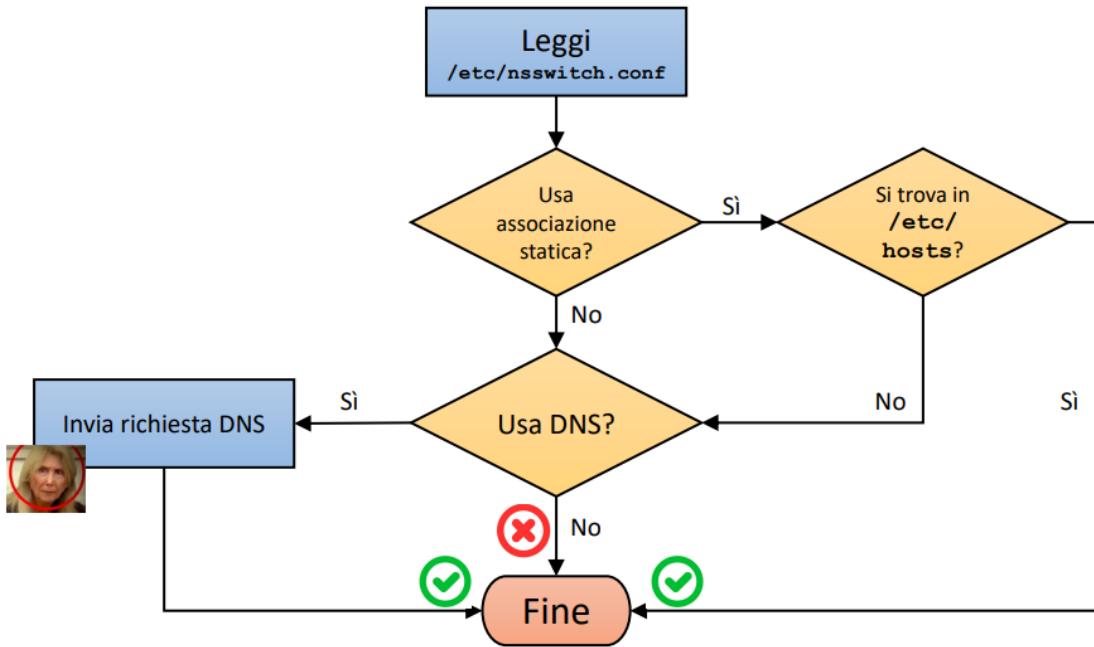
Come funziona la risoluzione del nome?

La mia macchina ha le traduzioni scritte in una lista in */etc/hosts* , se in quella lista non c'è allora l'host non riesce a risolverlo e quindi devo interagire con un server DNS.

Il Servizio DNS nella realtà è implementato come un enorme database distribuito su molteplici Server DS e di tipo gerarchico.

Quando il client effettua una richiesta a un server DNS , che risponde con l'indirizzo IP.

Dopo aver ricevuto l'indirizzo IP , il client lo salva nel file apposito per la risoluzione dei nomi (in modo da non dover rifare la stessa richiesta in seguito).



Ritardi della Rete

La trasmissione non è ideale ed è impossibile che ci sia un ritardo pari a 0.

Se il **rate di arrivo** (*Arrival Rate*, detto "a") è maggiore del **rate di invio** (*Output Rate*) si comincia ad accumulare un ritardo dovuto a l'accumularsi dei pacchetti in arrivo nel buffer di entrata (*Ritardo di Accodamento*).

Siano:

- **L:**
 - Dimensione del pacchetto.
 - Espressa in “Numero di Bit”.
- **R:**
 - Il Rateo del link di accesso, detto anche “Capacità del Link di Accesso”.
 - Espresso in “Numero di Bit trasmessi per Secondo” .

$$\text{Intensità del Traffico: } \frac{L * a}{R}$$

$$d_{\text{nodo}} := d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{drop}}$$

- **d_proc :**
 - Ritardo dovuto alla processazione del pacchetto.
 - Molto piccolo e dipende dall'hardware del nodo stesso.
 - Nell'ordine di microsecondi , circa 10^{-6} .
- **d_queue :**
 - Ritardo di Accodamento.
 - L'entità di questo ritardo è molto variabile, può essere trascurabile (caso molto fortunato , ossia quando il buffer di entrata del router è completamente

vuoto o comunque con pochissimi pacchetti in attesa) oppure molto grande in caso di buffer di entrata quasi pieno.

- **d_trans :**

- Ritardo dovuto alla trasmissione del pacchetto.

$$d_{trans} = \frac{L}{R}$$

- **d_prop :**

- Ritardo di Propagazione , ossia il tempo che richiede al pacchetto di propagarsi lungo il link.
 - Dipende dalle prestazioni del canale fisico.
 - $d_{drop} = d/s$
 - d : Lunghezza del mezzo fisico , espressa in metri.
 - s : Velocità di Propagazione , è costante ed è approssimabile a 20^8 m/sec.

Perdita di Pacchetto

Quando si perdono dei pacchetti, aumenta il ritardo, aumenta il traffico complessivo della rete (a causa dei rinvii dei pacchetti persi) e di conseguenza anche il consumo di energia elettrica.

Throughput

Rateo con cui i bit sono inviati dal mittente al destinatario.

Espresso in “Numero di Bit trasmessi per Secondo” .

Si divide in 2 categorie (come la velocità in fisica):

- **Istantaneo :**

- Misurato in un istante di tempo preciso.

- **Medio :**

- Misurato prendendo come riferimento un intervallo di tempo.

I throughput è diverso dalla capacità del link, infatti Throughput < Capacità Link.

Effetto “Collo di Bottiglia”

Nel caso di un percorso composto da vari link di accesso eterogenei:

Su tutto il percorso si guarda il link con capacità minore e si stabilisce il “*Collo di Bottiglia*”.

Il throughput generale della trasmissione è limitato superiormente dal collo di bottiglia.

Se un link con capacità massima pari a C è condiviso da n Host, allora la sua capacità si suppone sia C/n.

Pila dell’internet Protocol

- **Livello Applicazione.**

- Dove risiedono le applicazioni di rete.

- **Livello Trasporto.**

- Il nome è un pò ingannevole.
 - La funzionalità di importante è quella di multiplexing e demultiplexing, ossia recapitare i messaggi ai rispettivi processi che risiedono a livello applicazione.

- Altre funzionalità (opzionali) sono riassumibili in "portare gli stessi vantaggi di una comunicazione punto-punto" e sopperire a eventuali mancanze del livello Data Link.
- **Livello Network.**
 - Si occupa dell'indirizzamento e instradamento attraverso il percorso di rete più appropriato.
 - Di solito questo è il livello massimo a cui arrivano i dispositivi di rete "intermedi".
- **Livello Data Link.**
 - Si occupa di permettere un trasferimento affidabile dei dati attraverso un canale considerato a priori inaffidabile.
- **Livello Fisico.**
 - Si occupa di fare la conversione da digitale ad analogico e viceversa.

A ogni livello, durante la discesa del pacchetto , aggiungono intestazioni al pacchetto. Le intestazioni servono a dare importanti informazioni al pacchetto.

Gli host implementano tutta la pila protocollare.

Applicazioni di Rete

Le applicazioni girano solo sugli host della rete, non sui router (comunque il router ha dei processi interni ma che differiscono per natura dalle applicazioni di rete che invece possono girare solo sugli host).

In particolare i programmi server nelle macchine server e i programmi client nelle macchine client.

I possibili modelli di applicazioni di rete sono 2:

- **Client-Server**
 - Ci sono molteplici processi ma si dividono in 2 categorie:
 - **Server**
 - Offre servizi ai Client.
 - Sempre Acceso.
 - Indirizzo IP permanente.
 - La Macchina Server ha elevate prestazioni.
 - Richiede una banda elevata.
 - La Macchina Server ha molta memoria.
 - Spesso per motivi di scalabilità , sicurezza e affidabilità il processo server risiede in un data center , per evitare di comprare e gestire una macchina server "in casa".
 - **Client**
 - Si collega al processo Server per richiedere servizi.
 - Non sempre acceso, spesso rimane spento (o comunque non richiede servizi).
 - Indirizzo IP possibilmente variabile, assegnato dinamicamente.
 - Non comunica direttamente con altri client, un client comunica solo con i server.

- In genere richiede poche risorse, poiché di solito la maggior parte del carico computazionale è affidato al server.
- **Peer to Peer**
 - Tutti sono allo stesso livello e un host può ospitare sia un processo client che un processo server , dipende da quello che deve fare.
 - “*Self Scalability*” nuovi peers portano sia altra domanda che altra offerta , le risorse sono distribuite su una grande moltitudine di peers (risorse polverizzate) , invece nel modello client-server le risorse sono concentrate nel server.
 - Indirizzo IP non per forza permanente e sono connessi in modo intermittente.
 - I peers non devono essere per forza ricchi di risorse, ovviamente se lo sono è meglio, ma in questo caso la quantità può sopperire la qualità.

Socket

I processi devono comunicare , nel modello peer-to-peer la comunicazione avviene tra moltissimi processi contemporaneamente.

Quanti sono le modalità di comunicazione tra processi?

La zona di memoria condivisa non va bene , perché si può usare solo se due processi sono sulla stessa macchina.

Socket - Send & Recv

Il Sistema operativo mette a disposizione 2 primitive: send e recv per l'invio e la ricezione di messaggi tra processi , non per forza sulla stessa macchina.

Lo scambio di messaggi prevede che un processo invii tramite la send e che uno riceva tramite la recv.

Questo scambio , in caso di processi in macchine diverse, è gestito dal sistema operativo attraverso la rete, per fare questo ci sono delle strutture dati chiamate **socket** , ossia un astrazione creata dal sistema operativo che equivale ad una cassetta della posta.

Ogni processo coinvolto in uno scambio di messaggi ha accesso ad almeno un socket. Su un socket si può fare un'operazione di send (metto la posta nella cassetta) o di recv (apro la cassetta e vedo se c'è qualcosa).

Indirizzamento Processo - Numero di Porta

Un host deve essere identificato in modo univoco nella rete , un dispositivo host ha un indirizzo IP a 32 bit.

L'indirizzo IP identifica l'host , ma come identificare un processo all'interno dell'host?

Il numero della porta (associata al socket) identifica un processo all'interno di un determinato host.

IP Address: 128.119.245.12
Port Number: 80

Protocollo a Livello Applicazione

Ogni applicazione è regolata da un protocollo, che definisce il tipo, formato e la semantica dei messaggi che devono essere scambiati.

I protocolli si dividono in 2 categorie:

- *Open* : Realizzati da enti di standardizzazione , come ad esempio HTTP , SMTP etc.
- *Protocolli Proprietari* : Realizzati da privati , ad esempio Skype o BitTorrent.

Integrità dei Dati

Dipendentemente dal tipo di applicazione : si potrebbe richiedere che il 100% dei bit siano trasportati integri (ad esempio applicazioni di posta o applicazioni di trasferimento di file) indipendentemente da quanto tempo richiede , oppure ci sono applicazioni (ad esempio chiamate o audio) in cui non importa che arrivi il 100% dei bit, basta che sia veloce.

Servizi di Trasporto

Ce ne sono solo 2:

- **Protocollo TCP - Transmission Control Protocol**
 - Affidabile.
 - Controllo del Flusso
 - Controllo della Congestione.
 - Lento a causa dei controlli di integrità e sull'ordine di arrivo.
 - Richiede di stabilire una connessione tra le due parti prima dell'emissione dei dati.
 - i. Si apre una connessione TCP.
 - ii. Si inviano tutti i dati.
 - iii. Si chiude la connessione TCP.
 - Non garantisce sicurezza o un throughput minimo.
- **Protocollo UDP - User Datagram Protocol**
 - Non affidabile.
 - No controllo del flusso o della congestione.
 - Veloce , non ci sono controlli sull'ordine di arrivo o di integrità.
 - Non richiede una connessione.

Protocollo HTTP - HyperText Transfer Protocol

Porta 80 e utilizza TCP.

Questo protocollo serve per lo scambio di informazioni tra un client e un server nel web.

1. Il Client invia una richiesta di connessione TCP al Server.
2. Il Server accetta la richiesta di connessione TCP del client.
3. Si stabilisce la connessione TCP.
 - *Connessione TCP Persistente* :
 - In una connessione TCP vengono inviati molteplici oggetti.
 - *Connessione TCP Non Persistente* :
 - Viene aperta (e poi chiusa) una connessione TCP a ogni oggetto.
4. Il Client invia una richiesta HTTP per la risorsa desiderata.
5. Il Server elabora la richiesta ed invia una risposta HTTP.
6. La connessione TCP viene chiusa dal client, oppure dopo un timeout verrà chiusa dal server.

HTTP è Stateless

HTTP non mantiene informazioni relative a richieste client passate.

Se il browser manda n richieste, allora riceverà n risposte e tutte queste richieste verranno trattate allo stesso modo come se le precedenti non fossero mai esistite.

I protocolli che mantengono uno stato sono molto più complessi perché le informazioni relative allo stato devono essere gestite in modo che rimangano sempre consistenti.

Ai giorni nostri invece HTTP ha degli elementi "di tipo stateful" grazie all'introduzione dei Cookies , ma comunque la sua natura è stateless.

Messaggio di Richiesta HTTP

Formattato in formato ASCII, quindi leggibili da un essere umano.

Linee:

1. Linea di Richiesta

- Dice cosa fare (GET , POST) e quale risorsa vuole il client (URL).
- Specifica inoltre la versione HTTP usata.

2. Host

- Indica il nome del server a cui è diretta la richiesta.
- obbligatorio dall'introduzione dei Virtual Host , per cui più processi server HTTP risiedono su un'unica macchina, e quindi su un unico indirizzo IP.

3. User-Agent

- Indica il browser usato dal client e la sua versione.

4. Accept

- Indica i tipi di file che desidera il client.

5. Accept-Language

- Indica la lingua desiderata dal client.

6. Accept-Encoding

7. Connection

- Persistente (*keep-alive*) o non Persistente.

8. Body

- In genere nelle richieste è vuoto.

/r Ritorno Carrello.

/n line-feed.

Un messaggio HTTP può essere di due tipi:

- Request (Richiesta HTTP).
- Response (Risposta HTTP).

Stateless è sempre buono?

Non sempre, in caso di richiesta errata sarebbe meglio se il client si memorizzi l'esito della richiesta in modo da non disturbare il server.

Cookies

C'è modo di rendere il protocollo HTTP stateful? Sì, attraverso il meccanismo dei cookies.

Per implementare i cookie basta inserire qualche linea delle risposte e richieste HTTP.

I siti web e i client usano i cookie per memorizzare alcuni stati tra le transazioni.

Si hanno 4 componenti:

- Cookie header line of http response message.
- Cookie header line in next HTTP request message.
- Cookie file kept on the user's host , managed by the user's browser.
- Back-end Database del Sito Web.

Meccanismo dei cookies:

1. Il client invia una richiesta HTTP al server.
2. Se l'utente non ha nessuna corrispondenza nel sistema del server (quindi se è la prima volta che visita la pagina), il Server crea un User ID (*Supponiamo 1678*) per il richiedente.
3. Il server inserirà il cookie in un suo Database back-end.
4. Oltre alla risposta HTTP, il server invia il cookie con il relativo User ID, il client se lo memorizza "in casa" per le richieste future.
5. Alla prossima richiesta del client, quest'ultimo metterà in una particolare linea di intestazione del messaggio di richiesta il codice del punto 3. "cookie: 1678".
6. Segue una classica risposta HTTP dal server.

Attraverso i cookie, il server ha modo di "riconoscere" gli utenti.

I cookie infatti sono sfruttati per associare dati ad ogni client che ha avuto a che fare con il nostro server, uno dei possibili utilizzi è quello di non far inserire di nuovo certi dati dagli utenti, come ad esempio le generalità per evitare ogni volta di fare il login.

Sono usati anche (soprattutto) a fini pubblicitari, sempre nel rispetto della privacy...

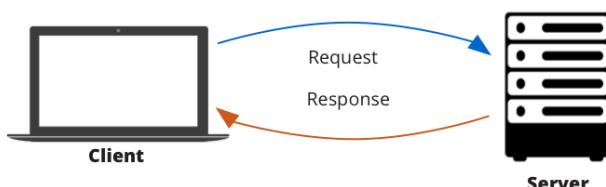
Gli usi dei cookie sono regolati secondo precise leggi sulla privacy.

Web Caches - Server Proxy

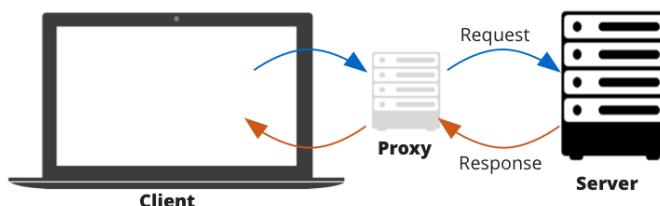
E' possibile soddisfare le richieste del client senza coinvolgere il server?

Infatti quando geograficamente il server risulta molto lontano il ritardo di propagazione risulta non trascurabile, senza contare eventuali congestioni.

connessione diretta client – server



connessione intermediata da un proxy



I proxy server sono macchine su cui gira il processo proxy, esso fa sia da client che da server.

1. Il client fa una richiesta HTTP verso un server destinazione.
2. Se nel Client è stato selezionato un server proxy, la richiesta viene inoltrata a lui
 - a. Il proxy fa da server web rispetto al client.
 - b. Il Client non si accorge che c'è il server proxy.

3. Il proxy (dopo aver eventualmente processato la richiesta) inoltra la richiesta al server destinazione
 - a. Il proxy fa da client rispetto al server web di destinazione.
4. Il proxy server riceve la risposta del server, memorizza una copia dell'oggetto in una sua memoria
5. Il Proxy inoltra la risposta al client.
 - a. Il client riceve la risposta con un piccolo ritardo.
6. Alla prossima richiesta verso lo stesso oggetto, il server proxy restituisce l'oggetto senza interpellare il server destinazione, riducendo il tempo di risposta in modo drastico e riducendo il traffico.

Il Server Proxy risulta molto efficace quando si parla di documenti o file da scaricare, comunque di oggetti statici e non dinamici che cambiano spesso e che quindi richiederebbero aggiornamenti continui.

Vantaggi del Server Proxy

- Meno carico sul server web di destinazione, perché non dovrà rispondere al client ogni volta.
- Ci guadagna un'organizzazione dove molti client si connettono mediamente alle stesse risorse, la velocità di risposta aumenta.
- Il traffico sull'intera internet cala.
 - Meno traffico di rete, quindi meno problemi di congestione.
 - Meno consumo di elettricità.

Svantaggi del Server Proxy

- Tempi di risposta un po' più lenti quando si richiede un oggetto per la prima volta.
- Copie nel server proxy potrebbero diventare obsolete, ogni tanto va fatto un aggiornamento.
- In caso di pagine web dinamiche o con molta interazione con il server il proxy risulta abbastanza inutile.

Esercizio sui Ritardi di Rete e sul Proxy

Access Link Rate: 1.54 Mbps

RTT (ritardo) from institutional router to server: 2 secondi.

Web Object Size : 100 Kilobit.

Average request rate form browsers to origin servers: 15 Richieste al secondo.

Average Data to Browsers: 1.5 Mbps

LAN Utilization: .0015 → “Ritardo sperimentato nella rete locale”

Access Link Utilization = 0.97 → 97% → “Il link è caricato al 97% della sua capacità”

Dato l' elevatissimo utilizzo il Ritardo sarà molto elevato.

Rappresenta un Collo di Bottiglia.

$$\text{Data Rate} = \text{Acc. Link Rate} * \text{Acc. Link Ut.} = 1.54 * 0.97 = 1.49 \text{ Mbps}$$

```
eed = end-end delay = Internet Delay + Access Link Delay + LAN  
delay
```

Link con capacità superiore

Ad esempio metto un link con un *Access Link Rate* di 150 Mbps.

A quel punto l'*Access Link Utilization* cala al 9.7%.

Il costo in denaro però è molto elevato.

Inserire un server proxy nella rete locale

Molte delle richieste saranno soddisfatte dal proxy, riducendo l'uso del link di accesso. Come soluzione risulta molto più economica, perché basta piazzare un comune pc con prestazioni poco elevate che ospiti solo un processo proxy.

Supponendo che si abbia un *Cache Hit* sul 40% delle richieste, queste richieste verranno soddisfatte direttamente dal Server Proxy.

Quindi solo il 60% delle richieste usano il link di accesso.

Data Rate = $0.6 * 1.54 = 0.9$ Mbps.

Access Link Utilization = $0.9 / 1.54 = 0.58 \rightarrow 58\%$.

Per il 60% dei casi si ha un *eed* = 1.2 s.

Quello che si avrebbe normalmente se non ci fosse il proxy, ma comunque con l'utilizzo del canale abbassato.

Per il 40% dei casi si ha un *eed* quasi nullo, si parla di microsecondi, dovuto solo al ritardo nella LAN.

Il costo in denaro della Soluzione Proxy è decisamente molto inferiore.

Conditional GET

Ogni tanto le copie nel proxy devono essere aggiornate, ma non troppo spesso, altrimenti i vantaggi del Server Proxy scompaiono.

Quando si manda un messaggio di richiesta, il proxy può inviare una get condizionale al server.

"Inviami l'oggetto SOLO SE è stato modificato dopo questa data"

Dove la data è contenuta nel server proxy ed è associata alla copia dell'oggetto in questione. Se non è mai stata modificata il server invia al proxy un messaggio vuoto.

HTTP 2

Nella seconda versione è aumentata la flessibilità per quanto riguarda l'invio di messaggi dal client al server.

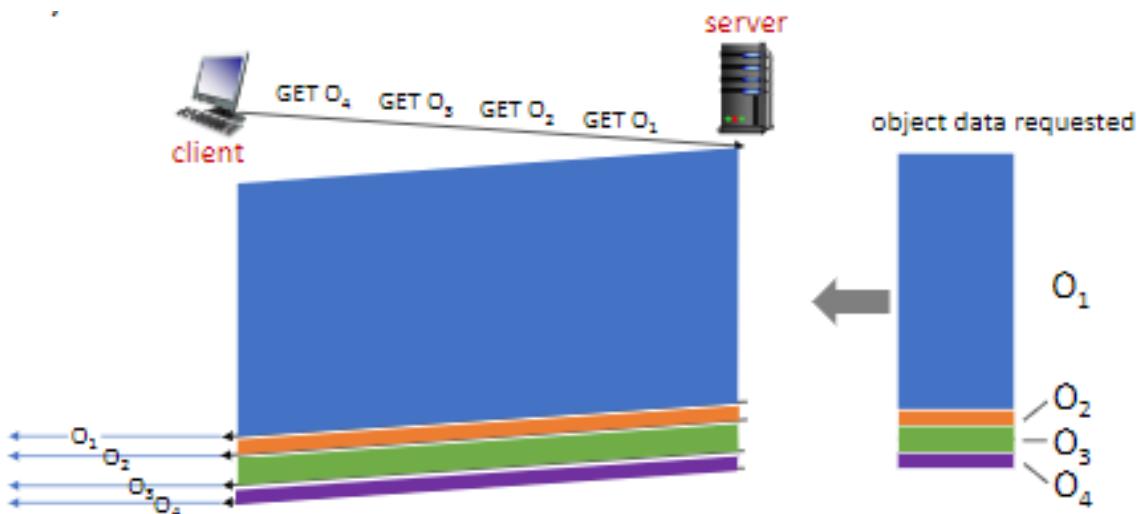
Head-of-Line Blocking (HOL)

Se io voglio scaricare molti oggetti, un possibile problema è quello del *"Head-of-line blocking"*, ossia quando qualcuno che deve scaricare tantissime cose blocca l'intera rete per molto tempo, perché per soddisfare la sua richiesta serve molto tempo.

C'è modo di risolvere questo problema?

Se quel qualcuno apre una connessione diversa per ogni oggetto in parallelo. Per come funziona il protocollo TCP, esso tende ad essere “giusto”, ossia tutte le connessioni vengono trattate allo stesso modo a prescindere da chi le apre, quindi è vantaggioso aprire tante connessioni, tutto questo a costo del consumo di memoria del Server.

Lo head-of-line blocking è la conseguenza della politica di scheduling FCFS (First Come First Served), ma se qualcuno deve fare tantissime cose blocca tutta la coda.

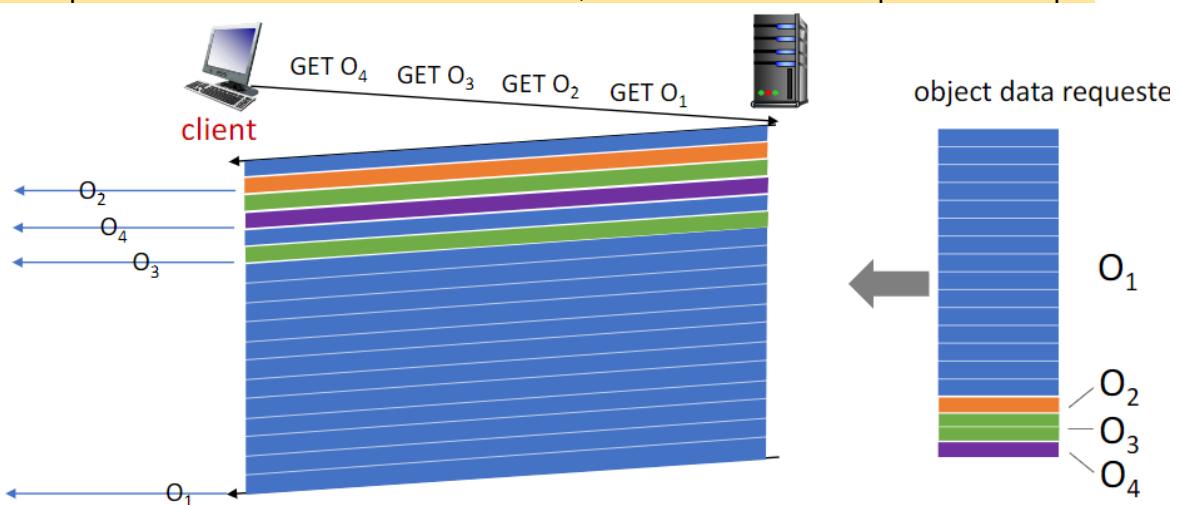


Round Robin

Invece che schedulare in maniera FCFS è meglio utilizzare la politica “Round Robin”, dove ognuno ha un pò di tempo (in questo caso ognuno ha un pò di frame).

Gli oggetti da spedire vengono divisi in vari frame.

A turno le richieste vengono soddisfatte per un pò alla volta, tramite questa politica l’esperienza utente migliora, perché gli utenti scaricano poco vengono soddisfatti subito, mentre per chi scarica molto non cambia molto, ci metteranno comunque molto tempo.



HTTP 3

Aggiunge sicurezza e controllo sugli errori e sulla congestione.

Posta Elettronica

Basata sul paradigma client server.

Lo User Agent è il programma che il client usa per gestire (inviare , ricevere , eliminare etc) la posta elettronica.

Il server di posta elettronica è quello che effettivamente gestisce la posta, lo user agent è solo uno strumento per interfacciarsi con il server.

Nel caso della posta esiste il protocollo **SMTP** (*Simple Mail Transfer Protocol*).

Mailbox

Struttura dati che contiene tutte le mail destinati all'utente.

SMTP - Simple Mail Transfer Protocol

Il protocollo SMTP è descritto in RFC 5321.

Utilizza il TCP come protocollo di trasferimento , perché le mail sono considerate file.

La porta di SMTP è la 25 (inizialmente); ora SMTP ha diverse varianti, ognuna con la sua porta.

Come avviene la comunicazione tra processo server e client?

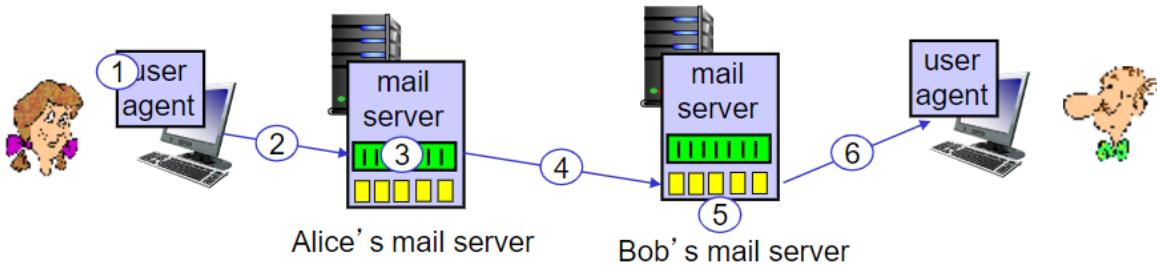
1. Il Client apre una connessione TCP persistente con il suo Server SMTP.
2. Si esegue il protocollo SMTP.
3. Chiusura Connessione TCP.

L'interazione è di tipi Command/Response → Il client invia una richiesta e il server risponde (come nel protocollo HTTP).

Il messaggio è codificato su ASCII NVT 7-bit.

Inizialmente non c'erano i file allegati.

1. A usa il suo User Agent per comporre la mail per B.
2. A invia la mail al suo Mail Server.
 - a. User Agent di A apre una connessione TCP con il Mail Server di A
 - b. Invio Messaggio.
 - c. Chiusura Connessione TCP.
3. Il Mail Server di A riceve il Messaggio.
4. Il messaggio di A è accodato nella coda dei messaggi nel Mail Server di A.
5. Il Mail Server di A invia il messaggio al Mail Server di B.
 - a. Il Mail Server A apre la connessione con il Mail Server di B
 - b. Invio Messaggio.
 - c. Chiusura Connessione TCP.
6. Il Mail server B capisce che il messaggio è destinato a B.
7. Il Mail Server B mette il messaggio nella Mailbox di B.
8. B successivamente si connette con il suo User Agent al suo Mailbox server.
 - a. User Agent B apre la connessione con il Mail Server di B
 - b. Scarica il Messaggio dalla Mailbox B.
 - c. Chiusura Connessione TCP.



La parte verde è la coda dei messaggi in uscita.

La parte gialla è la user mailbox.

Interazione SMTP

1. Richiesta del client verso il server
2. 220 ham.edu → *Convenevoli di Apertura, il server si presenta.*
3. HELO cre.fr → *Convenevoli di Apertura , il client si presenta.*
4. 250 Hello cre.fr , pleased to meet you. → *Convenevoli di Apertura , il server risponde*
5. MAIL FORM: <alice@cre.fr>
6. 250 alice@cre.fr... sender ok
7. RCPT TO: <bob@ham.edu>
8. 250 bob@ham.edu ... Recipient ok
9. DATA → *Il Client vuole inviare i Dati.*
10. 354 Enter Mail, end with “.” on a line by itself. → *Il server dice quale è il carattere di terminazione.*
11. Do you like ketchup? → *Testo del messaggio.*
12. How About Pickles? → *Testo del messaggio.*
13. . → *Carattere che indica la fine del testo.*
14. 250 Message accepted for delivery.
15. QUIT → *Convenevoli di Chiusura.*
16. 221 ham.edu closing connection. → *Convenevoli di Chiusura.*

Protocolli “Push”

Il client apre una connessione con il server e la tiene costantemente attiva (always-on connection).

Il server invierà (push) tutti i nuovi eventi al client usando quella connessione sempre accesa.

Protocolli “Pull”

Il client in modo periodico si connette al server , controlla l'esistenza di nuovi eventi e nel caso scarica tutti gli eventi recenti e infine si disconnette dal server.

Il client ripete questa procedura ogni volta che deve ottenere (pull) nuovi eventi dal server. Un esempio di protocollo Pull è HTTP.

Note SMTP

- SMTP ha una tendenza di tipo push, perché i server SMTP mantengono tra di loro una connessione costante
- L'unica componente Pull di SMTP è quando il client del destinatario si connette al suo Server Mail.

- SMTP usa connessioni TCP persistenti, quindi se il client deve inviare tanti messaggi si usa sempre la stessa connessione TCP
- In SMTP molteplici oggetti sono associati ad un unico messaggio, il quale viene spezzettato in molteplici sottomessaggi.
 - In HTTP invece ogni oggetto è incapsulato in un proprio messaggio..
- In SMTP usa “CRLF.CRLF” (ritorno carrello ripetuto 2 volte) per terminare una riga.
- Il mail server di A può risiedere nel computer di A?
 - In linea di principio sì , ma il computer di A dovrebbe rimanere sempre acceso , altrimenti si perderebbero nuovi eventi.

Struttura del Messaggio SMTP

Il messaggio che il client invia al server SMTP è formato da 3 parti:

- **Header.**
 - Ogni riga di intestazione inizia con una Keyword , abbastanza autoesplicativa.
 - **To:** Utente Destinatario.
 - **From:** Utente Mittente.
 - **Subject:** Oggetto della Mail , *Copiato dal messaggio dell'utente.*
- **Riga Vuota.**
- **Body.**
 - Corpo del messaggio , con solo caratteri ASCII.

Protocollo POP - Post Office Protocol

Protocollo di livello applicativo di tipo client-server che ha il compito di permettere, mediante autenticazione, l'accesso da parte del client ad un account di posta elettronica presente su un host server e scaricare le e-mail dell'account stesso.

I messaggi di posta elettronica, per essere letti, devono essere scaricati sul computer (questa è una notevole differenza rispetto al protocollo IMAP), anche se è possibile lasciarne una copia sull'host.

In pratica, il client, attraverso un account configurato POP, accede al server remoto e salva in locale i messaggi (definitivamente e non, come fa il protocollo IMAP, in una sorta di cache).

Il protocollo per inviare posta è invece il protocollo SMTP.

Protocollo IMAP - Internet Mail Access Protocol

Le mail restano sempre memorizzate nel server , il client non deve scaricare le mail.

Tramite il client posso gestire (inviare , eliminare o ordinare) le mail direttamente nel server , se mi connesso con un altro dispositivo trovo comunque le ultime mail.

Con IMAP posso organizzare le mail in directory.

I server dovranno essere più potenti e con molta più memoria.

Protocollo DNS - Domain Name Service

Usare nomi simbolici (come ad esempio www.grullo.com) invece che indirizzi IP per navigare in rete.

DNS si basa su UDP.

Database Distribuito

Il Servizio DNS si basa su un sistema distribuito tra numeri server, organizzato secondo un sistema gerarchico.

In cima c'è il server Root.

Seguono i top level domain.

1. Root Server

2. Top Level Domain (TLD)

- a. .edu
- b. .com
- c. .gov
- d. .edu
- e. **Country Top Level Domain (Compresi nei TLD)**
 - i. .it
 - ii. .uk
 - iii. .fr
 - iv. .de
 - v. .eu

3. Organizzazione Internazionali

- a. .int

Univocità dei Domini

L'univocità dei domini è garantita da una unità di registrazione.

In ognuno di essi stabilisco un'unità di registrazione che prima di registrare un nuovo dominio si rivolge all'autorità di registrazione (propria di ogni top level domain).

Per registrare si paga una tassa.

Sottodomini

I top level domain poi si dividono in altri sottodomini.

Le varie organizzazioni registrano il proprio dominio ,ad esempio *unipit.it*

All'interno delle organizzazioni grosse ci sono pure sotto domini, questi ultimi sono controllati da un'autorità di registrazione propria dell'organizzazione.

Ogni sottodominio è organizzato allo stesso modo : Un'autorità di registrazione a cui le persone fanno richiesta di inserire un nuovo nome.

Servizi del Sistema DNS

- **Risoluzione dei Nomi**

- Dato un nome simbolico, restituisce l'indirizzo IP.
- Questa cosa è fatta mediante una tabella con righe del tipo : < Nome Server , IP >.

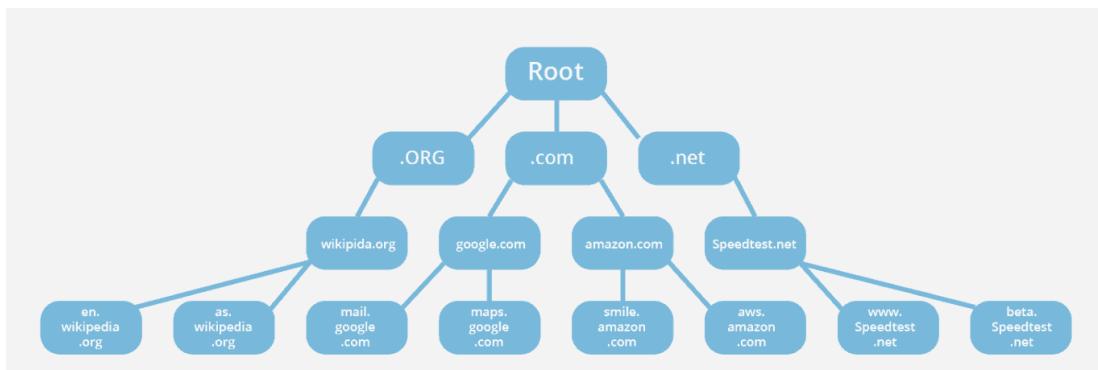
- **Host Aliasing**

- I server web hanno tutti nomi molto chiari (www.fiat.it o www.amazon.it), questi sono i nomi che vede l'utente (gli Alias), in realtà i nomi sono molto più brutti come ad esempio (www.server1.it).
- Tramite Aliasing posso dare più Alias alla stessa Macchina Server.

- Fatta mediante una tabella con righe del tipo: < Alias , Nome Server >.
 - Nome Canonico (www.server1.it).
 - Alias (www.amazon.it).
- **Mail Server Aliasing**
 - Stessa cosa ma per i server di Posta.
- **Load Distribution**
 - Spesso i grandi Siti WEB risiedono su molteplici macchine server replicate, questo per non centralizzare tutto e aumentare l'affidabilità (evitando i single point of failure), si nota perché abbiamo svariati indirizzi IP per un solo nome simbolico.
 - Il Server DNS ha la facoltà di “scegliere” quale indirizzo IP restituire, in questo modo è possibile distribuire il carico su varie macchine server senza che l'utente si accorga di nulla.
 - Come fa il DNS a scegliere? Fà a rotazione sulla lista di indirizzi IP associati a quel nome simbolico , in questo modo distribuisce in modo equo su tutti i server della distribuzione.

Gerarchia dei Server DNS

I server DNS sono distribuiti secondo una approccio gerarchico (un albero) .



Come funziona la Risoluzione dei Nomi

1. Supponiamo che un client cerchi un sito “amazon.com”.
2. Il client cerca il server DNS “.com”.
3. Il client invia una richiesta al server DNS dedicato ai domini “.com”.
 - a. Se il server DNS lo sa invia l'indirizzo IP.
 - b. Se il server DNS non lo sa invia l'indirizzo IP di un server DNS di un sottodominio.

Root Name Servers

Sono 13 i server che implementano la root, ogni server è replicato molte volte (più di 200). ICANN gestisce i Root DNS Servers.

Per ogni dominio (.edu , .com etc) però ci sono le proprie autorità di registrazione.

Authoritative DNS Servers

Un Server DNS è **authoritative** in un determinato dominio se è SEMPRE in grado di restituire un indirizzo IP appartenente ad un dominio

Di solito le query DNS finiscono sempre per trovare il risultato in uno di questi server.

Local DNS Server

Come un server proxy ma per le traduzioni.

Non appaiono nella gerarchia perché si possono considerare allo stesso livello degli Host Normali.

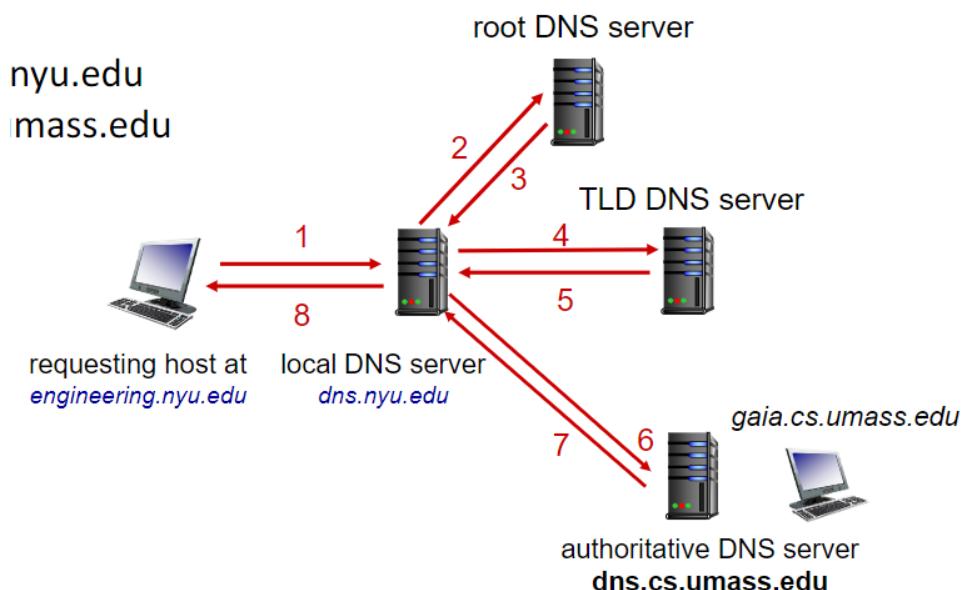
Di solito ogni ISP ne ha uno chiamato “*Default Name Server*”, infatti spesso le reti domestiche si interfacciano inizialmente con il Local DNS server del proprio ISP e sarà quest’ultimo ad eseguire la query con il root DNS server.

Query DNS

Le query DNS possono essere fatte seguendo 2 approcci.

Approccio Iterativo

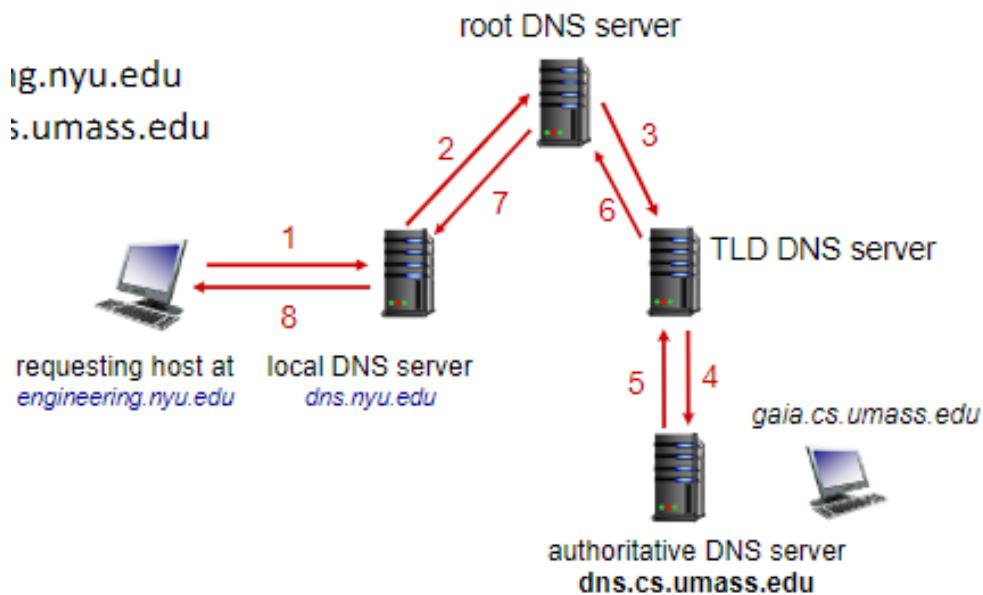
1. Quando un client fa una richiesta per “gaia.cs.umass.edu” di traduzione la invia prima al DNS locale
 - o Il Local DNS server se ha già la traduzione la restituisce subito (perché ha la funzione di caching per le traduzioni).
 - o Altrimenti:
 - i. Si rivolge al Root DNS Server, se non lo sa mi restituisce l’indirizzo IP del TLD DNS Server dei “.edu”.
 - ii. Si rivolge al TLD DNS Server, se non lo sa mi restituisce l’indirizzo IP Authoritative DNS Server per “.umass.edu”.
 - Quest’ultimo lo sa perché è Authoritative e restituisce l’indirizzo IP al Local DNS.
 - iii. Il Local DNS salva in cache locale una copia della traduzione.
 - iv. Il Local DNS Restituisce l’indirizzo IP al Client iniziale.



Approccio Ricorsivo

1. Quando un client fa una richiesta per “gaia.cs.umass.edu” di traduzione la invia prima al Server DNS locale
 - a. Il Local DNS server se ha già la traduzione la restituisce subito.
 - b. Altrimenti:

- i. Si rivolge al Root DNS Server, se non lo sa allora il Root DNS Server si rivolge al TLD DNS Server dei “.edu”.
- ii. Il Root DNS si rivolge al TLD DNS Server, se non lo sa allora il TLD DNS Server si rivolge all’ Authoritative DNS Server per “.umass.edu”.
- iii. Quest’ultimo lo sa perché è Authoritative e restituisce l’indirizzo IP al TLD DNS Server.
- iv. Il TLD DNS Server restituisce l’indirizzo IP al Root DNS Server.
- v. Il Root DNS Server restituisce l’indirizzo IP Al Local DNS Server.
- vi. Il Local DNS Server restituisce l’indirizzo IP all’host.



DNS Records.

Formato RR → (Name , Value , Type , TTL)

- **TTL** → Indica il tempo di vita del record , esaurito questo tempo , il record può essere eliminato dal database in base al valore del campo type.
- **Type** → Indica il tipo di Record.
 - **Type = A**
 - Name è il nome di un Host
 - Value è un indirizzo IP.
 - **Type = NS**
 - Name è il nome di un dominio. (es. foo.com)
 - Value è il nome di un host di un server authoritative.
 - **Type = MX**
 - Name è un indirizzo di posta elettronica. (Es → gmail.com)
 - Value è il nome canonico (quello brutto) del mail server
 - **Type = CNAME** (è un record per il servizio di traduzione alias → nome canonico):
 - Name è l’alias.
 - Value è un indirizzo IP.

Messaggi DNS

Richiesta e risposta hanno lo stesso formato:

- Header → Composto da 12 Byte.

- **Identification:** Numero a 16 bit che identifica una richiesta , le risposte usano lo stesso identificativo.
- **Flags:**
 - Indica se il messaggio è una Richiesta (query) o una Risposta (reply).
 - Indica dove il client può richiedere la ricorsione.
 - Il server indica se la ricorsione è possibile o meno.
 - Se acceso → Il server è authoritative.
- Numero di Domande.
- Numero di Record di Risposta.
- Numero di Record Authority (di risorsa).
- Numero di Additional Informations.
- Body:
 - Questions → Nome e Tipo per una richiesta.
 - Answers → RR in risposta ad una richiesta.
 - Authority → Records per i server authoritative.
 - Additional Information.

Registrare il Nome simbolico di un server

Bisogna registrare il nome del dominio, “networkutopia”.

Supponiamo di voler registrare sotto il dominio “.com”.

Ci sono delle società di registrazione, dette DNS registrar, dove è possibile registrare il nome del dominio (se il nome non è stato già registrato), e forniscono il nome del dominio, l’indirizzo IP dell’autoritative name server (primario e secondario).

A quel punto il registrar inserisce due entry all’interno del “.com” TLD server:

1. **Networkutopia.com, dns1.networkutopia.com, NS** : Serve per localizzare il DNS server primario;
2. **Dns1.networkutopia.com, 212.212.212.1, A** : Si specifica qual è l’indirizzo IP del DNS server primario.

Stessa cosa si ripete per il DNS Server secondario

Nelle Authoritative DNS server si mette poi record di type A e record di type MX.

Architettura Peer to Peer

Esempio di applicazioni peer to peer solo alcune di file sharing (BitTorrent), alcune di streaming (KanKan) o VoIP (Skype).

Spesso hanno un’architettura ibrida: la ricerca dei file è client-server; la file distribution è p2p.

1. Il client richiede al server dove può trovare il file.
2. Il server restituisce al client un indirizzo IP.
3. Il client allora in modalità P2P si fa inviare il file dall’host corrispondente all’indirizzo IP ricevuto.

Indice per lo scambio dei file in un’architettura P2P

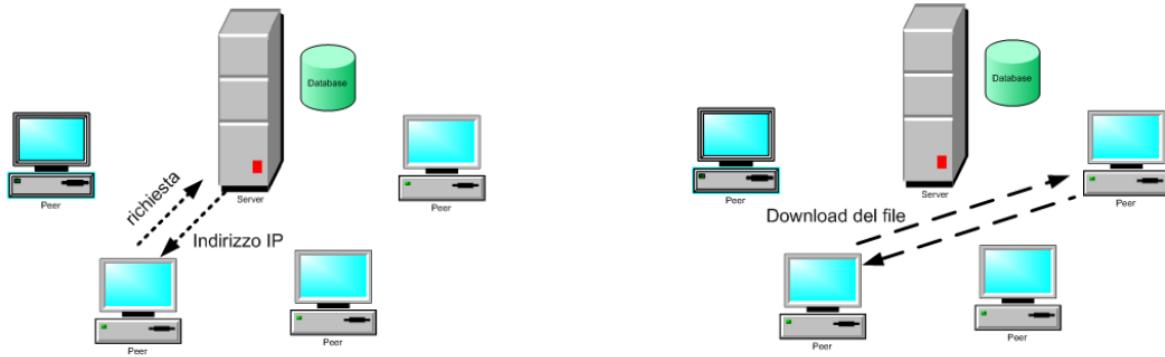
In molti casi le applicazioni P2P non sono P2P “pure”, ma hanno un’architettura ibrida (es. nelle applicazioni di messaggistica peer to peer bisogna sapere con chi scambiare contenuti, poi es. ci si può collegare con dispositivi diversi, con indirizzi IP diversi...) e bisogna sapere se interlocutore è collegato e qual è il suo IP.

Serve allora un indice, e cioè un database dove ogni record è fatto da due campi:

- Il contenuto che si vuole cercare (es. un video che si vuole scaricare, il nome di una persona che si vuole chiamare).
- Indirizzo IP corrispondente a tale contenuto (quindi l'indirizzo IP dell'host che ha tale video, o della persona che si vuole chiamare).

Indice → < Led Zeppelin IV , 203.17.123.38 >

Centralized Index



Chi vuole avere un certo contenuto manda una query al DB chiedendo la chiave, ed ottiene il valore corrispondente a tale chiave, ossia l'indirizzo IP con cui deve comunicare.

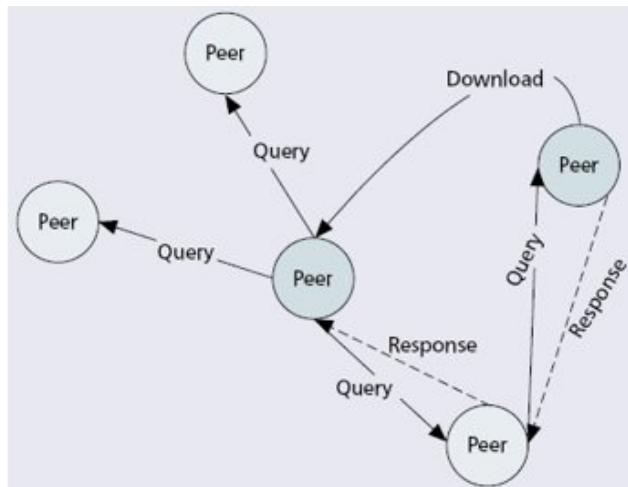
A quel punto l'host destinatario farà da server e il richiedente farà da client, la cosa è P2P perché le due macchine assumono quel ruolo solo in quel determinato frangente.

I problemi di avere un indice centralizzato sono gli stessi di quando si decide di fare una soluzione centralizzata: “*single point of failure*” e il server potrebbe non riuscire a gestire tutte le richieste.

Entrambi i problemi si risolvono aumentando il numero di server.

In caso di problemi legali viene attribuita esclusivamente a chi mantiene il server di indicizzazione.

Query Flooding



Utilizzato in Gnutella.

Soluzione completamente decentralizzata → Distribuire l'indice su tutti i peer.

Ogni peer ha un suo database, costituito dalle entrate relative ai contenuti che ha a disposizione (quindi un'entrata per ogni contenuto che hanno, l'indirizzo IP non è necessario visto che sono loro ad avere il contenuto).

1. Quando un peer vuole un certo contenuto, manda una query a tutti i peer vicini (per "vicini" si intende i peer con cui ha fatto almeno una connessione TCP in passato), chiedendo tale contenuto.
2. Se questi peer, consultando il proprio database, vedono che non hanno il contenuto, mandano la stessa query ai loro nodi vicini, e così via.
3. Alla fine, se viene trovato qualcuno che ha questo contenuto, si stabilirà una connessione diretta tra il richiedente e il possessore.
 - a. Il possessore diventa un nuovo "vicino" del client richiedente.

Il vantaggio principale è la completa decentralizzazione e quindi l'assenza totale di single point of failure.

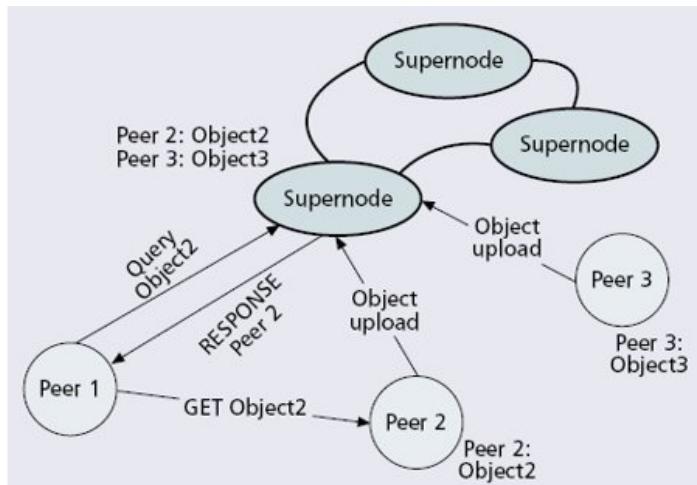
Si hanno vari svantaggi del query flooding:

- La ricerca di un contenuto può richiedere molto tempo,
- Il traffico sulla rete aumenta poiché è "inodata" di queste richieste di controllo, e così come è stato definito, la richiesta potrebbe essere inoltrata all'infinito.

Una soluzione a questi svantaggi potrebbe essere fare in modo che, dopo un certo periodo di propagazione, gli host smettono di propagare la query.

Questo però può portare a *falsi negativi*: un host non trova la risorsa non perché non c'è, ma perché non si è riusciti ad arrivare ad almeno un host che la fornisce, però nella rete dei peer quella risorsa ci potrebbe essere.

Hierarchical Overlay



Usato nel Gnutella moderno.

E' una via di mezzo tra le due soluzioni : esistono i cosiddetti *super Nodes*, e cioè dei peer speciali che oltre ad avere molta banda ed un'elevata disponibilità (è acceso per la maggior parte del tempo), hanno il compito di realizzare e distribuire gli indici dei file.

L'indice è distribuito, ma non troppo.

1. Se un peer ha un contenuto da dare alla comunità → Contatta un Supernode
2. Viene registrato sul Supernode un'entrata <Contenuto , IP del Peer>
3. Se un secondo peer vuole un certo contenuto → Contatta un Supernode:

- Se il supernodo sa chi ce l'ha, manda l'indirizzo IP al richiedente;
- Altrimenti, inoltra la richiesta agli altri Supernode nello stesso modo in cui viene fatto il query flooding.

Gli indici vengono tutti ammassati sui supernodi e il database degli indici è distribuito solo sui supernodi.

Quindi il query flooding viene fatto solo tra supernodi.

DHT - Distributed Hash Table - NO ESAME

Un DHT è un database distribuito gestito da Peers.

Usato in emule.

Sistema distribuito che fornisce un servizio di ricerca simile a una tabella hash : le coppie *chiave-valore* sono archiviate in un DHT e qualsiasi nodo partecipante può recuperare in modo efficiente il valore associato a una determinata chiave .

Il vantaggio principale di un DHT è che i nodi possono essere aggiunti o rimossi con un lavoro minimo intorno alla ridistribuzione delle chiavi.

Le chiavi sono valori univoci che mappano a valori particolari , che a loro volta possono essere qualsiasi cosa.

La responsabilità di mantenere la mappatura delle chiavi ai valori è distribuita tra i nodi. Ciò consente una scalabilità estremamente elevata , portando il sistema a poter gestire un numero enorme di nodi.

P2P vs Client-Server

Sia F la dimensione di un File.

Sia N il numero di Peer della rete.

Client-Server

Quando ci mette un server a distribuire F a N client?

Il Server deve fare un upload sequenziale di N copie di F.

Supponiamo che U_s sia la capacità di invio del Server.

- Per inviare una copia : $\frac{F}{U_s}$, supponiamo costante.
- Per inviare N copie : $\frac{N*F}{U_s}$, aumenta linearmente con N.

Supponiamo che d_{min} sia la capacità minima di download di un client.

- Per scaricare una copia : $\frac{F}{d_{min}}$, supponiamo costante.

Tempo per distribuire tutte le N copie:

$$D_{c-s} \geq \max\left\{\frac{N*F}{U_s}, \frac{F}{d_{min}}\right\} \rightarrow \text{Aumenta linearmente con N.}$$

P2P

Supponiamo che U_s sia la capacità di invio del Server.

All'inizio solo il Server fa l'upload del file.

- Per inviare una copia : $\frac{F}{u_s}$, supponiamo costante.

Supponiamo che d_{min} sia la capacità minima di download di un client.

- Per scaricare una copia : $\frac{F}{d_{min}}$, supponiamo costante.

I clienti in tutto devono scaricare $N * F$ bit

Supponendo che U_i sia la capacità di upload del client i.

Anche i client nella architettura P2P possono fare l'upload del file quando lo ottengono tutto.

$$\text{Quindi la velocità di upload } \underline{\text{totale}} \text{ è limitata superiormente da: } u_s + \sum_{i=1}^N u_i$$

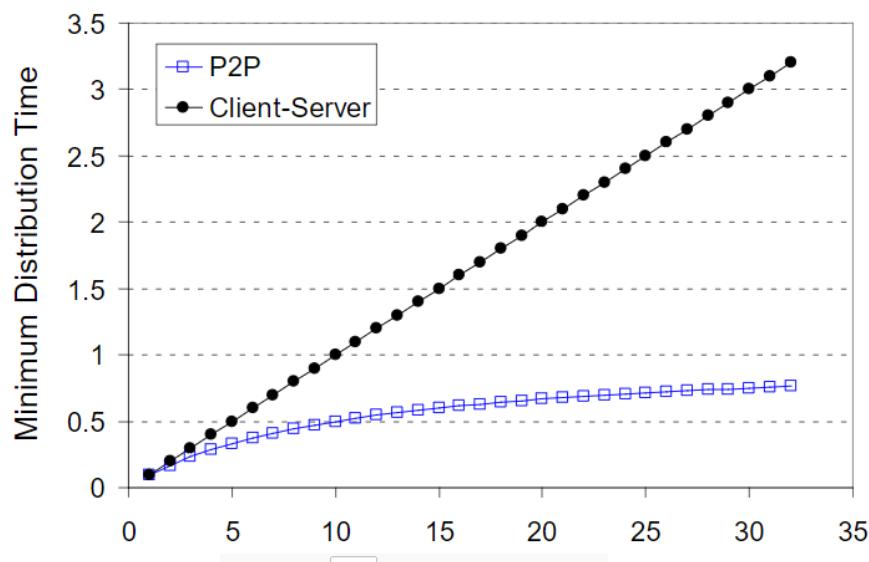
Tempo per distribuire tutte le N copie:

$$D_{p2p} \geq \max\left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{N*F}{(u_s + \sum_{i=1}^N u_i)} \right\}$$

Al crescere di N anche la sommatoria cresce allo stesso modo.

Quindi il risultato non aumenta linearmente con N.

$$\text{client upload rate} = u, F/u = 1 \text{ hour}, u_s = 10u, d_{min} \geq u_s$$



- In che senso noi intendiamo che un dispositivo può essere client o server?
(Un dispositivo si dice che è server se in esso gira il processo server)

- Nell'approccio P2P un dispositivo assume uno dei due ruoli dipendentemente dalla situazione, se deve fare upload assumerà il ruolo di server e il peer corrispondente farà da client.
- I peer devono essere sempre attivi?
 - No, una rete P2P si affida al numero dei peer per funzionare, se uno dei peer è spento di sicuro ce ne saranno altri accessi in grado di sostituirlo.
- È più vantaggioso utilizzare un approccio client-server o peer to peer per inviare un file?
 - L'approccio P2P risulta molto vantaggioso quando il numero dei peer attivi e che condividono è alto.
 - L'approccio CS ci dà la garanzia che in ogni caso qualcuno che ha il file e che è sempre acceso ci sia ogni volta che ne abbiamo bisogno.
- Quando N tende ad infinito cosa succede? Sia caso client-server che caso peer to peer.
 - Nel caso CS il tempo di attesa minimo aumenta linearmente con N .
 - Nel caso P2P tempo di attesa minimo aumenta in maniera logaritmica, nel caso in cui tutti i peer rimangano a condividere il file.

BitTorrent

Nel query flooding, il file viene scaricato da un solo peer, anche se la risorsa è disponibile da più peer.

Se il contenuto è disponibile su più peer, lo si può scaricare da più peer contemporaneamente.

Un peer possessore è molto lento, rappresenta un collo di bottiglia.

L'idea di scaricare il file da più peer è considerata ad esempio nel protocollo BitTorrent.

In BitTorrent ci sono diversi "personaggi":

- **Chunk:**
 - Pezzetto di un file, è grande 256 Kbyte.
- **Torrent Server:**
 - Server non facente parte della rete di distribuzione dei chunk (ossia non condivide chunks).
 - Ha il compito di mantenere in memoria i file '.torrent' e permette a chiunque di scaricarli.
- **Torrent:**
 - Un gruppo di peer che si scambiano chunks di un determinato file.
 - Per ogni file dunque si ha un torrent.
- **Tracker :**
 - Server con il compito di tracciare i peer che partecipano in ogni torrent.
 - Per ogni torrent stilano una lista dei peer coinvolti.
 - I Server Tracker sono specificati nel file .torrent.
- **Leechers:**
 - Peer che non ha ancora completato il download del file.
 - Ricevono i chunks, poi dopo averli ricevuti tutti ricostruiscono il file originale.
 - Allo stesso tempo carica i chunk che possiede.
- **Seeders:**
 - Peer che hanno scaricato tutti i chunk di un file, ma che continuano a rimanere nel torrent per distribuirli.

- Inviano i chunks.
1. L'Host A vuole scaricare un file.
 2. Per aggiungersi ad un torrent attivo, contatta il *Torrent Tracker* che gli invia l'indirizzo IP del *Tracker*.
 3. L'Host A contatta il Tracker.
 4. Il Tracker invia all'Host A la lista dei peer che fanno parte del torrent che ha scelto.
 - Il Tracker aggiunge A come nuovo peer nel torrent.
 5. Host A inizia ad aprire **Connessioni TCP** con tutti i peer della lista.
 - La richiesta di connessione può fallire se:
 - Il peer non ha più il file.
 - Il peer è spento o comunque non raggiungibile.
 6. Dopo aver stabilito un certo numero di connessioni TCP, i peer con cui Host A ha stabilito una connessione vengono chiamati **Vicini**.
 7. Periodicamente, Host A chiede la lista dei chunk mancanti ai suoi vicini.
 - Host A inizia con il chiedere i chunk più rari
 - Ossia i chunk posseduti da meno peer nella lista, perché se chiudessero la connessione non ci sarebbero peers pronti a sostituirli dato che ce li hanno solo loro.
 - I Chunks possono essere spediti in qualsiasi ordine, quando arrivano tutti verranno ordinati da Host A in modo opportuno
 8. Intanto l'Host A invia i suoi chunks ai vicini che li chiedono, ma non può rispondere a tutti i vicini che fanno richiesta del chunk, quindi in questo caso si applica un criterio di scelta chiamato **Tit-for-Tat (Dente per Dente)**.
 - Host A da priorità ai peer che "lo hanno trattato meglio", ossia a chi gli ha mandato più chunks per unità di tempo.
 - Di solito vengono selezionati i primi 4 della classifica.
 - In genere la graduatoria viene fatta ogni 10 secondi.
 - I primi peers in questa classifica vengono privilegiati.
 - Uno svantaggio in cui si incappa è che i nuovi peers si ritroveranno per forza in fondo a queste liste e ci si fossilizzerebbe sugli stessi peers "storici".
 1. Se la cima della classifica non cambiasse mai verrebbero scelti sempre gli stessi peers escludendo tutti gli altri.
 - Per evitare questa problematica ogni 30 secondi viene scelto un altro peer a cui verranno inviati dei chunks proiettandolo in cima alla classifica, quindi dandogli l'occasione di inviare chunks sperando che questo peer sia più veloce rispetto agli altri vicini.
 - Questa scelta permette all'host A di non fossilizzarsi solo su questi 4, perché il nuovo peer potrebbe avere velocità superiore a uno dei 4 top peers.
 - Questo sistema penalizza i *Free-Rider*, ossia coloro che ricevono chunks ma che non ne inviano mai.
 - Un free-rider non andrà mai in testa alla classifica e quindi avrà una velocità di download minima, invece chi oltre che a scaricare invia pure avrà velocità molto più apprezzabili:
 - (Non Importante) Ad Esempio : Quando installi per la prima volta QBitTorrent e scarichi il tuo primo torrent (ovviamente solo .iso di

Linux) avrai una velocità nell'ordine di *KiloBits/s* , ma se terrai tutte le tue bellissime .iso nel pc e permetterai a QbitTorrent di fare anche upload , quest'ultimo invierà continuamente chunks ad altri peers , dopo un pò di tempo avrai velocità di scaricamento nell'ordine di *MegaBits/s*.

- come si fa a sapere quali peer fanno parte del torrent?
 - Contatta il Server Tracker.
- Come fa un peer per entrare in un torrent e scaricare un file.
 - Contatta il Torrent server e si fa dare la lista dei Tracker.
 - Contatta il Tracker e si fa dare la lista dei peer del torrent.
 - Crea una comunicazione TCP con ogni peer del torrent e invia a tutti loro la lista dei chunk mancanti.
- Se uno si registra con il torrent server come fa il tracker a sapere che il nuovo peer si è registrato?
 - Il peer invia una richiesta di registrazione al server Tracker.
- Come misura lui la velocità di upload degli altri peer verso di lui?
 - Il peer esegue questa misura in base al numero di chunk che un determinato peer gli ha inviato.
- Poi fa una classifica, e una volta fatta la classifica?
 - Invia i chunk solo ai primi k classificati della classifica.
 - In genere k è 4.
- Nomi del meccanismo della graduatoria e della scelta casuale?
 - Tit-for-Tat.
- Quali sono le caratteristiche innovative di bit torrent nella condivisione di file?
 - Il fatto che il file è diviso in chunks e i peers si scambiano i chunks e non il file intero.
 - I chunks poi verranno ricostruiti in ogni leecher.

Video Streaming

Un video è una sequenza di immagini visualizzate con un certo rate → $Rate_{img} = n_immagini / s$.

Ognuna di queste immagini può essere vista come un array di pixel.

I pixel sono dei quadrati a cui si può associare un'informazione di luminosità e colore.

Queste due informazioni sono codificate con un certo numero di bit → $n_{BitPixel}$.

$$bitrate = (n_{BitPixel} * Rate_{img})$$

Questo bitrate, come presentato ora, uscirebbe molto elevato.

Tuttavia, possiamo sfruttare due proprietà interessanti:

- **Ridondanza Spaziale**
 - All'interno di un unico frame, si hanno (spesso) alcuni pixel uguali, e quindi si potrebbe codificare il primo pixel, e poi indicare in qualche modo che gli altri pixel vicini sono uguali;
- **Ridondanza Temporale**

- Confrontando due frame consecutivi, molti pixel non variano, e quindi invece di codificare interamente il frame successivo, basta solo indicare quali sono i pixel che variano.

Il risultato è che la quantità di bit trasmessa diminuisce.

Un contenuto può essere codificato in 2 modi:

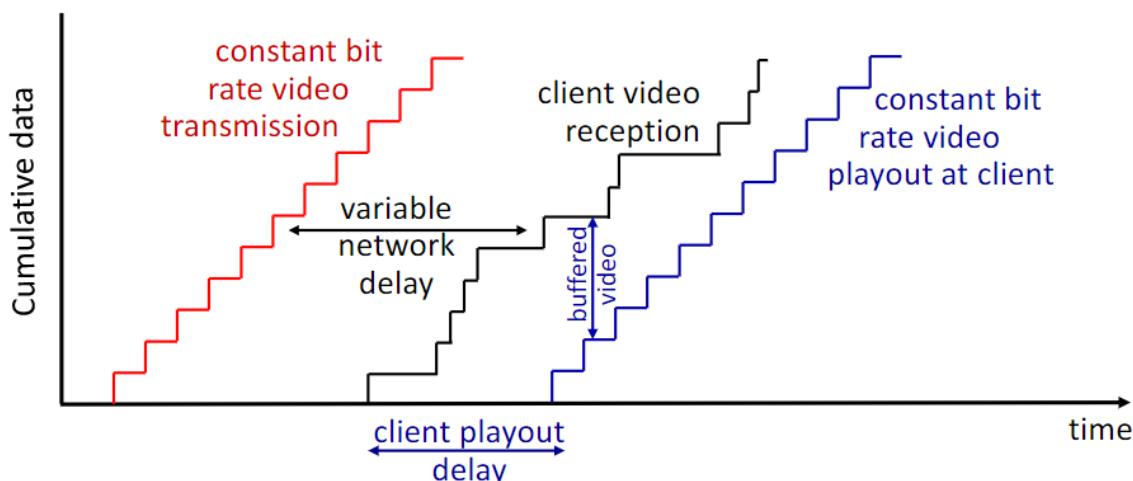
- **Constant Bit Rate (CBR):**
 - *bitrate* costante.
- **Variable Bit Rate (VBR):**
 - *bitrate* variabile, sfruttando le due ridondanze specificate prima.

Detto questo, un'applicazione di streaming consiste in un video che si trova in un server. L'utente lo scarica da tale server e nel frattempo può riprodurlo.

Streaming Live

Il video viene scaricato man mano che viene registrato e memorizzato nel server → Non lo facciamo.

L'utente inizia a scaricare, le informazioni scaricate si accumulano in un buffer di entrata. L'utente riproduce i frame contenuti nel buffer (si aspetta prima di accumulare un po' di contenuto nel buffer, perché la banda può variare, e può succedere che per un po' di tempo non è stato scaricato nulla dal server streaming , quindi ci si crea una scorta di frame per non interrompere il servizio).



Client Playout Delay & Buffering

Quindi il video ha un certo numero di frame/s , i frame però non vengono mandati ad una velocità costante.

Abbiamo quindi un ritardo chiamato “*Client Playout Delay*”.

Abbiamo il ritardo dovuto al client che accumula dei frame , compensa il “*Variable Network Delay*”, anche detto “*Jitter*” o in italiano comune “*Latenza*” , ossia un ritardo dovuto alla non stabilità dell'arrivo dei contenuti.

Ecco che allora si ha un delay iniziale in cui viene prima riempito il buffer, così che il video poi venga riprodotto allo stesso numero di frame al secondo, anche se il server gli invia più o meno frame al secondo (ci avvantaggiamo in modo da poter resistere in caso di calo di velocità).

Se il buffer si svuota e se la velocità di invio del server è inferiore a quella del video si ha il fenomeno del buffering.

Dynamic Adaptive Streaming over HTTP (DASH)

Recentemente, si è introdotta la tecnica DASH, ossia una tecnica di streaming adattivo.

Il server divide il file in vari chunk di una certa dimensione, e ogni chunk viene codificato svariati in bitrate diversi (e quindi a qualità diverse).

Quindi il server, per ogni chunk, ha molteplici codifiche.

I chunk verranno inviati al client tramite protocollo HTTP.

Si ha poi un *manifest file* (MPD) dove viene specificato, per ogni chunk:

- URL.
- Risoluzione Video.
- Velocità in Bit.

Il client periodicamente misura il throughput attuale tra lui e il server.

Il client consulta il Manifesto, seleziona la codifica migliore in quel momento (in modo tale che il buffer non si svuoti) e richiede il chunk.

Questa cosa è fatta prima di richiedere ogni chunk.

Questa tecnica permette di adattarsi a condizioni variabili della rete e diminuire fenomeni di buffering.

(es. basti pensare alla classica situazione quando si vede un video, ad un certo punto si vede bene, poi si vede peggio, poi di nuovo bene e così via).

Organizzazione del Servizio

La soluzione è mantenere più copie dello stesso video su vari server geograficamente distribuiti.

Questi server vengono messi il più vicino possibile agli utenti.

La soluzione dell'unico server è inaccettabile, non riuscirebbe a soddisfare tutte le richieste e rappresenterebbe un single point of failure, se crasha il server il servizio è interrotto.

La soluzione richiede molti server e distribuiti geograficamente.

CDN - Content Distribution Network

I server che distribuiscono contenuti multimediali vengono inseriti in reti chiamate CDN (Content Distribution Network).

In particolare, si hanno due approcci:

- **Approccio Enter Deep:**
 - Molti Server CDN dentro molte reti di accesso;
 - Vicino agli utenti.
 - Usato da Akamai.
- **Approccio Bring Home:**
 - Pochi Server CDN e molto grandi, ma vicino (e non dentro) alle reti di accesso

Es. client chiede a server Netflix dove ottenere video, server netflix gli dice dove, client sceglie server più vicino (può scegliere un altro se es. il link per tale server fosse congestionato).

Indirizzi IP Dinamici

Cambiano nel tempo , perché sono pochi ed è giusto riciclarli , se una persona non è attiva gli si toglie.

Server DHCP - Dynamic Host Configuration Protocol

Gli host nella rete di casa mia sono configurati automaticamente grazie al protocollo DHCP.
Quale dispositivo fa questa cosa? Il router, lì dentro c'è anche un processo Server DHCP.

Il DHCP dà agli host i 4 parametri per la configurazione necessari a navigare in internet.

- Indirizzo IP.
- Subnet Mask.
- Default Gateway.
- Indirizzo IP del Server DNS.

Un client si può connettere in DHCP solo se ha l'opzione per farlo, altrimenti si deve configurare in modo manuale tramite il file di configurazione.

Comunicazione DHCP

In genere, tutti gli step del DHCP avvengono in broadcast (dipendentemente dal server DHCP , a volte avviene anche in unicast).

Il messaggio va in broadcast perché l'host non sa chi c'è nella rete e neanche com'è fatta.

Il protocollo DHCP si compone di 4 fasi:

1. Discover.

- L'host client invia un pacchetto di tipo *DHCPDISCOVER* con
 - Indirizzo IP = 0.0.0.0 , Porta = 68 e con dst = 255.255.255.255 (*Indirizzo di Broadcast*).
 - 0.0.0.0 → “Non sono nessuno”.
 - 255.255.255.255 → “A chiunque mi senta”
 - 68 → Porta DHCP → “Mi serve una configurazione da un Server DHCP”.

2. Offer.

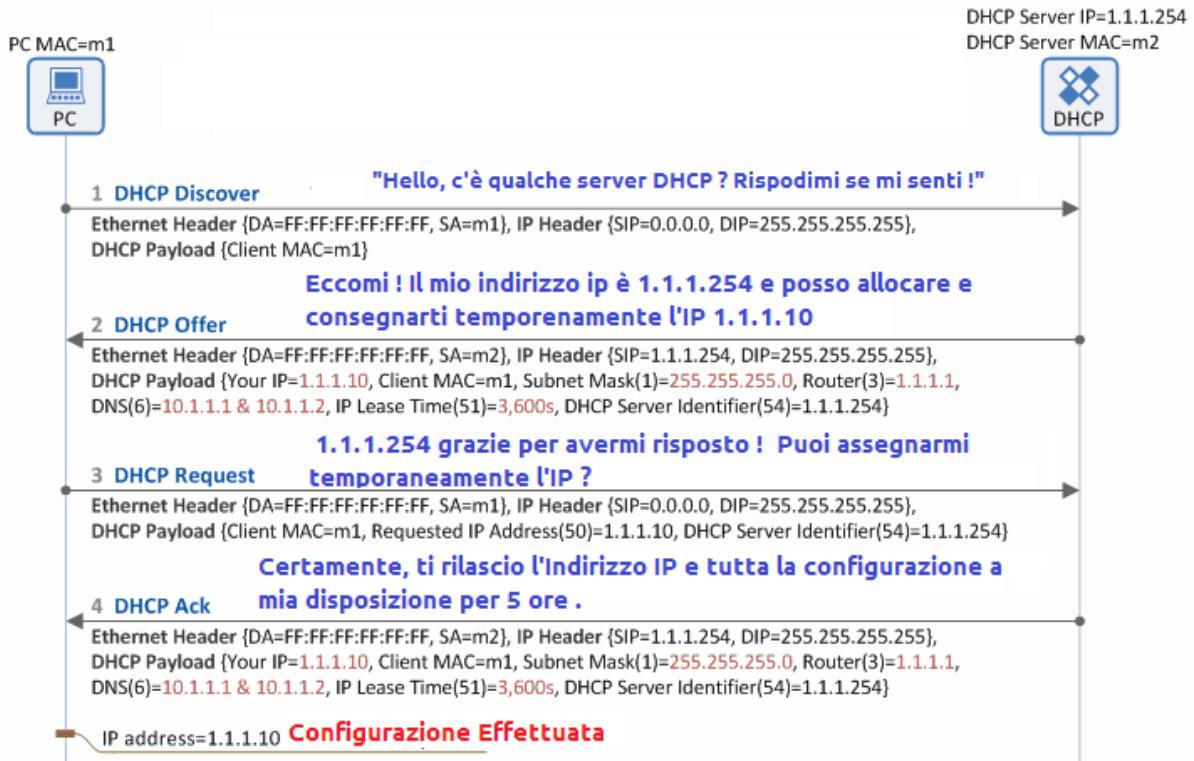
- Il Server DHCP risponde con un pacchetto di tipo *DHCPOFFER* contenente i 4 parametri:
 - src : 192.168.1.100 , porta = 67

3. Request.

- Il client accetta l'offerta inviando al server un messaggio di tipo *DHCPREQUEST*

4. Acknowledge.

- Il DHCP risponde con un messaggio di tipo *DHCPACK*.
- L'offerta ha una scadenza, se quell host non è più connesso dobbiamo riciclare le configurazioni..



Quando nella rete è presente un server DHCP non è detto che tutti gli host siano configurati in DHCP.

Ad esempio, ci sono macchine nella rete che potrebbero volere un indirizzo IP fisso (ad esempio un server web o un server DNS locale).

Il server DHCP nella rete locale ha un range di indirizzi IP assegnabili, gli indirizzi IP che sono stati assegnati staticamente devono essere tolti da quel range di indirizzi, in modo tale che il Server DHCP non li prenda mai in considerazione nelle offerte.

Se un indirizzo è assegnato dal DHCP non può essere assegnato staticamente e viceversa.

Quindi in genere ogni rete ha un range di indirizzi assegnabili staticamente e un range di indirizzi assegnabili dinamicamente.

In una rete ci possono essere molteplici server DHCP, ognuno che presenterà la sua offerta, il client ovviamente invierà il DHCPREQUEST solo a uno di loro.

- schema di come un host richiede e riceve la configurazione.
 - La comunicazione avviene in 4 fasi:
 - i. L'host in broadcast esegue una richiesta DHCP.
 - ii. I Server DHCP in grado di sentire la richiesta inviano in broadcast una DHCP offer, ossia una offerta.
 - Nella offer c'è l'indirizzo IP del server DHCP.
 - iii. L'host richiedente sceglie una delle offerte e invia in broadcast.

- Specificando in un campo apposito chiamato DHCP Server Identifier (non il destination address che rimane 255.255.255.255) l'indirizzo IP del server DHCP relativo all'offerta accettata.
- iv. Il Server DHCP coinvolto allora conferma la richiesta con il DHCP ACK inviando di nuovo la configurazione e assegnando un TTL..
- come avviene la comunicazione
 - La comunicazione avviene tutta in broadcast, perché l'host richiedente non ha nessuno dei 4 parametri per navigare.
- livello in cui opera tale protocollo
 - DHCP è una applicazione e risiede al livello 7 della pila protocollare.
 - DHCP usa UDP come protocollo di trasporto.

Reti a Connessione Diretta

Gli host della rete sono tutti connessi direttamente.

Gli Host e i Routers sono detti "Nodi".

I canali di comunicazione che connettono i nodi tra loro sono detti "Links", i quali possono essere wired (cavi in rame o in fibra ottica) o wireless(l'etere).

Livello Fisico della Comunicazione

Le informazioni per prima cosa vanno **codificate**, in particolare noi vogliamo inviare sequenze di bit.

Le nostre informazioni sono composte da solo 0 o 1.

Per inviare i bit usiamo un segnale elettrico caratterizzato da un certo valore di tensione , sopra una certa soglia è 1 logico altrimenti 0 logico.

Il segnale elettrico che transita sul canale è un segnale analogico.

Ogni informazione è codificabile in una sequenza di bit, data una sequenza di bit di lunghezza variabile, per ogni bit devo generare una certa quantità di tensione e farlo propagare sul link (supponiamo di natura elettrica come un cavo in rame), arrivato a destinazione occorre fare una decodifica del segnale elettrico ricevuto.

Il livello fisico implementa:

- Codifica e Decodifica del segnale.
- Ricezione e Trasmissione del segnale.

Il link di comunicazione **non è ideale**:

- Il segnale si attenua con il crescere della lunghezza del link, quindi ogni tanto occorre rigenerarlo.
 - Il link è soggetto a disturbi che possono corrompere il segnale (potrebbero far apparire uno 0 logico come un 1 logico "Bit Flippato").
- Ad esempio se due link di comunicazione sono (geograficamente) molto vicini potrebbero disturbarsi a vicenda.

Il link per sua natura è e rimarrà sempre inaffidabile, ovviamente ci sono link più affidabili di altri.

Bit Error Rate : “N° di bit errati su 100 bit inviati”.

- Fibre ottiche → 10^{-9}
- Cavo in Rame → 10^{-6}
- Collegamenti Wireless → 10^{-3}

Livello Data Link

Ha l'obiettivo di rendere la comunicazione affidabile su un link inaffidabile.

Framing

Invece di inviare tutta la sequenza per intero, conviene suddividere la sequenza in trame di lunghezza fissa e inviare le trame una alla volta.

Le trame sono molto più piccole del messaggio originale e le probabilità che un frame sia inviato senza errori aumenta.

Se un frame arriva con un errore basta inviare di nuovo solo il frame danneggiato e non tutta la sequenza.

Il ricevitore poi dovrà ricostruire il file originale.



Funzionalità del livello Data Link

- *Error Detection*
 - Rileva la presenza di errori in un frame.
- *Error Correction*
 - Rileva la presenza di errori in un frame e li Corregge.
- *Retransmission*
 - Il ricevitore segnala i frame non validi e se li fa rimandare.
- *Flow Control*
 - Il Sender fa in modo che la velocità di trasmissione si adatti alla velocità di ricezione del receiver per evitare effetti di collo di bottiglia.
- *Half-Duplex & Full-Duplex*
 - Con half-duplex il link può essere usato solo da un senso.
 - Con full duplex la comunicazione è bidirezionale.

Scheda di Rete - NIC - Network Interface Card

E' una periferica, quindi comunica con la CPU come tutte le altre.

Il livello data-link è in parte implementato in hardware nella scheda di rete e in parte in software.

Al livello data-link ricevo i datagram dal livello sopra (livello network).

Li incapsulo in frames, quindi aggiungo alcune informazioni nell'header e nel trailer.

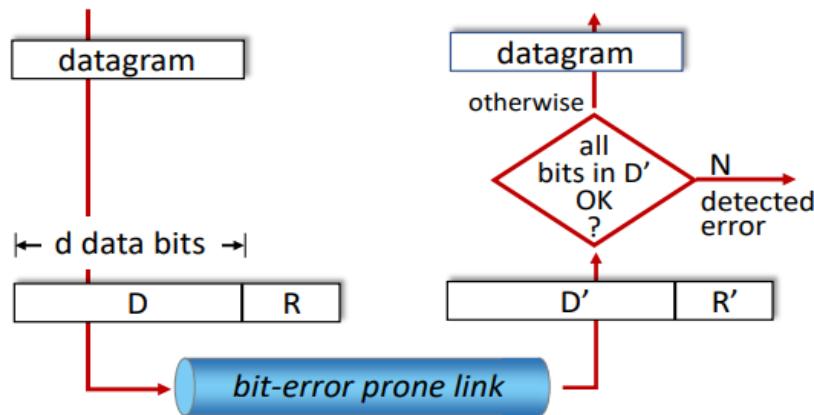
Error Detection

Attuata nel Ricevitore.

Ha lo scopo di dire se il frame appena ricevuto contiene errori o no.

Il controllo viene fatto attraverso dei bit speciali chiamati “*Bits di Ridondanza*”, calcolati mediante un algoritmo preciso.

Durante l’incapsulamento nel livello data link, oltre al blocco di dati aggiungo dei bit R di ridondanza, calcolati secondo un certo algoritmo.



La error detection non è mai affidabile al 100%, ci possono essere dei falsi negativi (ossia una sequenza sembra corretta ma non lo è).

Ovviamente l'affidabilità dell'error detection dipende fortemente dal tipo di algoritmo usato per calcolare R.

1. Il Sender riceve un datagram dal livello superiore e lo incapsula in un frame.
2. Il Sender calcola e aggiunge R secondo un certo algoritmo.
3. Il Sender invia $\langle D, R \rangle$.
 - a. Attraverso il Link Inaffidabile: $\langle D, R \rangle \rightarrow \langle D', R' \rangle$
4. Il Ricevitore riceverà $\langle D', R' \rangle$.
5. Il Ricevitore eseguirà lo stesso algoritmo per calcolare R su D'.
6. Da D' ottiene F, se $F = R' \rightarrow$ Non ho subito errori.

Error Detection - Falso Negativo

La grandezza della probabilità che avvenga questo evento è inversamente proporzionale al numero di bit che compongono R.

Il falso negativo accade quando i bit di D' e R' filippano in modo tale che risulti $F = R'$, la sequenza appare come perfettamente sana.

Tecnica di Error Detection - Bit di Parità

R è composto da 1 bit.

Questa tecnica può essere implementata in 2 modi:

- **Parità Pari :**
 - Si aggiunge 1 in modo che il numero complessivo di 1 in $\langle D, R \rangle$ sia pari.
- **Parità Dispari:**
 - Si aggiunge 1 in modo che il numero complessivo di 1 in $\langle D, R \rangle$ sia dispari.

Affidabilità - Quando avviene il falso negativo nel Bit di Parità

Se il numero di bit che “filippano” è pari → l’errore non viene rilevato.

Questo metodo è usato quando la probabilità di errore è molto bassa.

Il vantaggio è che calcolare R è molto facile e la dimensione del frame aumenta di pochissimo.

Tecnica di Error Detection - Internet Checksum

Usato nel livello di trasporto.

R è composto da 16 bit.

1. Il Sender interpreta il pacchetto come una sequenza di interi a 16 bit
2. Il Sender fa la somma tra tutti i blocchi su 16 bit.
 - a. Il risultato è detto "checksum" ed esso compone R.
3. Il ricevitore dall'altro lato esegue lo stesso algoritmo sui blocchi ricevuti.
 - a. Se la somma coincide → No Errori.

Affidabilità - Quando avviene il falso negativo nell'internet Checksum

Ci sono possibilità che flippano certi bit la somma non cambi, di sicuro più affidabile del bit di parità.

Tecnica di Error Detection - Cyclic Redundancy Check - CRC

Usato nel livello data-link, in Ethernet e 802.11 Wi-Fi.

Sia D l'insieme dei dati (supponiamo che siano d bit) che dobbiamo inviare.
Interpretiamo i bit di D come un numero intero.

Sia $G > 0$, detto "generatore" composto da ($r + 1$) bit, noto a entrambe le parti.

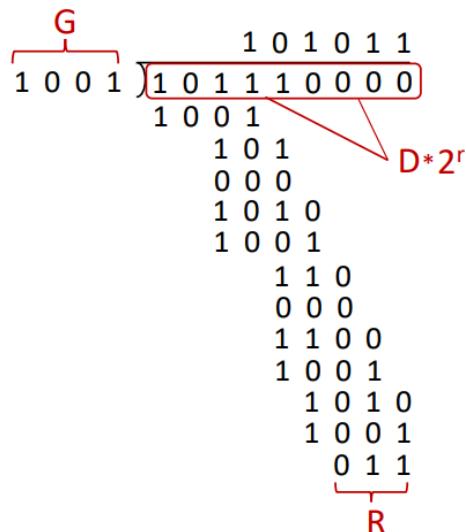
$$\langle D, R \rangle = [(D * 2^r) \oplus R] \rightarrow "D \text{ Concatenato a } R"$$

CRC - Generazione di R

Secondo CRC devo scegliere R in modo che $\langle D, R \rangle$ sia divisibile per G in modulo 2.

Quindi trovare R tale che $\exists n$ intero per cui $D * 2^r \oplus R = n * G$.

$$\text{Conoscendo } D \text{ e } G \rightarrow R = \text{Resto} \left[\frac{D * 2^r}{G} \right] = |D * 2^r|_G$$



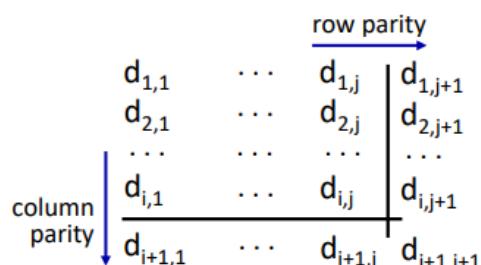
Spesso CRC è implementato in Hardware, quindi risulta molto veloce.

Tecnica di Error Correction & Detection - Bit di Parità Bidimensionale

Disponiamo di bit di D come una matrice di dimensione ($i \times j$).

Per ogni riga e per ogni colonna della matrice calcolo un bit di parità (pari o dispari) unidimensionale.

Poi calcolo il bit di parità anche alla colonna e riga di ridondanza.



Infine aggiungo l'ultimo bit di parità calcolato basandosi sulla colonna di parità e sulla riga di parità ossia il bit $d_{i+1,j+1}$.

Quindi alla fine la matrice finale (ossia $\langle D, R \rangle$) sarà di dimensione $(i+1) \times (j+1)$.

R è composto da ($i + j + 1$) bit di parità.

1. Prima si controllano i bit di parità delle righe e poi quelle delle colonne.
2. Se fallisce il test della riga h allora almeno uno dei bit di quella riga ha subito errori.
3. Se fallisce il test della colonna z allora almeno uno dei bit di quella colonna ha subito errori.
4. Se ha fallito il test della riga h e della colonna z \rightarrow il bit $d_{h,z}$ è errato.

no errors: 1 0 1 0 1 1 1 1 1 1 0 0 0 1 1 1 0 1 <hr/> 1 0 1 0 1 0	detected and correctable single-bit error: 1 0 1 0 1 1 1 0 1 1 0 0 → parity error 0 1 1 1 0 1 <hr/> 1 0 1 0 1 0 ↓ parity error
--	--

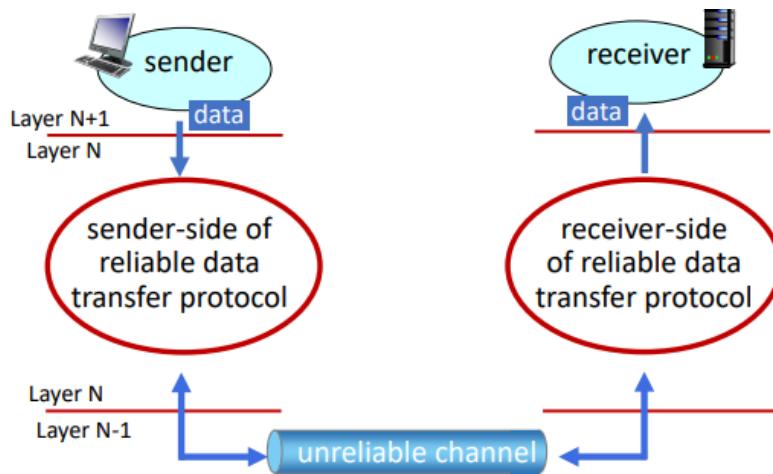
Questo metodo è molto efficace (ma non è affidabile al 100%) ma anche molto costoso, soprattutto quando gli errori sono tanti.

Questa tecnica si usa in contesti dove il *BitErrorRate* è molto elevato.

Reliable Data Transfer - RDT

RDT è un set di algoritmi con lo scopo di fornire garanzie del trasferimento affidabile dei dati attraverso una rete che può essere soggetta a perdita e/o danneggiamento dei dati.

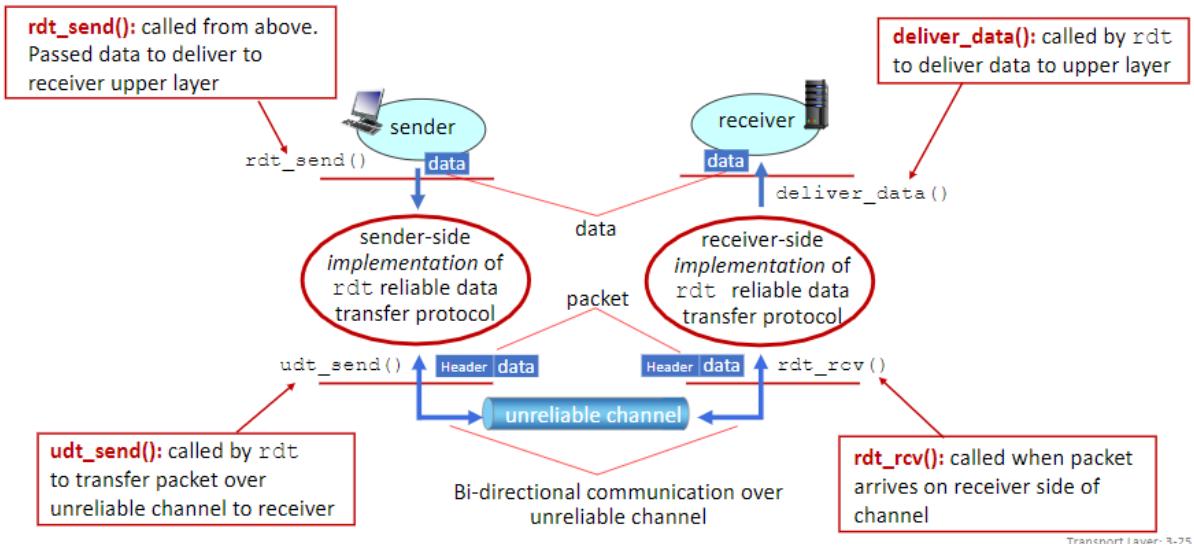
Mettiamo sopra il canale inaffidabile delle funzionalità che permettono di rendere il canale affidabile.



Quando il sender deve inviare qualcosa usa la funzione `rdt_send()` per inviare alla parte sender dell'implementazione del protocollo rdt.

Il ricevitore pensa che il canale è affidabile e che i dati sono stati inviati direttamente dal sender, senza passare da nessun livello intermedio.

Tutti i controlli di affidabilità sono trasparenti al sender e al receiver.



Macchina a Stati Finiti per definire un Protocollo

Per definire il protocollo di trasmissione useremo un formalismo → “la macchina a stati finiti”.

- Quando si verifica un evento si può generare una transizione.
- Quando si verifica una transizione si possono eseguire certe azioni.

Quando si definisce un protocollo si deve fare:

1. Definire tutti gli eventi possibili
2. Definire in quali condizioni si verifica una transizione e quando no.

1° Versione

Supponiamo che il Canale sia totalmente affidabile , quindi niente errori o perdite di pacchetto , tutto ciò che è inviato arriva a destinazione senza problemi.

Il sender invia dati al canale sottostante e il ricevitore riceve dati dal canale sottostante.

Eventi Lato Sender:

- Il lato superiore invia al sender un pacchetto da spedire usando `rdt_send(data)`.
 - a. Il sender prepara il frame , tramite la funzione `make_pkt(data)`.
 - Il trasferimento è affidabile → non servono informazioni di error detection.
 - b. il sender invoca la `udt_send(packet)` e spedisce il frame al livello fisico.
 - c. Ritorna in stato di attesa.

Eventi Lato Receiver:

- Il Receiver invoca `rdt_receive(packet)` per prendere il frame.
 - a. Il receiver invoca `extract(packet)` per estrarre i dati.
 - b. Il Receiver invoca `deliver_data(data)` per spedire i dati al livello superiore.
 - c. Ritorno in stato di attesa



2° Versione

Il Canale è parzialmente inaffidabile, ossia i bit possono flippare, ma non si ha packet loss.

Introduciamo 2 tipologie di messaggi:

- *acknowledgement* → ACKs “Il receiver dice al sender che è tutto ok”.
- *negative acknowledgement* → NAKs “Il Receiver dice al sender che il pacchetto è corrotto”

Il sender oltre ai dati incapsula anche le informazioni del CRC.

Quando il sender invia con `udt_send()` il pacchetto passa al prossimo stato, dove aspetta ACK o NACK dal receiver.

- Se riceve ACK, si ritorna al 1° stato.
- Se riceve NACK, si rimanda il frame precedente eseguendo di nuovo la `udt_send()`.

Ma se il NACK o ACK arrivasse corrotto?

Nessuno mi garantisce che arrivino sempre corretti, quindi il protocollo così com'è non va bene.

Gestione Duplicati

Inoltre cosa accadrebbe se venissero trasmessi più volte i ACK/NACK che si erano precedentemente corrotti? Ossia, come posso riconoscere dei duplicati?

Per riconoscere i duplicati basta mettere un **numero di sequenza** ai pacchetti, in modo tale che il receiver possa capire quali pacchetti ha già ricevuto e quali invece no.

Ogni volta che si invia un pacchetto si incrementa in maniera circolare una variabile contatore, essa fungerà da numero di sequenza.

La gestione dei duplicati (successiva alla ricezione) dipende poi dalla implementazione.

Una possibile implementazione potrebbe essere: se arriva un pacchetto duplicato si invia ACK (almeno il sender non lo invierà di nuovo) per quel pacchetto ma poi si scarta.

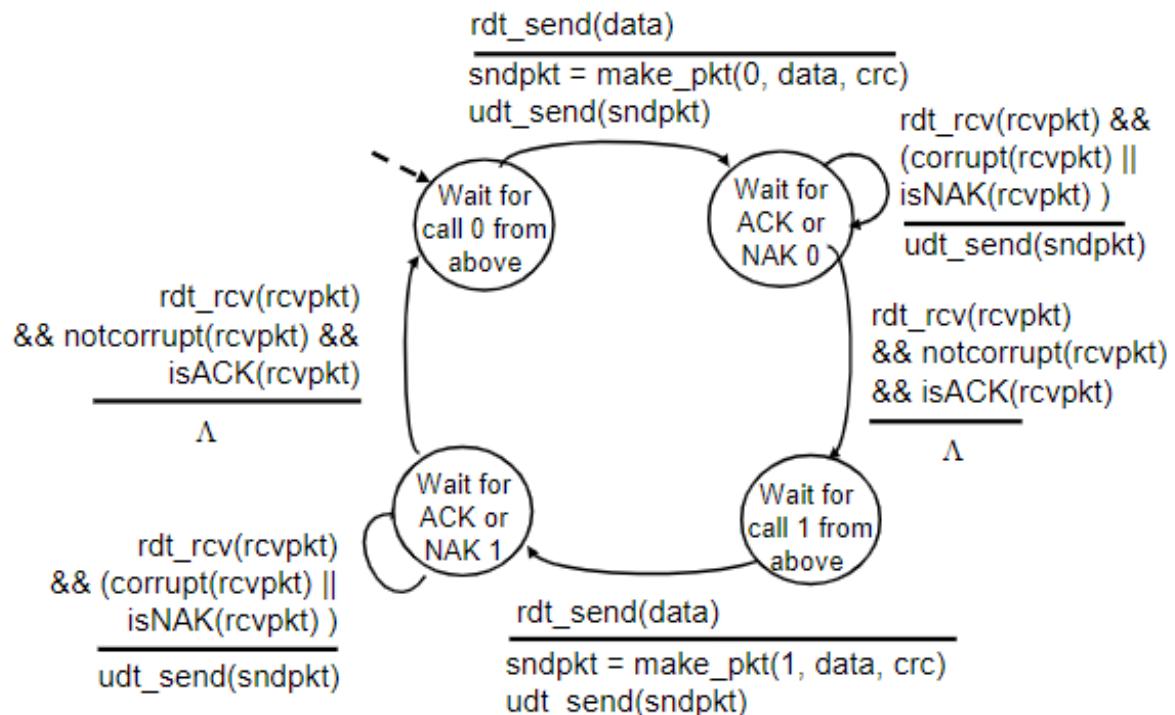
Il receiver invierà conferma al sender con un pacchetto chiamato “rcvpckt”.

Quindi il sender dopo aver inviato un pacchetto richiamerà `rdt_rcv()` per aspettare il rcvpkt.

⚠ → “Non fa Nulla”

Nell'esempio si è posto 1 bit per il numero di sequenza.

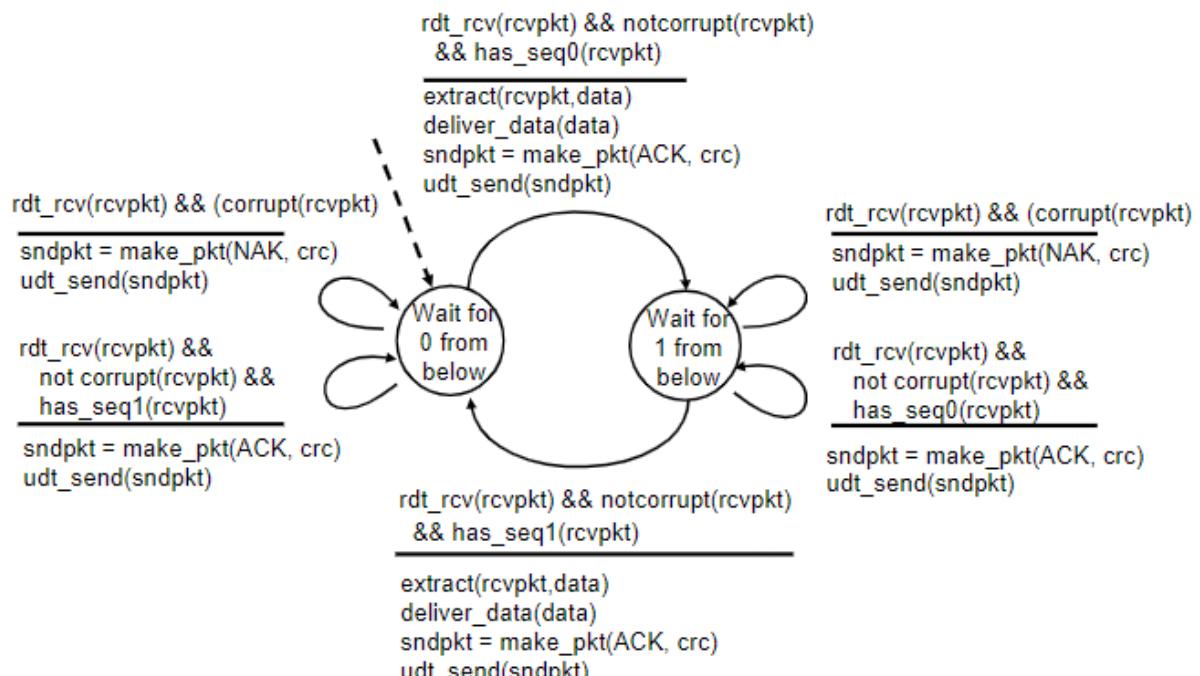
Sender:



Il Sender aspetta di ricevere un NACK o un ACK.

Se il pacchetto che contiene uno dei due è corrotto oppure se è NACK, invia il pacchetto inviato allo stato precedente.

Receiver:



Lato sender ho aggiunto i numeri di sequenza , con un numero di sequenza pari a 0 o 1 , basta un bit.

Si deve controllare anche se ACKs o NACKs sono corrotti.

Occorre ricordarsi il numero di sequenza.

Lato receiver deve capire quali pacchetti sono doppioni , lo fa tramite has_seq0 e has_seq1 , dove controlla i numeri di sequenza dei pacchetti.

Protocollo Free-NACKs

Invece di mandare il NACKs relativo al numero di sequenza attuale invio ACK con il numero di sequenza precedente.

3° Versione

Canale inaffidabile: *Si ha corruzione e perdita di pacchetto.*

Il sender dopo aver inviato il pacchetto aspetta una certa quantità di tempo(*timeout*), oltre al quale si considerano persi.

Se la ricezione ha semplicemente avuto un ritardo tale da far scadere il timeout , il sender invierà lo stesso pacchetto , a quel punto il receiver riceverà un doppione.

Poco male , il receiver può gestire i doppieni grazie al controllo eseguito sul numero di sequenza del pacchetto implementato nella 2° versione.

Durata del Timeout

Il timeout non può avere una durata causale, ma deve essere impostato in modo tale che sia possibile non violarlo.

Round Trip Time = RTT = (Tempo di Andata + Tempo di Ritorno).

Quindi io so che il tempo stimato che passa dall'invio del pacchetto all'arrivo della risposta è RTT.

Timeout > RTT

Altrimenti scatterebbe sempre, perché il pacchetto non ha il tempo materiale di arrivare.

Tempo di Andata.

Supponiamo che il pacchetto sia lungo L bit.

Supponiamo che il canale di comunicazione abbia un bitrate pari a R [bit / sec].

Ci vorranno $\frac{L}{R}$ secondi per trasmettere il pacchetto.

$$A questo sommo il tempo di propagazione \rightarrow T_{prop} = \frac{\text{Lunghezza Canale}}{\text{Velocità Propagazione}}$$

$$T_{andata} = \frac{L}{R} + T_{prop}$$

Tempo di Ritorno

Sia L_{ack} la lunghezza del pacchetto di ACK.

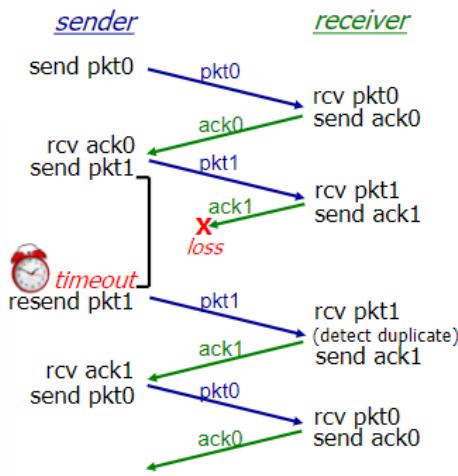
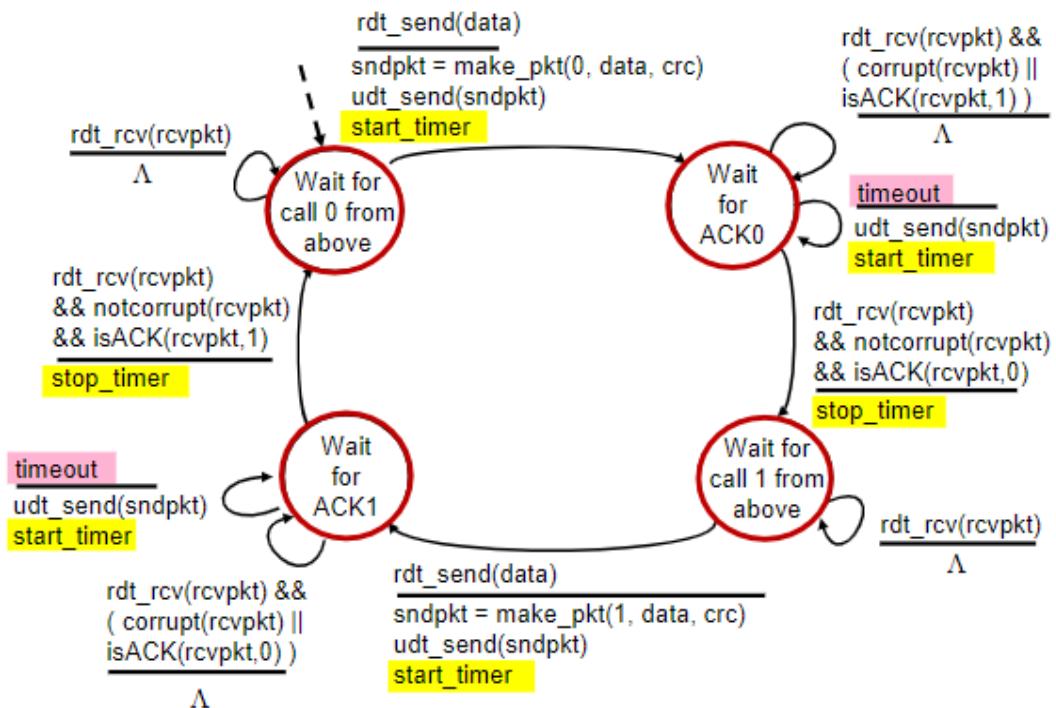
Ci vorranno $\frac{L_{ack}}{R}$ secondi per trasmettere il pacchetto di ACK.

A questo sommo il *tempo di propagazione* $\rightarrow T_{prop}$.

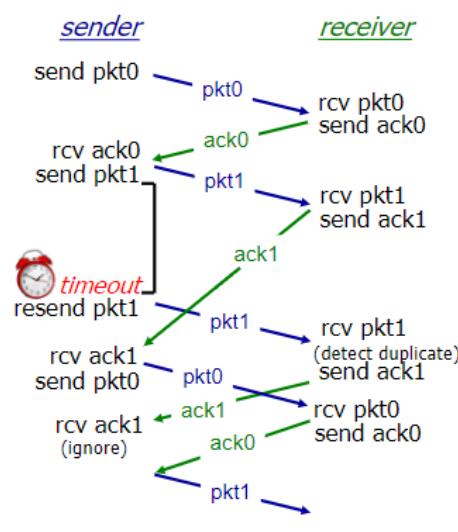
$$T_{ritorno} = \frac{L_{ack}}{R} + T_{prop}$$

$$RTT = \left(\frac{L}{R} + \frac{L_{ack}}{R} + 2 * T_{prop} \right)$$

Sender



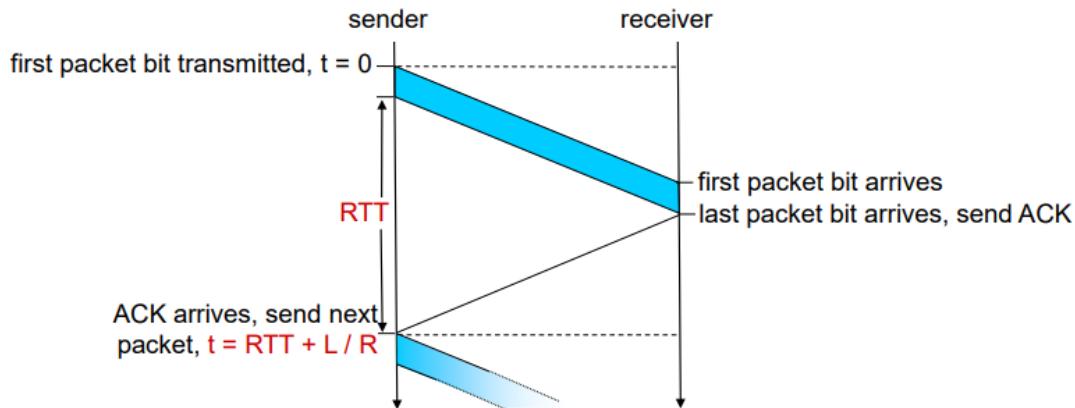
(c) ACK loss



(d) premature timeout/ delayed ACK

Performance della 3° versione (stop-and-wait)

- $U_{sender} \rightarrow \text{"Utilization"}$, frazione di tempo in cui il sender è in busy sending. Ossia in cui il sender sta effettivamente facendo qualcosa.
- Supponiamo di avere un link con un bitrate pari a 1 Gbps , quindi $R = 10^9 \text{ bit / sec}$.
- Supponiamo un Tempo di Propagazione pari a 15 millisecondi → $T_{prop} = 15 * 10^{-3} \text{ sec}$.
- Sia il pacchetto P , con L = 8000 bit.
 - Tempo per Trasmettere P → $\frac{L}{R} = \frac{8000 \text{ bit}}{10^9 \text{ bits/sec}} = 8 * 10^{-6} \text{ sec}$.
 - $U_{sender} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{0.008}{30.008} = 0.00027\%$
 - Questo protocollo ha U_{sender} bassissima, fa abbastanza schifo.



Tecnica di Pipelining

Voglio aumentare l'utilizzo del sender.

Usata nel TCP.

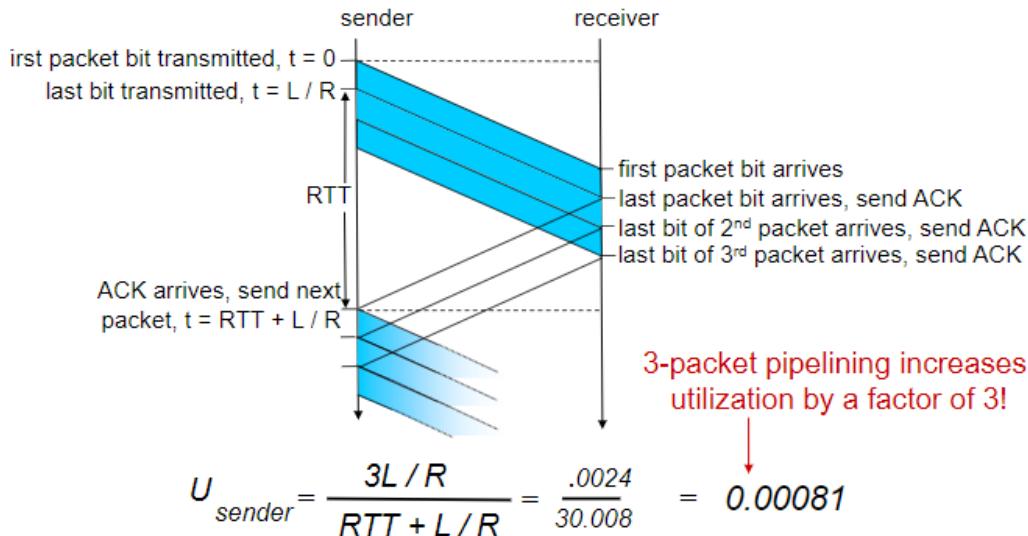
Mando tutti pacchetti uno dopo l'altro, poi aspetto di ricevere tutti i loro ACKs.

Le prestazioni generali e soprattutto U_Sender migliora :

Perché diminuisco il tempo di "idling", ossia i momenti in cui il sender non fa nulla ma aspetta solo il receiver.

Perché tra l'invio del pacchetto e l'arrivo del relativo ACK passa RTT.

Quindi se li mando tutti uno dopo l'altro, dopo RTT arriveranno in sequenza i relativi ACK.



Packet Loss e Error Recovery:

Il ricevitore deve decidere quando mandare l'ACK :

- Lo mando appena ricevo il pacchetto oppure mando tutti gli ack in maniera cumulativa (un ACKs per tutti , ossia → “Ho ricevuto fino al pacchetto numero K”)?
- Se mi arrivano fuori ordine (se mi arrivano il 3 e il 5, butto il 5 o aspetto il 4)?

Nella pipeline posso usare 2 tecniche :

- Go-Back-N.
- Selective Repeat.

Finestra

N → “Finestra”, esprime quanti pacchetti senza ACK può mandare il sender, ossia il numero di pacchetti “in-flight”(e quindi quanti pacchetti possono essere tenuti nel buffer del trasmettitore).

Tecnica di Pipelining - “Go-Back-N”

- Sender:
 - Può mandare al massimo N pacchetti senza relativo ACK nella pipeline.
- Receiver :
 - Manda solo ACKs cumulativi (“Ho ricevuto tutti i pacchetti fino al numero K”), non manda un ACK se c’è un pacchetto mancante (*gap*).
- Sender:
 - Ha un timer per i pacchetti in flight, se il timer scade rimando tutti i pacchetti con numero di sequenza > k.

Finestra del Sender:

- k-bit seq # in pkt header



Il sender aggiorna `send_base` con il valore di `rcv_base` che riceve dagli ACKs cumulativi.

Finestra del Receiver:

Receiver view of sequence number space:



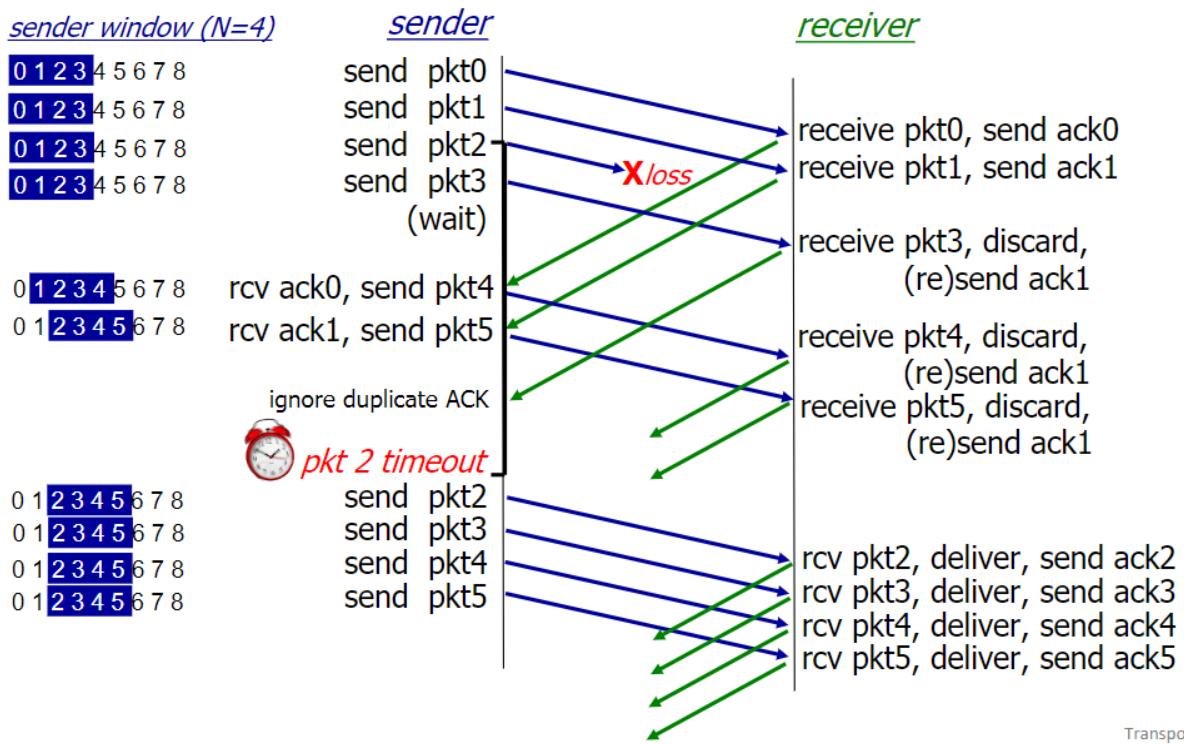
Il receiver ha ricevuto fino al numero k (`rcv_base`), allora si aspetta il $(k+1)$, quando arriva un pacchetto confronta il numero di sequenza con $(k+1)$, se coincide manda ACK $(k+1)$, dove dice al sender che ha ricevuto tutti i pacchetti fino al numero $(k+1)$.

Se (fuori ordine) ha ricevuto anche $(k+2)$ e $(k+3)$, allora all'arrivo di $(k+1)$ il receiver manderà un ACK cumulativo dove dice al sender che ha ricevuto tutti i pacchetti fino al $(k+3)$ -esimo.

Ossia se il receiver ha ricevuto il pacchetto 3 e il pacchetto 5, invierà sempre ACK 3.

Quando riceverà il pacchetto 4, allora invierà ACK 4.

Il receiver deve solo ricordarsi `rcv_base`, il quale memorizza il numero di sequenza relativo al pacchetto con numero di sequenza minore ma che non è mai arrivato, quindi l'arrivo di pacchetti fuori ordine ($> k+1$) non causa un incremento di `rcv_base`.



Gestione Pacchetti Fuori Ordine - “Go-Back-N”

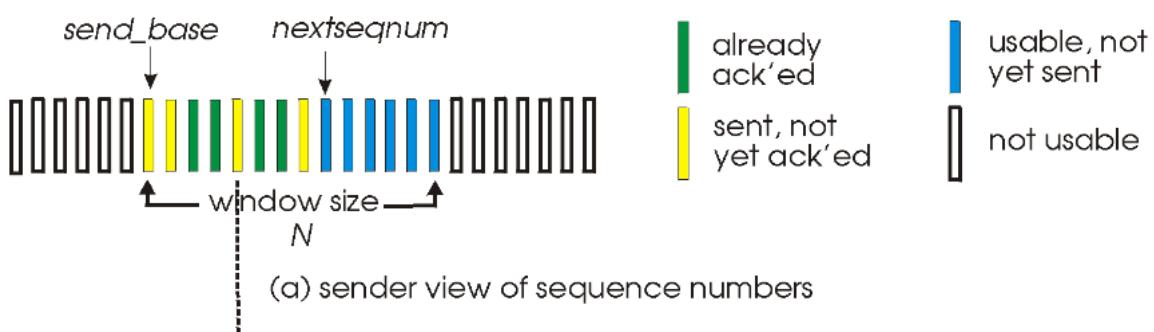
Il realtà questo è un comportamento non definito nel protocollo , dipende da chi lo implementa , un pacchetto fuori ordine può essere scartato o tenuto.

Tanto allo scadere del timer il sender rimanda tutti i pacchetti con numero di sequenza superiore a `send_base`.

Di solito i pacchetti fuori ordine si scartano.

Tecnica di Pipelining - “Selective Repeat”

- Sender:
 - Può mandare al massimo N pacchetti senza relativo ACK nella pipeline.
- Receiver:
 - Manda solo ACK individuali , ossia un pacchetto ACK per ogni pacchetto arrivato.
- Sender :
 - Ha un timer per ogni pacchetto inviato senza relativo ACK, se il timer di un pacchetto scade rimanda solo quel pacchetto..



`send_base` punta al pacchetto inviato ma senza relativo ACK di indice minore (il più a sinistra).

La finestra parte da `send_base` e avanza per altri N slot verso destra.

Selective Repeat - Timer dei Pacchetti senza ACK

Il sender deve tenere un timer per ogni pacchetto senza ACK (i timer sono tenuti in modo Software, non si usa il timer Hardware del dispositivo, poiché i timer hardware sono in numero molto limitato).

Quando il timer di un pacchetto scade : si spedisce solo il pacchetto relativo a quel timer.

In caso di pacchetti fuori ordine il receiver manda comunque un ACK, dicendo al sender che li ha ricevuti.

Eventi Sender:

- Data From Above :
 - a. Se è disponibile uno slot nella finestra, invio un pacchetto e incremento la finestra corrente.
- `timeout(n)` :
 - a. Rimando il pacchetto n
 - b. Reimpostare il timer del pacchetto n.
- $ACK(n)$: con $n \in [send_base, send_base + N]$
 - a. Segno n come ricevuto.
 - b. Se n è il più piccolo pacchetto senza ACK (ossia se è puntato da `send_base`)
 - Avanzo la finestra fino a che `send_base` punta al prossimo pacchetto senza ACK.

Eventi Receiver:

- Ricevo un Pacchetto con numero di sequenza n:
 - a. $n \in [rcv_base, rcv_base + N - 1]$
 - $ACK(n)$
 - Dove metto il pacchetto:
 1. *out-of-order* → Buffer.
 2. *in-order* → Deliver.
 - b. $n \in [rcv_base - N, rcv_base - 1]$
 - Fuori dalla finestra corrente, ossia un Duplicato o pacchetto vecchissimo.
 - Ignoro.

Caso senza Errore.

Lo spazio dei numeri di sequenza è {0, 1, 2, 3}

La finestra è $N = 3$.

1. Manda i pacchetti con n° 1, 2 e 3.
 2. Finestra piena, attende gli ACK
-
1. Il Receiver riceve il pacchetto numero 0
 2. Avanzo `rcv_base`, mando $ACK(0)$, aspetto il numero 1.
 3. Ricevo il pacchetto n° 1

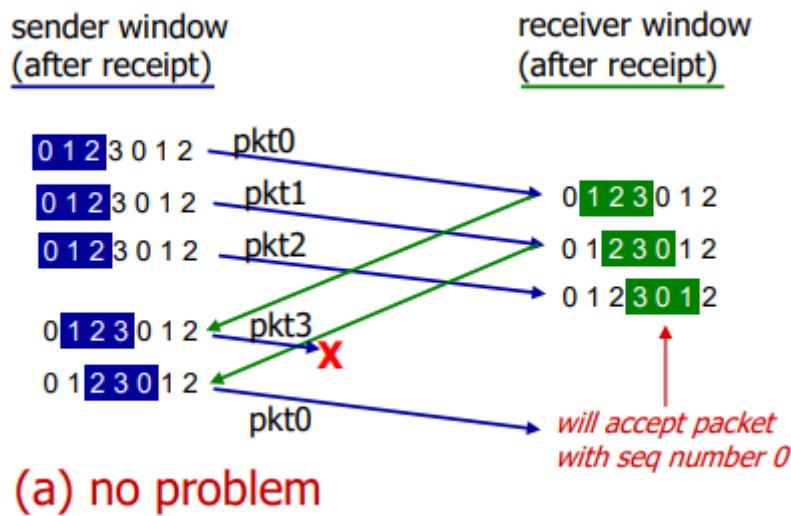
4. Avanzo `recv_base`, mando `ACK(1)`, aspetto il numero 2.
5. Ricevo il pacchetto n° 2
6. Avanzo `recv_base`, mando `ACK(2)`, aspetto il numero 3.

 1. Il sender riceve `ACK(0)`, avanza `send_base`.
 2. Si libera uno slot nella finestra, mando il pacchetto numero 3.

Il pacchetto numero 3 non arriverà mai

1. Il sender riceve `ACK(1)`, avanza `send_base`.
2. Si libera uno slot nella finestra, mando il pacchetto con numero di sequenza 0 (poiché è stato incrementato in modulo 4, quindi dopo il 3 c'è lo 0).

1. Il receiver riceve il pacchetto con numero di sequenza 0, dato che la finestra ha 3 slot, la finestra comprende {3, 0, 1}, quindi accetta il pacchetto.



Caso con Errore.

1. Il Sender Manda i pacchetti con n° 1, 2 e 3.
2. Finestra del sender piena, attende gli ACK dal receiver.

1. Il Receiver riceve il pacchetto n° 0.
2. Avanzo `recv_base`, mando `ACK(0)`, aspetto il numero 1.

ACK(0) non arriverà mai al sender.

3. Il Receiver riceve il pacchetto n° 1.
4. Avanzo `recv_base`, mando `ACK(1)`, aspetto il numero 2.

ACK(1) non arriverà mai al sender.

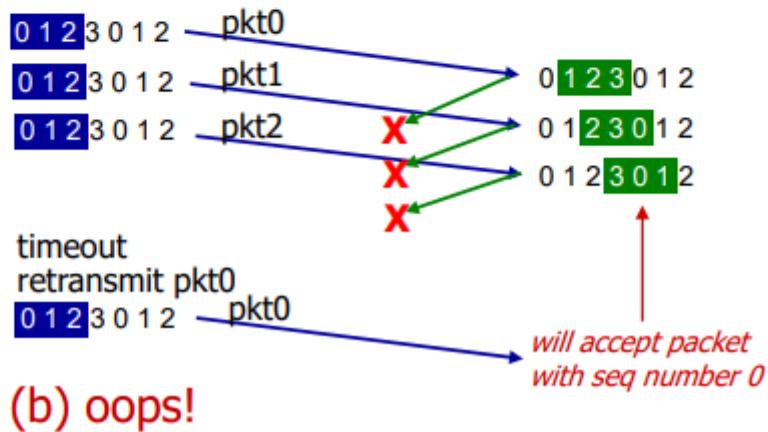
5. Il Receiver riceve il pacchetto n° 2.
6. Avanzo `recv_base`, mando `ACK(2)`, aspetto il numero 3.

ACK(2) non arriverà mai al sender.

1. Il sender non riceve `ACK(0)`.
2. Scade il timeout.

3. Il Sender manda di nuovo il pacchetto con numero di sequenza 0.

 1. Il receiver riceve il pacchetto con numero di sequenza 0 , si aspettava il numero 3 , ma comunque 0 è dentro la finestra corrente del receiver.
 2. Il Receiver conclude che il 3° si è perso.
 3. Il pacchetto 0 viene considerato come “nuovo” (ossia quello successivo al pacchetto 3) , ma in realtà è il **vecchio** pacchetto con numero di sequenza 0.



L'errore è che la finestra non ha la giusta dimensione in confronto allo spazio dei numeri di sequenza.

Lo spazio dei numeri di sequenza dovrebbe essere maggiore del doppio della dimensione della finestra.

Quindi se avessi usato almeno 3 bit per il numero di sequenza non ci sarebbero stati problemi.

Termine “Pacchetto”

Pacchetto è un termine generico , rappresenta un blocco di bit che viene inviato.

Il pacchetto a seconda del livello in cui siamo , gli diamo un nome particolare , per appunto individuare subito il contesto in cui ci troviamo:

Livello Applicazione → Pacchetto = Messaggio.

Livello Trasporto → Pacchetto = Segmento.

Livello IP → Pacchetto = Datagram.

Livello Data-Link → Pacchetto = Frame.

Livello Fisico → Segnale.

Livello Data-Link

Questo livello è il livello del protocollo che trasferisce i dati tra i nodi su un segmento di rete attraverso il livello fisico.

Il livello di collegamento dati fornisce i mezzi funzionali e procedurali per trasferire i dati tra entità di rete e può anche fornire i mezzi per rilevare ed eventualmente correggere gli errori che possono verificarsi nel livello fisico.

Il livello di collegamento dati si occupa della consegna locale di frame tra nodi sullo stesso livello della rete, ossia si assicura che tra due nodi adiacenti la consegna avvenga con successo.

Il routing tra le reti e l'indirizzamento globale sono funzioni di livello superiore, che consentono ai protocolli di collegamento dati di concentrarsi sulla consegna locale, sull'indirizzamento e sull'arbitrato dei media.

In questo modo, il livello datalink è analogo a un vigile urbano di quartiere, si sforza di arbitrare tra le parti che si contendono l'accesso a un mezzo, senza preoccuparsi della loro destinazione finale.

Quando i dispositivi tentano di utilizzare un supporto contemporaneamente, si verificano collisioni di frame.

I protocolli di collegamento dati specificano come i dispositivi rilevano e recuperano da tali collisioni e possono fornire meccanismi per ridurli o prevenirli.

Esempi di protocolli di collegamento dati sono Ethernet e Point-to-Point Protocol.

Point-to-Point Protocol - PPP

PPP spesso compone il livello datalink.

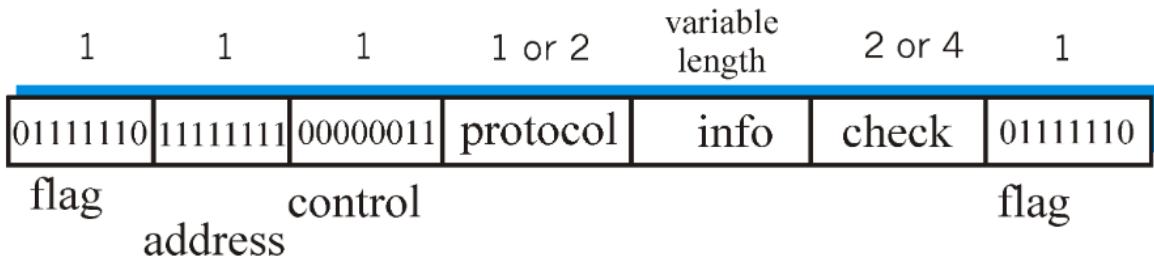
Comunemente usato per stabilire un protocollo di comunicazione tra due nodi della rete.

Usato nei collegamenti punto-punto o per collegare i router ADSL fino al router del provider. PPP è ancora usato in ambiti abbastanza specifici.

PPP implementa:

- **Packet framing**
 - Prende i datagram dal livello network e aggiunge le intestazioni del livello data-link, ottenendo un Frame.
- **Trasparenza dei Bit (*Bit Transparency*)**.
 - I bit nel pacchetto hanno una sequenza precisa.
- **Protocollo Non Affidabile**
 - Questo protocollo è pensato per i link punto-punto con error-rate relativamente bassi (collegamenti in Rame o fibra ottica di lunghezza non troppo grandi).
 - Si presuppone l'esistenza di un protocollo di trasporto come TCP, che esegua effettivamente le operazioni relative all'affidabilità, infatti le 4 procedure elencate qua sotto sono delegate al protocollo TCP.
 - Error Detection e Error Recovery.
 - Ritrasmissione del Pacchetto.
 - Flow Control.
 - Gestione dei Pacchetti *out-of-order*.
- **Connection Liveness**
 - Controlla e segnala eventuali *link failure* al livello network.
- **Network Layer Address Negotiation**
 - Si negozia l'indirizzo di livello rete (ad esempio l'indirizzo IP).

Pacchetto del PPP



- **Flag** → '01111110'
 - Delimiter di inizio frame.
- **Address - 1 Byte**
 - Non fa nulla, messo per lungimiranza, per poter permettere l'uso di PPP in comunicazioni multipunto.
 - Ma per le comunicazioni multipunto si usa ethernet.
 - Di solito ha valore '11111111', ossia l'indirizzo di Broadcast.
- **Control - 1 Byte**
 - Non fa nulla, messo per lungimiranza.
- **Protocol - 1/2 Byte**
 - Specifica a quale protocollo superiore sono diretti i dati del payload. (es IP).
 - Dato che abbiamo solo IP come protocollo superiore, il valore di questo campo è sempre lo stesso.
- **Info (payload) - [0 , 1500] Byte**
 - Il massimo in realtà è "negoziabile" con LCP.
 - È composto interamente dal datagram ricevuto dal livello network.
- **Check (CRC) - 2/4 Byte**
 - PPP fa cadere la connessione nel caso in cui vengano trovati troppi errori.
- **Flag** → '01111110'
 - Delimiter di fine frame.

PPP - Byte “Stuffing” & “Unstuffing”

Il delimiter del pacchetto è il byte 01111110, se nel payload ci fosse (da qualche parte) quel byte, come farebbe il receiver a capire che in realtà quello non è il delimiter di fine messaggio?

Usiamo una tecnica molto simile a quella del *Carattere di Escape*, che in questo contesto è chiamata *Byte Stuffing & Byte Unstuffing*.

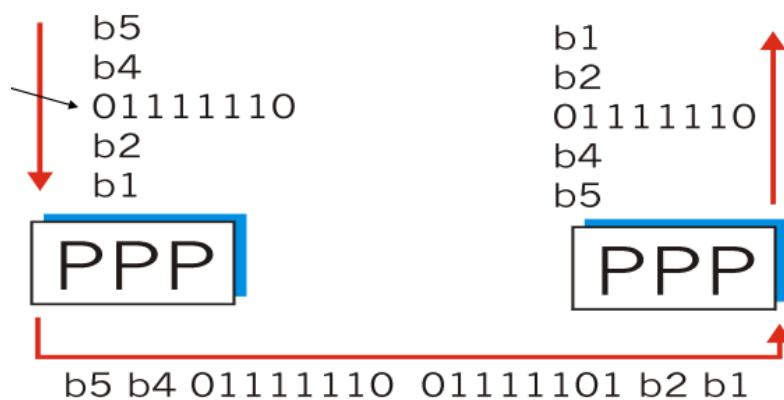
Sender (byte stuffing)

- 01111110 → 01111101 01111110
- 01111101 → 01111101 01111101

Receiver (byte unstuffing)

- 01111101 01111110 → 01111110
- 01111101 01111101 → 01111101

Se nel payload troviamo il byte 01111110, ci inseriamo subito prima un byte speciale (01111101) che verrà interpretato dal destinatario e quindi permetterà di interpretare il byte 01111110 come un semplice byte del payload invece che come delimiter di fine pacchetto. La stessa cosa è fatta anche per il byte 01111101.



LCP - Link Control Protocol

Sottoparte di PPP, con il compito di stabilire, configurare e testare il collegamento, nonché negoziare impostazioni, opzioni e utilizzo delle funzionalità.

Cosa fa LCP?

- Controlla l'identità del dispositivo collegato e accetta o rifiuta il dispositivo.
- Determina la dimensione del pacchetto accettabile per la trasmissione.
- Cerca errori nella configurazione.
- Può interrompere il collegamento se i requisiti superano i parametri.

Domande PPP

QUESTION MARK PPP - Spiegazione dei campi address e control.

Address è un campo messo per rendere PPP adatto al collegamento multipunto.

Control è un byte di controllo.

QUESTION MARK PPP - Byte Stuffing & Byte Unstuffing.

Il frame di PPP ha come delimiter 2 byte con un valore particolare, può accadere che il payload contenga un byte dello stesso valore, quindi per evitare che il receiver comprenda qualcosa di sbagliato, il sender esegue uno "stuffing", ossia inserisce prima del byte un byte

con un valore speciale, il quale ha lo scopo di dire al receiver “*Il prossimo byte non è il delimiter di fine messaggio ma è solo un byte del payload*”.

Questa cosa viene fatta con un byte dal valore speciale, in quale a sua volta richiede la stessa cosa se apparisse nel payload.

PPP - Formato del pacchetto?

E' composto da 7 campi:

1. Delimiter di Inizio Pacchetto.
2. Address.
3. Control.
4. Protocol.
5. Payload.
6. Check.
7. Delimiter di fine pacchetto.

PPP - Come si negoziano i campi inutilizzati?

Le negoziazioni avvengono tramite LCP.

PPP - A cosa servono i due byte di flag?

Servono a delimitare rispettivamente l'inizio e la fine del frame.

Reti ad Accesso Multiplo

Il canale è condiviso da più dispositivi contemporaneamente.

Il problema è il regolare l'accesso al canale condiviso → Ossia “*Parlare uno alla volta*”.

PPP non può gestire un canale ad accesso multiplo, perché è pensato solo per collegamenti punto-punto, ossia quando il canale è condiviso da soltanto due host.

Protocollo di Accesso

Protocollo che a ogni istante determina chi deve trasmettere, ossia regola l'accesso al canale in modo mutuamente esclusivo.

Protocollo di Accesso Ideale

Dato un link condiviso con una capacità pari a R bps.

Il protocollo ideale dovrebbe avere 3 caratteristiche.

- I nodi sfruttano tutta la capacità del canale, e quest'ultima è divisa equamente.
 - Quando M nodi trasmettono, dovrebbero trasmettere con un rate pari a $\frac{R}{M}$ bps.
 - Quando un solo ($M = 1$) nodo trasmette dovrebbe usare tutta la capacità del canale di comunicazione usato e quindi trasmettere a R bps.
- Completamente decentralizzato (**Fully Decentralized**):
 - Non ci sono nodi speciali che coordinano le trasmissioni.
 - Niente sincronizzazioni di clock (non voglio un clock comune).
 - Niente sincronizzazioni di slots.
- Semplice
 - Kiss → “*Keep It Simple Stupid*”.

Tassonomia dei Protocolli MAC

3 Classi principali:

- **Partizionamento di Canale**
 - Il mezzo viene diviso in pezzi, la definizione di “pezzo” dipende dall’implementazione.
 - Collisioni non possibili, perché l’accesso è regolato e solo un host alla volta può trasmettere.
 - A ogni nodo è allocato (secondo una certa politica) un “pezzo” del canale.
 - Le definizioni di “pezzo” possono essere varie, noi ne vedremo solo una in particolare:
 - **Slot di Tempo (TDMA).**
 - A ogni nodo è associato un quantitativo di tempo durante il quale può usare il canale come se fosse tutto suo.
 - **Frequenza (FDMA).**
 - A ogni nodo è associata una certa frequenza.
 - Codice.
 - etc...
- **Accesso Casuale (Random Access Protocol)**
 - L’accesso al canale non è regolamentato, le collisioni sono possibili.
 - Comunque sono definite tecniche per recuperare dalle collisioni.
 - Un host prima di usare il canale controlla che il canale non sia occupato.
- **Turnazione “Taking Turn”**
 - I nodi hanno un turno per l’accesso al canale, durante il quale è completamente dedicato al nodo.

Time Division Multiple Access - TDMA

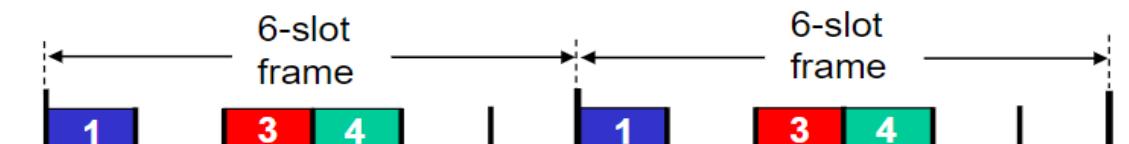
Il tempo è diviso in slot di tempo aventi una durata fissa.

A ogni Host sono assegnati uno (o più) slot, periodicamente quando è il loro turno, l’host può trasmettere qualcosa.

I nodi devono avere il clock sincronizzato → Elemento di Centralizzazione.

Le collisioni non sono possibili, perché tutti i nodi sanno precisamente quando possono e non possono trasmettere.

Durante uno slot di tempo, il canale è totalmente dedicato al nodo possessore di quello slot.



- TDMA è “fair”?
 - Dipende se ci sono abbastanza slot per tutti i nodi che utilizzano il canale.
 - Se tutti i nodi della rete hanno lo stesso numero di slot si.
- TDMA è completamente decentralizzato?

- Dipende dall'implementazione, se c'è un supernodo coordinatore che assegna gli slot no, se invece i nodi scelgono da soli il loro slot (tramite un sistema opportuno) lo è.

Quando un Protocollo è “Fair”?

Un protocollo è “fair”(giusto) se tutti i membri della rete hanno le stessa quantità di risorse e vengono trattati allo stesso modo.

In norma di principio nelle reti si dovrebbe cercare di rendere i sistemi il più fair possibili.

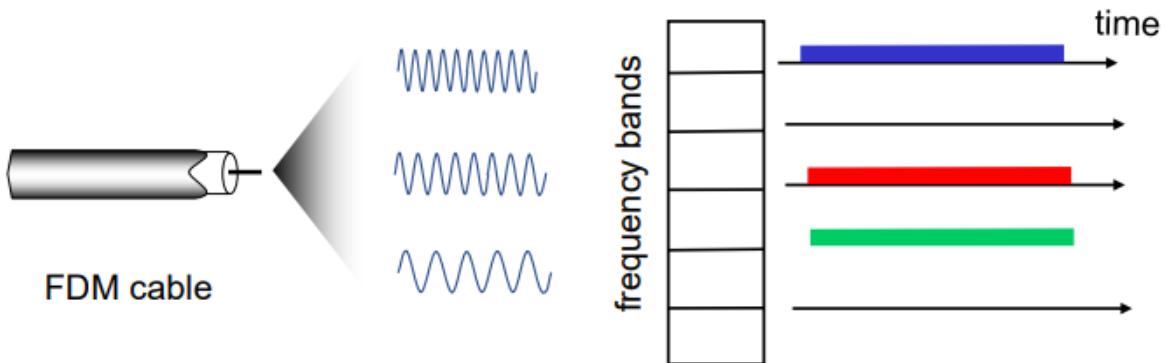
Frequency Division Multiple Access - FDMA

Usata dalle Radio.

Lo spettro delle frequenze del canale di comunicazione è diviso in bande.

Ogni stazione ha una sua banda fissata.

Sintonizzandosi su quella banda sentirò solo quella stazione radio.



Random Access Protocols

Trasmetto quando voglio, senza coordinarsi con qualcuno.

Ovviamente prima di trasmettere controllo che il canale sia libero, se è libero trasmetto.

Se ci sono 2 nodi che trasmettono contemporaneamente → Avviene una Collisione.

Un protocollo ad accesso casuale deve obbligatoriamente avere un meccanismo di gestione delle collisioni.

Slotted ALOHA

Versione “slotted” di ALOHA.

La versione originale la vedremo subito dopo.

L'accesso al canale è casuale ma il tempo è comunque diviso in slot temporali di durata nota.

Un Host può iniziare una trasmissione solo in concomitanza con l'inizio di uno slot.

Uno slot deve dare il tempo necessario per trasmettere un frame e ricevere il relativo frame di ACK, quindi in genere: $Durata_{Slot} > RTT_{medio}$

Supponiamo che tutti i frame abbiano la stessa dimensione.

I nodi sono sincronizzati con lo stesso clock.

Se avviene una collisione tutti i nodi lo sapranno.

Quando un nodo ha un pacchetto da trasmettere, attende l'inizio del prossimo slot senza verificare che non ci siano altri nodi che vogliono trasmettere.

- *Non ci sono Collisioni:*
 - Il nodo riconosce che non c'è stata perché riceve il relativo ACK.
 - Il nodo può mandare un altro frame nello slot seguente.
- *Ci sono collisioni:*
 - Il nodo ri-trasmette lo stesso frame nello uno slot seguente con una certa probabilità p.

Forse ti stai chiedendo perché semplicemente non ascolta prima il canale...ricorda che gli Host possono trasmettere solo all'inizio di uno slot, quindi se avviene una collisione è perché due host hanno deciso di trasmettere nello stesso identico slot.

Slotted ALOHA - “La probabilità p”

“Al prossimo slot ri-trasmetto o sto fermo?”

E' come lanciare una moneta truccata.

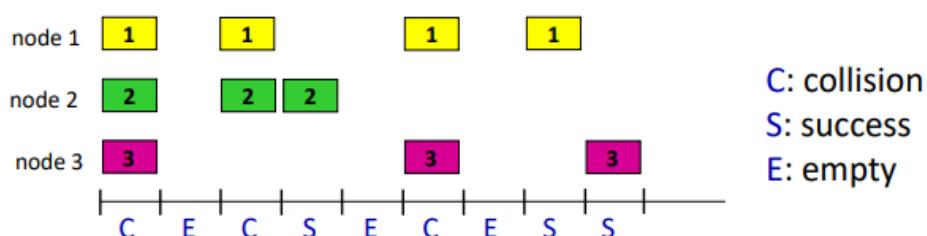
Truccata perché in genere $p < 0.5 \rightarrow$ “Trasmetto con una probabilità inferiore al 50%”.

Il lancio della moneta viene fatto dopo una collisione dai trasmettitori che hanno trasmesso i pacchetti che hanno colliso.

Nello slot seguente a quello in cui è avvenuta una collisione, in base al risultato del lancio della moneta:

- Trasmetto il frame per intero.
- Aspetto lo slot seguente e lancio di nuovo la moneta.

Il valore di p lo decide il progettista della rete.



Al primo slot, $p = 1$, quindi l'Host trasmette di sicuro.

In questo esempio, per trasmettere 3 pacchetti mi ci sono voluti 9 slot temporali.

- Con 3 slot di “*idling*”, ossia in cui nessuno trasmette.
- Con 6 slot di lavoro, ossia quando almeno un nodo trasmette.
 - Con 3 slot in cui è avvenuta una collisione.
 - Con 3 slot di avvenuta trasmissione.

Slotted ALOHA - Statistiche Caso Generale

Supponiamo di avere N nodi nel sistema.

Supponiamo che ogni nodo abbia sempre un pacchetto pronto per la trasmissione (*condizione di overload asintotico*).

Nel primissimo slot di sicuro avverrà una collisione tra tutti gli N nodi.

Dal secondo in poi si lancia la moneta.

Probabilità che il nodo i trasmetta con successo in uno slot $\rightarrow P_i(p) = p(1 - p)^{N-1}$

Probabilità che un qualsiasi nodo trasmetta con successo in uno slot \rightarrow

$$P_N(p) = Np(1 - p)^{N-1}$$

Sia p^* il valore di p che massimizza (secondo p) il valore di P_N .

(p^* è trovato tramite le tecniche di analisi 1, ossia il valore di p che azzera la derivata prima di P_N).

Supponiamo che $N \rightarrow \infty$ e fissiamo $p = p^*$.

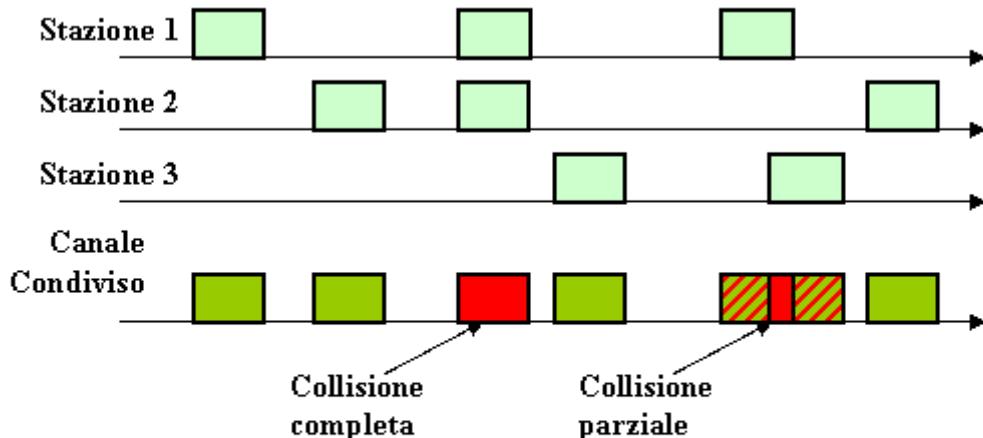
$$\lim_{N \rightarrow \infty} Np(1 - p)^{N-1} = \frac{1}{e} = 0.37 \rightarrow \text{Massima Efficienza} = 37\%$$

L'efficienza proposta non è bellissima ma non è la peggiore in circolazione.

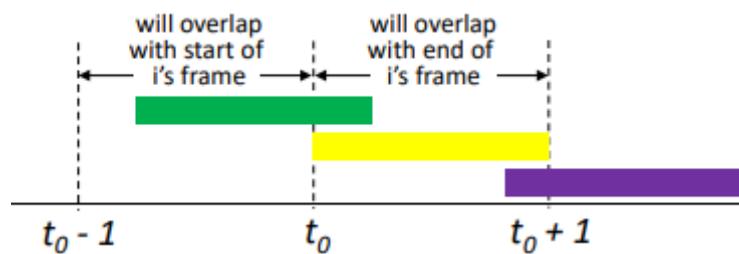
Pro	Contro
Un nodo in uno slot può sfruttare al massimo la banda del canale , perché il quello slot (a scanso di collisioni) è totalmente dedicato a lui.	Ci possono essere slot di idling, in cui il canale rimane inutilizzato.
Generalmente Decentralizzato, ma gli slot nei nodi devono essere sincronizzati. Quindi i clock dei nodi devono essere sincronizzati.	Alta probabilità di collisioni, quindi la probabilità di perdere qualche slot è alta.
Semplice	I nodi potrebbero rilevare le collisioni in meno tempo rispetto a trasmettere un pacchetto
In generale è <i>fair</i> , perché p è un valore casuale..	Sincronizzazione del Clock. Gli slot di tempo devono durare lo stesso tempo per tutti i nodi della rete

Pure ALOHA (Unslotted ALOHA)

Tutti i frame hanno la stessa dimensione, e tutti i nodi impiegano lo stesso quantitativo di tempo per trasmetterlo e per ricevere il relativo ACK.



Il trasmettitore trasmette senza controllare che il canale sia occupato, quindi aumenta la probabilità di collisione, perché una seconda trasmissione può iniziare in qualsiasi momento.



Per non verificare una collisione con il verde, nessuno dovrebbe trasmettere tra l'istante t_0-1 e t_0+1 .

Se una stazione dovesse trasmettere nell'intervallo di tempo tra t_0-1 e t_0+1 accadrà di sicuro una collisione.

$$\begin{aligned}
 P(\text{success by given node}) &= P(\text{node transmits}) * \\
 &\quad P(\text{no other node transmits in } [t_0-1, t_0]) * \\
 &\quad P(\text{no other node transmits in } [t_0-1, t_0]) \\
 &= p \cdot (1-p)^{N-1} \cdot (1-p)^{N-1} \\
 &= p \cdot (1-p)^{2(N-1)} \\
 \dots \text{ choosing optimum } p \text{ and then letting } n \\
 &= 1/(2e) = .18 \rightarrow \infty
 \end{aligned}$$

Carrier Sense Multiple Access - CSMA

Prima ascolto il mezzo e vedo se è libero.

- Se è occupato → Aspetto un pò di tempo.
- Se è libero → Trasmetto.

Le collisioni possono comunque verificarsi, perché quando il canale è libero tutti i nodi vedono che è tale e quindi si fondono a trasmettere.

E'opportuno accorgersi il prima possibile che sta avvenendo una collisione.
Durante una collisione il segnale risultante nel canale è incomprensibile poiché pesantemente disturbato.
E quindi durante una collisione il canale è inutilizzabile.

Carrier Sense Multiple Access Collision Detection - CSMA/CD

Utilizzabile solo in link di tipo "Wired".

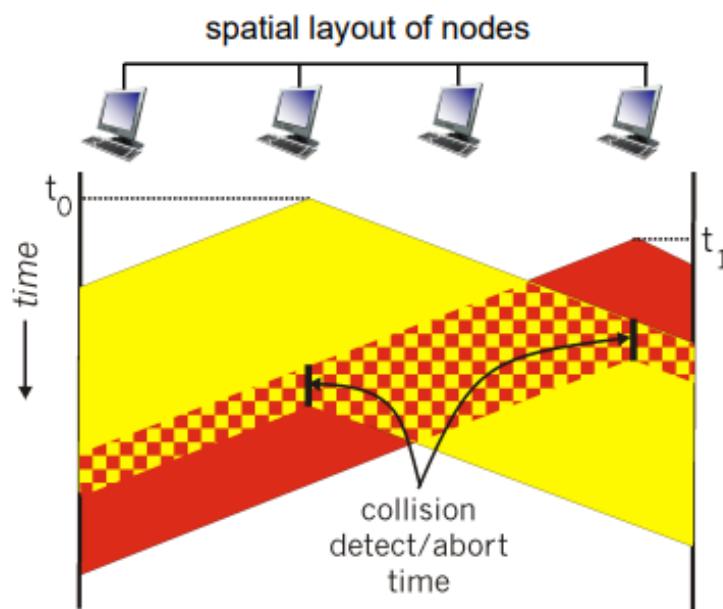
Le collisioni vengono rilevate in poco tempo

Le trasmissioni che hanno colliso vengono abortite, riducendo lo spreco di tempo.

CSMA/CD - Rilevamento delle Collisioni

Il rilevamento delle collisioni ha a che fare con la propagazione del segnale.

All'istante $t_1 > t_0$, l'host 4 non ha ancora visto il segnale giallo (invia a t_0 dall'host 1), quindi trasmette il segnale rosso.



Ad un certo punto, l'host 1 e l'host 4 in due momenti diversi rilevano la collisione e smettono di trasmettere.

Per rilevare la collisione si confronta la potenza del segnale rilevata nel canale con quella del segnale trasmesso.

Il trasmettitore conosce bene i dettagli di quello che trasmette, anche la potenza del segnale. Se si accorge che nel canale c'è una potenza superiore a quella che trasmette lui allora per forza c'è anche un altro nodo che sta trasmettendo.

Il CSMA/CD punta a ridurre il più possibile il tempo in cui il segnale è incomprensibile a causa di una collisione.

Note su CSMA/CD

- CSMA/CD è usato in Ethernet.

- CSMA/CD è fair?
 - Si, tutti i nodi hanno le stesse risorse.
- CSMA/CD è *fully decentralized*?
 - Si, ogni nodo si comporta in maniera sincrona in confronto agli altri.
 - La rilevazione delle collisioni è fatta dai nodi stessi.
- CSMA/CD è semplice?
 - Si.

CSMA/CD - Come Funziona

1. NIC riceve un datagram dal livello superiore.
 - Viene creato un frame con tutte le informazioni necessarie.
2. NIC resta in ascolto (*idle*) del canale per 96 Bit Time.
 - In secondi $\rightarrow t_{idle} = \frac{96\ bit}{bitrate}$
 - Se non rileva nulla → Vai al punto 3.
 - Se rileva qualcosa → Vai al punto 2.
3. Inizia a trasmettere il frame, ma nel mentre ascolta il mezzo.
 - Se non rileva nulla, continua a trasmettere.
 - Se rileva un'altra trasmissione, ossia sta avvenendo una collisione:
 - i. Abortisce la trasmissione corrente.
 - ii. Manda un segnale di *Jam*.
 - iii. Tutti i trasmettitori coinvolti entrano in uno stato di "exponential backoff".
 - Dopo la i -esima collisione:
 - NIC sceglie un numero casuale $K \in \{0, 1, 2, 4, \dots, 2^m - 1\}$
 - Con $m = \min(i, 10)$.
 - Quando $i > 10$, si scarta il pacchetto.
 - NIC imposta un *Backoff Timer* = $(K * 512)$ bit.
 - NIC aspetta che scada il *Backoff Timer* e ritorna al punto 2.

CSMA-CD - Segnale di Jam

Il segnale di Jam serve per avvertire tutti i trasmettitori che si è appena verificata una collisione.

Perché i nodi trasmettitori si accorgono della collisione in momenti diversi e può capitare che un nodo che sta trasmettendo durante una collisione smetta di trasmettere prima di rilevare la differenza di potenza del segnale.

Quando il segnale dell'altro trasmettitore arriverà a questo trasmettitore quest'ultimo avrà smesso di trasmettere e quindi non rileverà niente di strano nella potenza del segnale e arriverà a concludere che non ci è stata nessuna collisione, il segnale di Jam serve ad avvertire i trasmettitori che rientrano in questo caso.

CSMA-CD - Backoff Timer

Il Backoff Timer serve a diminuire la probabilità che due trasmettitori trasmettono di nuovo contemporaneamente.

In altre parole, serve a desincronizzare i trasmettitori che hanno causato una collisione e quindi prevenire altre (e quasi certe) collisioni.

Questa tecnica è “Adattiva”, più il canale è utilizzato, più è alto il numero di collisioni e quindi più alti saranno i valori possibili del Backoff Timer.

Calcolo del Bit Time

Un Bit Time è il tempo necessario a mandare 1 bit.

Considerato un bitrate di 10 Mbps, Il bit time è di circa 0.1 Microsecondi.

$$Bit_{Rate} = 10 \text{ Mbps} = \frac{10'000'000 \text{ bit}}{1 \text{ sec}} = \frac{1 \text{ bit}}{1 * 10^{-7} \text{ sec}}$$

$$Bit_{Time} = 10^{-7} \text{ sec} = 10^{-1} \mu\text{sec} = 0.1 \mu\text{sec}$$

Protocolli MAC di tipo “Channel partitioning”

Conveniente quando ci sono tanti nodi, perché le collisioni sono evitate.

Il canale è condiviso in modo efficiente e giusto a fronte di carichi alti.

Protocolli MAC di tipo “Random Access”

Conveniente quando ci sono pochi nodi (Reti di tipo “low load”), perché permette ai nodi di utilizzare il 100% del canale.

Pericolo di Collisioni, le quali però a fronte di pochi nodi sono rare e facilmente gestibili.

Protocolli “Taking Turn”

I protocolli “Taking Turn” prendono il meglio dei due tipi di protocolli, ossia che funziona bene sia con tanti nodi che con pochi nodi.

Protocollo “Taking Turn” - Polling

C’è un master node che “invita” gli altri nodi a trasmettere a turno.

Il master chiede (l’ordine con cui interagisce con gli slave è definito da un algoritmo di scheduling del master node, il quale può non essere equo) a tutti “*Hai qualcosa da trasmettere?*”, se sì lo slave trasmette.

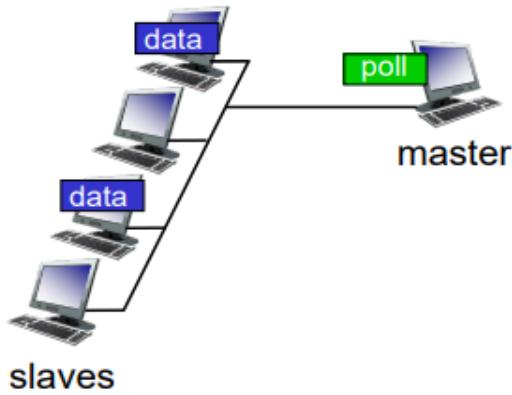
Il master è single point of failure, anche se in caso di morte del master uno slave potrebbe essere eletto a master secondo una certa politica.

I pacchetti di poll comunque aggiungono un pò di overhead, a prescindere dal numero di nodi nella rete.

Perché si spreca tempo ad aspettare che il master invii al prossimo trasmettitore il messaggio di poll.

Il protocollo è Fair? Dipende dalla politica di distribuzione dei turni adottata dal nodo master.

Il protocollo è Decentralizzato? No, c’è un nodo master.

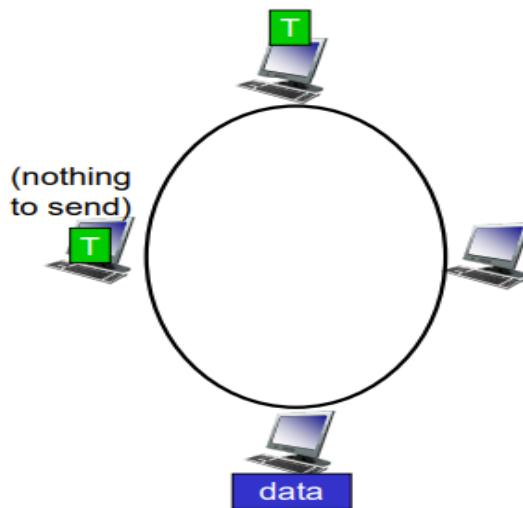


Protocollo “Taking Turn” - Token

Chi ha il token trasmette, il token è passato in maniera circolare come un testimone a tutti i nodi della rete.

Si introduce un overhead dovuto al passaggio del token, quindi l'utilizzazione del canale non sarà mai al 100%, se non considero il token allora a quel punto si ha una piena utilizzazione. Il nodo che possiede il token è un possibile single point of failure, perché se il token non viene trasmesso la rete si blocca.

Il protocollo è pienamente decentralizzato? No, Il token è un elemento di centralizzazione.



Local Area Network - LAN

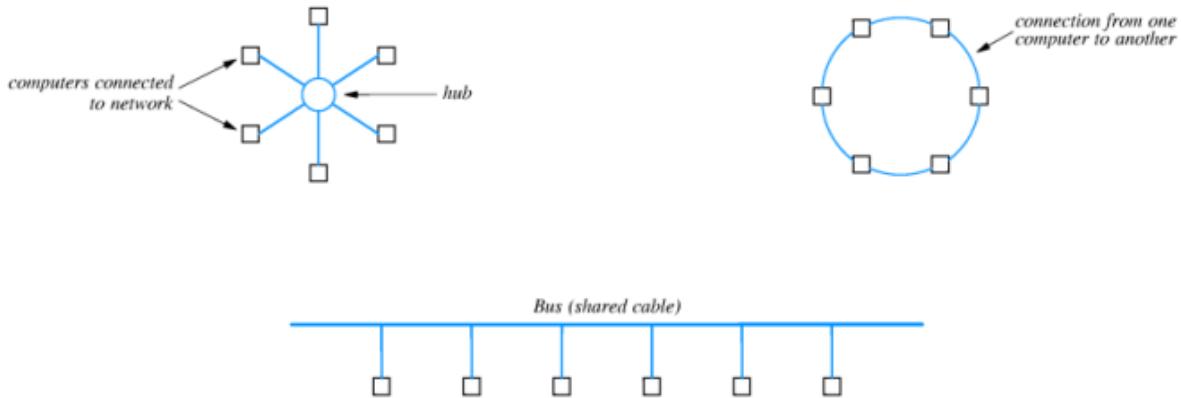
Rete che connette un numero limitato di computer, con un mezzo unico condiviso di tipo broadcast (come un cavo in rame o l'etere).

La copertura è nell'ordine del centinaio di metri.

In genere le LAN hanno un canale di comunicazione con un bitrate che va da 100 Mbps fino a 10 Gbps.

Topologia delle LAN

Le LAN possono avere diverse tipologie, dalla topologia della rete possono derivare alcune proprietà e ogni tipologia ha i suoi punti deboli.



Indirizzo MAC

Posso avere una comunicazione riservata (*unicast*) in un mezzo broadcast?

Si, tramite un indirizzo che identifica in modo univoco il nodo, l'indirizzo MAC.

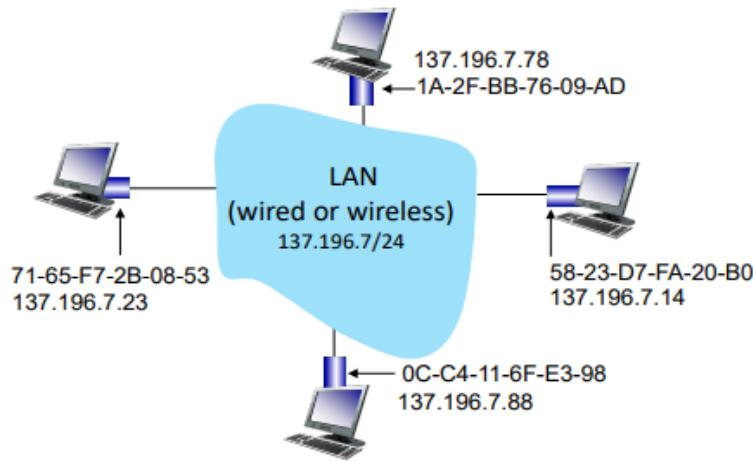
L'indirizzo MAC è su 48 bit nelle reti locali → 1A-2F-BB-76-09-AD

Ogni coppia di lettere è su 8 bit.

L'indirizzo MAC è installato via hardware direttamente nella scheda di rete (NIC), ma ormai anche gli indirizzi MAC (a volte) sono modificabili via software.

L'indirizzo MAC trascende la rete di appartenenza perché è **unico nel mondo** e appartiene solo alla singola scheda.

L'indirizzo MAC è dato dalla casa costruttrice, la quale ha acquistato da una entità ufficiale un blocco di indirizzi MAC.



L'assegnazione degli indirizzi MAC è amministrata dalla IEEE, da cui le aziende costruttrici comprano blocchi di indirizzi MAC da usare nelle loro schede di rete.

Tramite l'indirizzo MAC la comunicazione diventa anche **UNICAST** (se i nodi sono tutti onesti).

Quando arriva un messaggio nel canale condiviso, ogni nodo controlla l'indirizzo MAC del destinatario, se non è il suo il nodo ignora il segnale del bus.

Indirizzo MAC - Modalità Promiscua della Scheda di Rete

La scheda di rete può essere impostata in modalità "promiscua", nella quale ignora il controllo dell'indirizzo MAC e quindi acquisisce tutti i pacchetti che vengono trasmessi nel mezzo di broadcast.

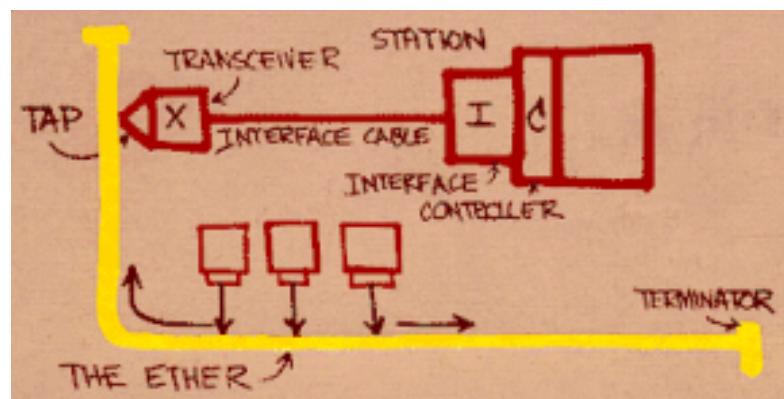
Indirizzo MAC di Broadcast

Comunque è definito un indirizzo MAC di "Broadcast", con cui ogni nodo vedendo quello specifico indirizzo MAC accetterà il pacchetto come se fosse diretto a lui.

Ethernet

Ethernet è una famiglia di tecnologie standardizzate per reti locali.

Ha avuto molto successo perché è estremamente semplice, quindi molto robusta ed economica.



In ethernet si può avere un bitrate che varia tra 10 Mbps ai 400 Gbps.

Il bus viene chiamato "*the ether*".

Ethernet - Scheda di Rete

Il computer ha la scheda di rete, composta da 4 componenti:

- **Controller.**
- **Interface.**
- **Interface Cable.**
 - Collega l'interfaccia al Transceiver, ora è stato eliminato
- **Transceiver.**
 - Genera i segnali da propagare nel bus, ora è direttamente integrato nell'interfaccia.

Ethernet - Versione Bus

Ethernet prevede una topologia a BUS mediante un cavo coassiale.

Ogni nodo può collidere con tutti gli altri nodi.

Ethernet - Versione Hub

Ma ethernet prevede anche una topologia a stella tramite un Hub come nodo centrale, l'hub manda in broadcast il messaggio ricevuto da un nodo (ovviamente rigenerando il segnale, quindi agisce anche da ripetitore).

Ogni nodo può collidere con tutti gli altri nodi.



Ethernet - Versione Switch

Ora si usa la versione a stella ma con uno Switch al posto di un Hub.

Lo switch è molto più complesso di un Hub e agisce ad un livello più alto (livello data link), invece il bus opera a livello fisico poiché il suo compito è solo quello di rigenerare il segnale e di inviarlo a tutti.

Lo switch è un componente Hardware che per sua natura evita le collisioni, poiché applica una politica di tipo “*Store and Forward*”.

Il nome è rimasto Ethernet per motivi anche commerciali, ma oramai è del tutto diversa.

Ethernet - Collisioni

In ethernet (nella versione bus e hub, ma non in quella switch) tutti i nodi possono collidere, poiché il mezzo è condiviso e l'accesso non è regolamentato

Ethernet - Formato del Frame

Il frame ethernet intero è minimo 64 Byte.



- **Preambolo - 7 byte**

- Byte del tipo → 10101010.
- Informazioni inutili che servono a sincronizzare (la sincronizzazione avviene “*on the fly*”) il clock del ricevitore con quello del trasmettitore.
 - Se il trasmettitore fosse più veloce del ricevitore quest’ultimo si lascerebbe scappare dei bit.
- Nella pratica i 2 clock non saranno perfettamente sincronizzati, quando si hanno velocità molto alte la *bit error rate* aumenta.

- **Start Frame Delimiter - 1 Byte**

- Byte del Tipo → 10101011.
- Indica l'inizio del frame.

- **Destination MAC Address - 6 Byte**

- **Source MAC Address - 6 Byte**
- **Type - 2 Byte**
 - Indica un protocollo di livello superiore al quale vanno passati i dati del payload dopo aver verificato l'assenza di errore.
 - Il protocollo indicato è quasi sempre il protocollo IP.
- **Payload - [48 , 1500] Byte**
- **CRC - 4 Byte**

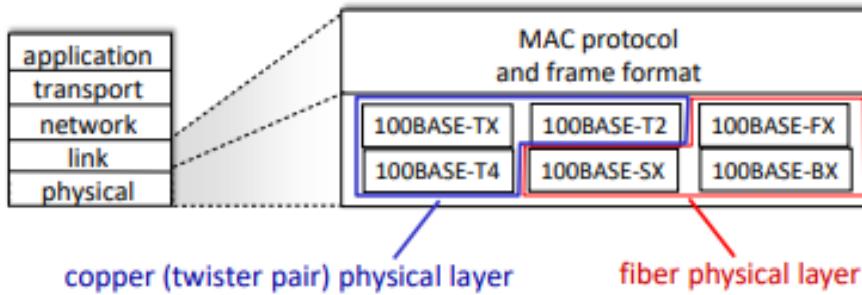
Caratteristiche di Ethernet

- **Connectionless**
 - Non serve un handshake iniziale, il trasmettitore invia la roba e basta (seguendo CSMA/CD per l'accesso al canale).
 - Il ricevitore ha il compito di sincronizzare il clock con il trasmettitore e di ricevere le informazioni.
- **Unreliable**
 - In base al risultato dell' *error detection*:
 - Se il risultato è positivo (no errori) si isola il payload e quest'ultimo si passa al livello superiore.
 - L'error detection utilizzata non è affidabile al 100%, potrebbe produrre falsi positivi, ma di sicuro non produrrà mai falsi negativi.
 - Se il risultato è negativo **butto via tutto il frame** appena ricevuto come se non fosse mai arrivato.
 - Il pacchetto non verrà mai passato al livello superiore, le conseguenze verranno gestite da altri (ossia dai livelli superiori).
- **CSMA/CD** con Backoff Binario per rilevare e gestire le collisioni.
 - Binario → K è sempre una potenza di 2.

Gli Standard del Livello Fisico - NO ESAME

Ogni standard del livello fisico è identificato da una sigla di 3 componenti.

- **Numeri:**
 - Capacità del mezzo trasmissivo espressa in Mbps.
- Parola che dà **Informazioni sul segnale**.
 - BASE → Segnale Non Modulato.
- **Mezzo Tramissivo** utilizzato
 - TX → Doppino Telefonico.
 - T2 →
 - T4 →
 - FX → Fibra ottica.
 - SX →
 - BX →



Packet Switched Networks - Reti a Commutazione di Pacchetto

Reti che si basano su un componente ulteriore che si occupa di effettuare il packet switching.

Hub

Nelle prime versioni “a stella” di ethernet era usato un dispositivo “stupido” (ossia che lavorava esclusivamente a livello fisico) chiamato *hub* che fungeva da **semplice ripetitore del segnale**.

Viene usata spesso nelle case per connettere tra loro qualche dispositivo (di solito non più di 10), questo perché comunque un Hub costa molto meno rispetto ad uno switch e per un caso del genere possiamo considerare uno switch “*un po’ troppo*”.

Switch

Lo switch lavora a livello 2 (data-link) e ha un ruolo “attivo” nella rete.

Lo switch applica una politica di “*store and forward*” sui frame che riceve, ovviamente ha anche il compito di ripristinare il segnale che riceve (come l’hub).

1. Riceve un pacchetto Ethernet.
2. Legge il destinatario dall’header del pacchetto. (*Store*)
3. Inoltra il pacchetto esclusivamente al destinatario. (*Forward*)

Lo switch a livello data-link associa un certo indirizzo MAC ad una certa porta di uscita.

Infatti lo switch autonomamente inoltra i pacchetti solo al destinatario.

La presenza dello switch è trasparente all’utente, ossia per l’utente il fatto che ci sia un hub o uno switch non cambia nulla.

Con l’utilizzo di uno switch si evitano del tutto le collisioni ed altri effetti negativi per la velocità di trasmissione.

Come è configurato lo switch?

La configurazione dello switch è interamente contenuta nella tabella di switching.

La tabella dello switching è compilata automaticamente e dinamicamente dallo switch stesso (si può dire che lo switch è *plug and play*).

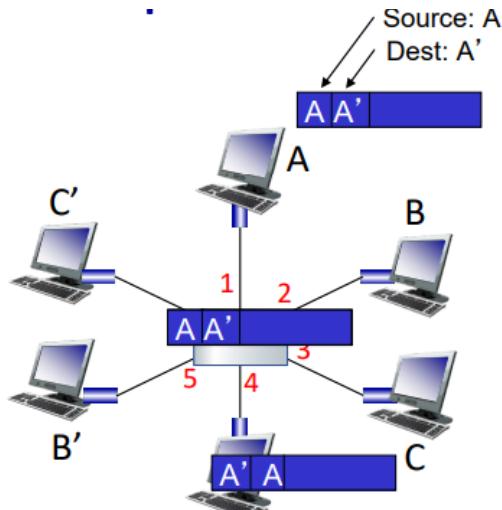
Lo switch impara quali host possono essere raggiunti da quale interfaccia in base ai frame che riceve.

Quando un nodo invia un qualsiasi frame viene subito identificato dallo switch.

Come Funziona lo Switch

Supponiamo che la tabella di switching sia inizialmente vuota.

1. Lo switch riceve un frame in ingresso da un'interfaccia e consulta la tabella di switching.
 - a. L'interfaccia di origine non ha corrispondenze con la tabella.
 - b. Aggiunge l'indirizzo MAC del sorgente del frame, associandolo all'interfaccia da cui è arrivato il frame.
2. L'indirizzo di destinazione non ha corrispondenze con la tabella.
 - a. Quindi non sapendo a chi mandarlo, invia il frame in broadcast, ossia dentro a tutte le porte.
3. Il destinatario lo riceve e invia il relativo ACK.
4. Lo switch riceve il frame di ACK.
 - a. L'interfaccia di origine non ha corrispondenze con la tabella.
 - b. Aggiunge l'indirizzo MAC del mittente del frame di ACK, associandolo all'interfaccia da cui è arrivato il frame di ACK.
5. Entrambi i nodi ora sono identificati e hanno entrambi una riga nella tabella di switching.



MAC addr	interface	TTL	
A	1	60	switch t
A'	4	60	(initially e)

Time to Live - TTL

Le righe nella tabella di switching non possono esistere in eterno, quindi si dà ad ogni riga un tempo di vita, oltre al quale diventa non più valida e quindi viene eliminata.

Esprime quanto *tempo* quell'informazione resterà dentro la tabella di switching, per non avere informazioni potenzialmente obsolete o inutili (perchè quegli host non esistono più).

E' importante mantenere la tabella di switching piccola, per mantenere l'overhead introdotto dallo switch il più basso possibile e per non avere informazioni obsolete che potrebbero causare forwarding errati.

Switched Ethernet

In una rete ethernet basata su switch due o più nodi possono trasmettere contemporaneamente.

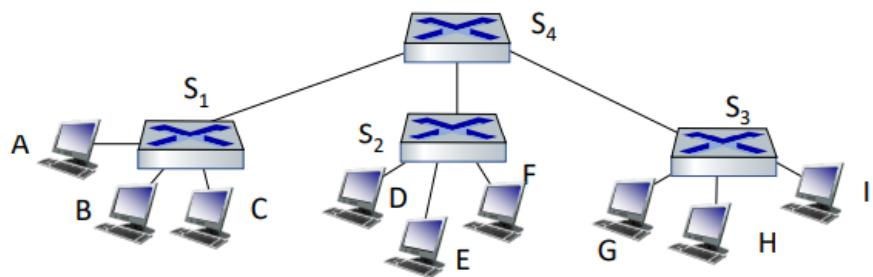
Tanto lo switch applicando lo “store and forward” impedisce ai segnali di collidere.

Perché sarà lo switch stesso a inoltrare in modo opportuno i frame e anche perché lo switch rende il mezzo ancora condiviso ma fa in modo che i segnali rimangano isolati gli uni dagli altri.

Rete con gerarchia di Switch

Con tanti switch posso creare una specie di albero di switch.

Supponiamo che il Nodo A conosca il MAC del nodo G.



Supponiamo che le tabelle di switching siano tutte vuote.

Supponiamo che il Nodo A conosca il MAC del nodo G.

1. A vuole mandare un pacchetto a G.
2. S1 aggiunge A alla tabella di switching.
3. S1 lo riceve non ha il MAC di G nella tabella e quindi lo manda in broadcast.
4. S4 aggiunge A alla tabella di switching.
5. S4 lo riceve non ha il MAC di G nella tabella e quindi lo manda in broadcast.
6. S3 aggiunge A alla tabella di switching.
7. S3 lo riceve non ha il MAC di G nella tabella e quindi lo manda in broadcast.
8. G riceve e risponde con un ACK indicando come destinatario il Nodo A.
9. S3 aggiunge G alla tabella di switching.
10. S3 lo riceve, ha il MAC di A nella tabella e quindi lo manda dall'interfaccia da cui è arrivato prima, quindi verso S4.
11. S4 aggiunge G alla tabella di switching.
12. S4 lo riceve, ha il MAC di A nella tabella e quindi lo manda dall'interfaccia da cui è arrivato prima, quindi verso S1.
13. S1 aggiunge G alla tabella di switching.
14. S1 lo riceve e manda il frame ad A.

In S1 cosa rimane nella tabella di switching?

- A è raggiungibile mediante la porta corrispondente.
- G è raggiungibile attraverso la porta corrispondente a S4.

In S4 cosa rimane nella tabella di switching?

- A è raggiungibile mediante la porta corrispondente a S1.

- G è raggiungibile attraverso la porta corrispondente a S3.

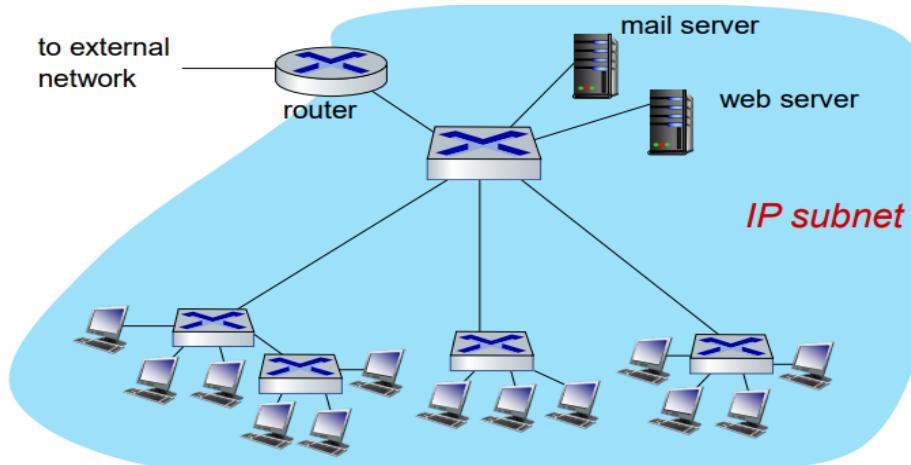
In S3 cosa rimane nella tabella di switching?

- A è raggiungibile mediante la porta corrispondente a S4.
- G è raggiungibile attraverso la porta corrispondente.

Nella tabella di switching posso avere molteplici host associati alla stessa interfaccia.

Reti Istituzionali

Gli switch permettono di dividere gli host di un'unica grande rete in gruppi, ognuno con un loro switch.



Il problema è che:

- Con tanti switch si aumenta il ritardo, perché ogni switch ha bisogno di tempo per elaborare il pacchetto e inoltrarlo (ogni switch introduce overhead).
- Più switch metto, più “hop” ho, quindi più probabilità che un pacchetto subisca corruzione, a ogni hop aumenta la possibilità di errore.

Zona Demilitarizzata - DMZ

In una rete istituzionale, in genere i server si mettono tutti molto vicini allo switch radice (o comunque vicini all'esterno) per ridurre al minimo i numeri di hop necessari a ricevere o inviare un frame con l'esterno (perchè i server sono visitabili dall'esterno) e per motivi di sicurezza.

Gli host interni della rete devono essere inaccessibili all'esterno.

I server invece li metto nella “Zona Demilitarizzata” che permette agli utenti esterni di collegarsi.

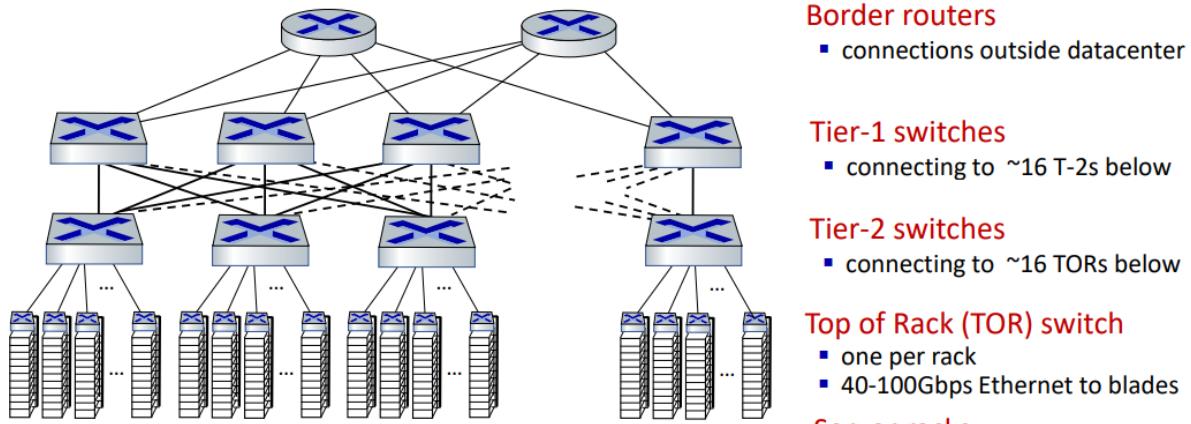
Questa cosa la approfondiremo quando faremo i router.

Reti per Datacenter

In una rete del genere ho centinaia di host molto vicini.

In genere la si trova in aziende che ospitano numerosi server.

I server vengono messi in un datacenter per efficienza di gestione, energetica e soprattutto per motivi di sicurezza e affidabilità.



Obbiettivi:

- Ho molte applicazioni, ognuna serve moltissimi clienti.
- Affidabilità.
- Gestire e distribuire il carico tra tutti i server.
- Evitare colli di bottiglia.

Anche qua si può usare una rete packet switched ma con alcune modifiche, il concetto di base è lo stesso.

Alla base ho dei “Rack” ognuno dei quali contiene una trentina di server → Hosts.

Ogni rack ha un TORs (*Top of Rack Switch*) , uno switch con bitrate molto alto , ha il compito di interconnettere un intero rack.

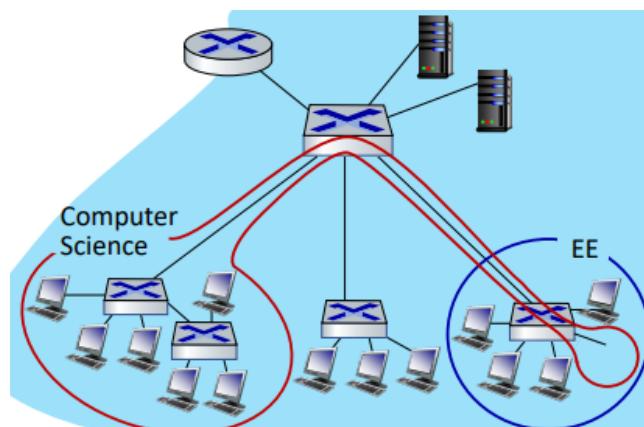
Gli switch di livello 2 collegano i TORs.

Gli switch di livello 1 collegano gli switch di livello 2 , tramite un sistema di collegamenti molto diramato per fare in modo che in caso di guasto di un link di comunicazione la rete continui ad essere operativa.

LAN Virtuali - VLAN

Posso virtualmente dividere una LAN in varie parti, questa cosa si fa assegnando a ogni VLAN un insieme di porte di uno switch.

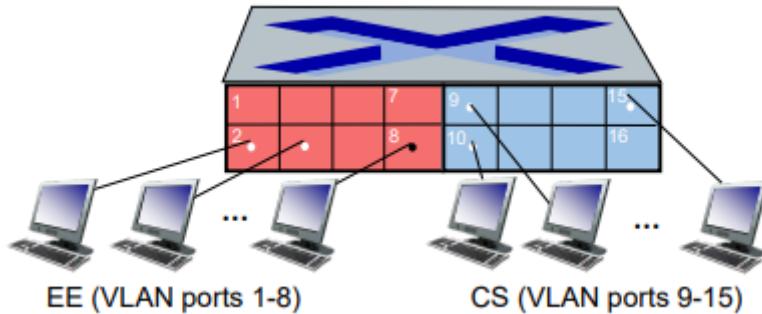
“Le porte dalla 1 alla 8 sono della VLAN EE, il resto della VLAN VR”.



Sicurezza → Posso isolare una VLAN da un'altra.

Le VLAN vengono configurate direttamente negli Switch.

E' come se dividessi un grande switch in vari switch più piccoli indipendenti tra loro.



Tramite le VLAN posso isolare il traffico.

Un pacchetto proveniente da un host della VLAN EE può arrivare solo ad altri host di EE senza mai mandare nulla agli host di VR.

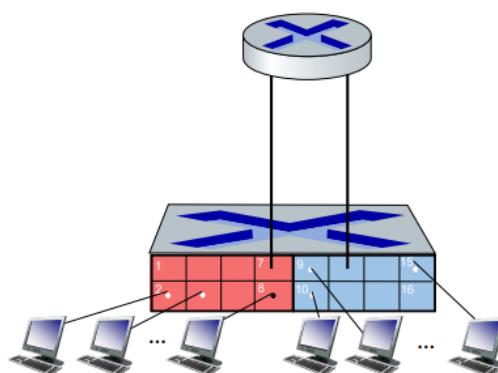
Ogni VLAN è un dominio di broadcast a sé.

Un pacchetto proveniente da un host della VLAN EE inviato in broadcast arriva solo agli host di EE.

Comunicazione tra VLAN diverse

La si può ottenere in 2 modi;

- Spostando la propria porta nella VLAN desiderata (*Membership Dinamica*).
- Passando al livello 3 e quindi usando un router che connette le due VLAN come se fossero due reti diverse.



Una VLAN può anche essere fatta basandosi su Indirizzi MAC anziché su porte.

Gli switch moderni, i quali possono lavorare anche a livello network, possono creare le VLAN basandosi anche sugli indirizzi IP.

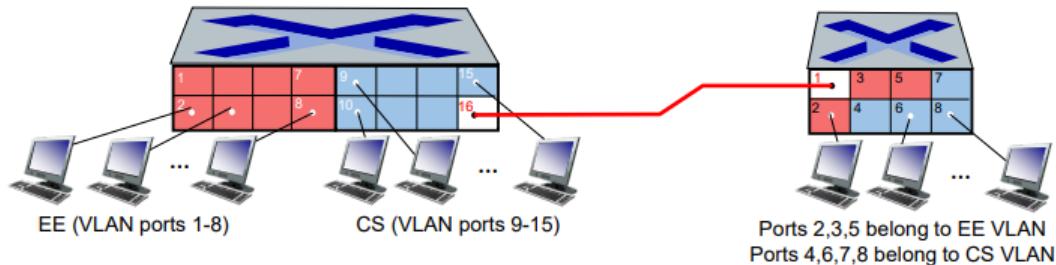
VLAN - Trunking

In seguito la tecnologia è stata sviluppata per collegare tra loro due switch come se fossero uno solo.

Il trunking può avere ripercussioni unendo le VLAN presenti su di essi (VLAN trunking), permettendo così la realizzazione di VLAN che si estendono nelle diverse parti di una rete aziendale, anche su scala geografica.

Questo mi permette di unire molteplici switch in un unico grande switch.

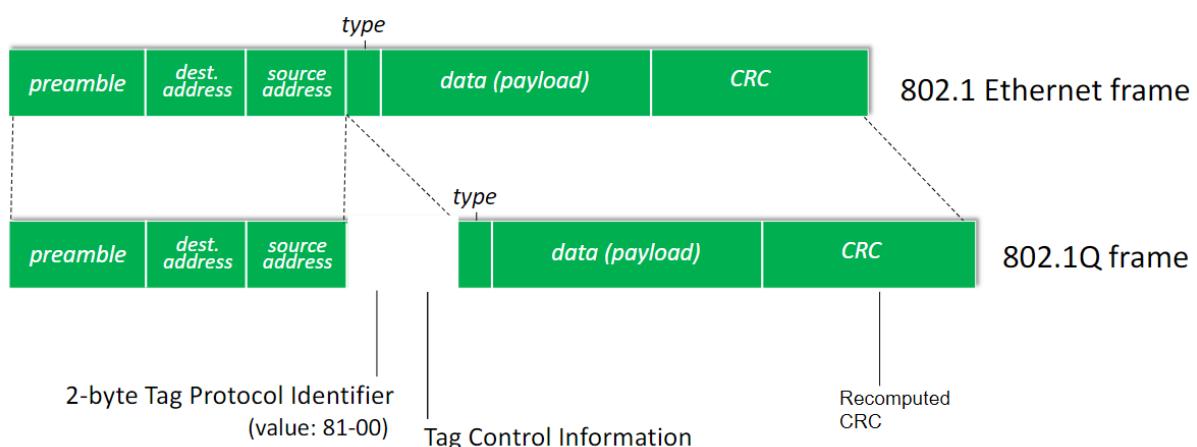
Il trunking può essere visto come “*operazione inversa*” alle VLAN, ossia unisco due switch diversi in uno solo.



VLAN - Formato Messaggio

Al normale messaggio di Ethernet si aggiungono 2 campi tra il campo *Source Address* e il campo *Type*:

- **Tag Protocol Identifier – 2 Byte**
 - L'appartenenza ad una VLAN è dettata dal protocollo encapsulato in 802.3.
 - Ad esempio, i pacchetti IP possono appartenere ad una VLAN diversa da quella usata dai pacchetti IPX.
- **Tag Control Information – 12 Bit**
 - Identifica la VLAN a cui appartiene il mittente.
 - Necessario per utilizzare le porte trunk.
 - Lo switch ricevente interpreterà questo campo come l'identificatore della VLAN.



Middle-Box

Dispositivi che svolgono il compito di tanti dispositivi messi assieme.

Ad esempio il “router domestico” fa tantissime cose diverse:

- Router.
- Switch.
- Access Point.

- Firewall.
- Server DHCP.
- Modem DSL.
- Server Proxy (alcuni).

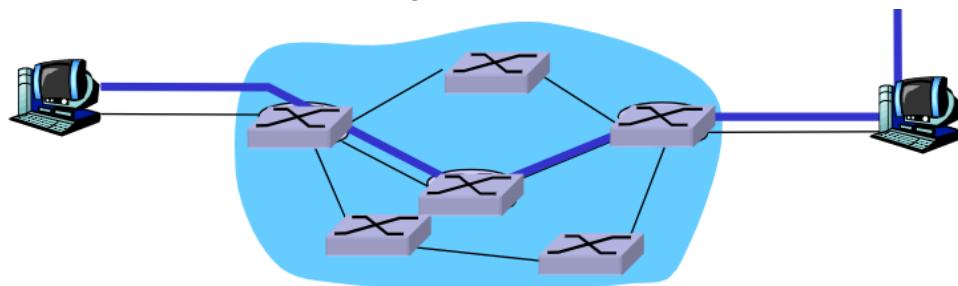
Wide-Area Packet Switched Networks - WAN

Cominciamo a trattare reti di tipo switched con estensione molto ampia (Regionale, Nazionale, Internazionale).

Ogni nodo della WAN ha un indirizzo fisico (diverso dagli indirizzi IP ma simile all'indirizzo MAC di ethernet) che lo identifica in modo univoco.

Il servizio in questa tipologia di reti si divide in 2 categorie:

- Servizio Connection.
- Servizio Connectionless (o Datagram).



Servizio Connection

Prima della trasmissione si **crea un circuito virtuale tra gli switch che verranno attraversati**, tutti i pacchetti seguiranno sempre lo stesso percorso.

ha come obiettivo il portare gli stessi vantaggi che si hanno in una connessione punto-punto.

L'ordine dei pacchetti viene mantenuto.

Alla fine il percorso virtuale viene cancellato.

La banda necessaria per il trasferimento lungo il circuito virtuale verrà pre-allocata nei vari link, quindi è possibile fare un controllo del flusso ed avere **prestazioni minime garantite**.

Virtual Circuit - Come si crea un Circuito Virtuale e come avviene il Trasferimento

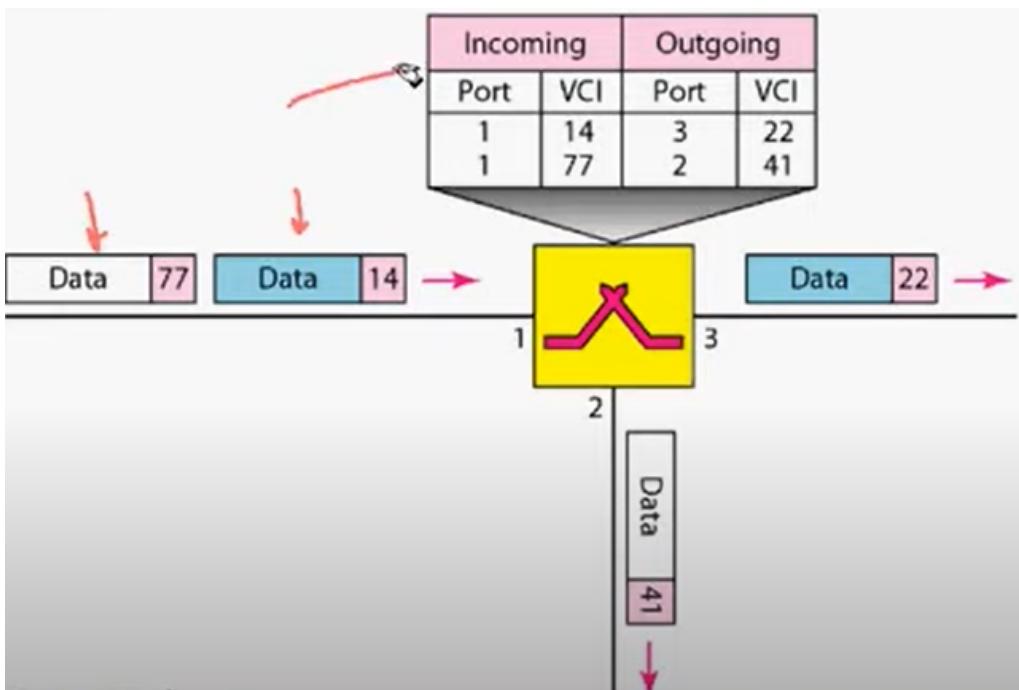
1. *Call Setup* → Procedura con cui si stabilisce il percorso:
 - a. Il nodo sorgente invia un *pacchetto di richiesta* in maniera Datagram.
 - Quindi senza circuito virtuale.
 - b. La richiesta attraverserà un certo percorso tra gli switch, man mano che gli switch vedono questo pacchetto se lo segnano.
 - c. Il destinatario accetta e rimanda indietro il pacchetto, gli switch faranno in modo che la risposta segua lo stesso percorso.
 - Ciò è possibile perché **gli switch si sono segnati il percorso** (ossia la porta da cui è entrato e quella da cui è uscito).
 - d. Gli switch del percorso vedendo che il pacchetto di richiesta torna indietro si segnano che la richiesta di circuito è stata accettata.
 - **Il percorso fatto è lo stesso che verrà usato dopo ovviamente.**
 - e. Quando la risposta arriva al nodo sorgente il Circuito è Completato.
2. *Trasferimento Dati*
 - a. Vengono inviati tutti i pacchetti uno dopo l'altro.

- b. Gli switch riconoscono se un pacchetto appartiene o no ad un circuito virtuale.

■ Se è questo il caso lo switch lo instrada nella porta associata al circuito, senza guardare l'indirizzo MAC di destinazione.

3. Tear Down

- a. Viene disattivata la connessione.
- b. Gli switch cancellano i record relativi al circuito virtuale.



“Tutti i pacchetti entranti dalla porta 1 con VC = 14 devo mandarli alla porta 3, ponendo VC = 22 nel pacchetto uscente”

Virtual Circuit - Cosa accade dentro lo Switch

Ogni circuito virtuale è identificato all'interno di uno switch da un numero, detto *VC Number*. Esso è stato assegnato dallo switch stesso durante la Call Setup.

Oltre alla Switching Table ora ho anche la “*Forwarding Table*”, dove al posto degli indirizzi MAC ho il VC Number.

Nella forwarding table c’è anche scritto il VC number che deve avere il pacchetto in uscita.

Servizio Connectionless (Datagram)

Ogni pacchetto è impacchettato e inviato individualmente.

Usato in Ethernet, gli switch non devono mantenere informazioni di stato, fanno solo il loro compito classico usando la tabella di switching.

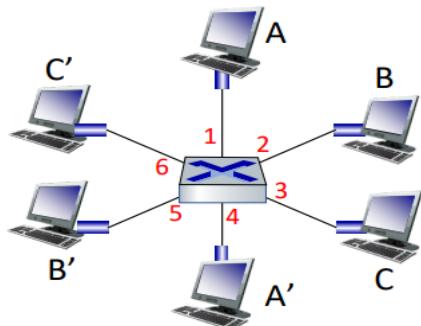
Arrivo “*in order*” non garantito.

Prestazioni minime non garantite.

I pacchetti sono inoltrati dagli switch tramite l’indirizzo MAC, quindi usando la switching table.

Il numero di possibili destinazioni è limitato, per rendere la tabella di switching di dimensioni limitate.

La tabella deve essere piccola perché lo switch deve essere una macchina veloce.



MAC address	Interface	TTL
A	1	60
A'	4	60

Servizio Connectionless in Reti Wide-Area

Nelle reti wide-area non funziona il concetto di switching table poiché in caso di tanti host essa si riempirebbe subito e renderebbe l'overhead introdotto (perché serve tempo a scorrere tutta la tabella di switching) dallo switch inaccettabile.

La soluzione è quella di generalizzare i record, quindi averne di meno ma che possano regolare più casi contemporaneamente.

Viene usato un altro tipo di Forwarding table, che usa un range di indirizzi.

“Se un indirizzo è in questo range, inoltra verso la porta 2”.

Questa cosa è molto efficace in reti con gerarchie di switch particolarmente grandi.

Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111	0
11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011011 11111111	2
otherwise	3

Internetworking

Ossia la comunicazione tra reti diverse.

Router

Lo scopo principale di un router è quello di interconnettere reti multiple e inoltrare i pacchetti destinati alle reti direttamente connesse o più remote.

E' considerato un dispositivo di livello 3 (network) perché la sua decisione primaria è basata sulle informazioni dei pacchetti IP, nello specifico l'indirizzo IP di destinazione.

Forwarding

Consiste nel prendere un datagram entrante da una certa porta di ingresso e inoltrarlo verso una certa porta di uscita.

I router di input (ossia quelli che ricevono il pacchetto) hanno il compito di inoltrare i pacchetti verso altri router con lo scopo di raggiungere la rete del destinatario.

Routing

Consiste nel determinare la route (*strada*) che i datagram percorreranno per andare dal router sorgente verso un qualsiasi router destinazione.

La route è determinata usando l'algoritmo di routing per la ricerca del cammino di costo minimo.

Piani di Lavoro del Router

In un router si possono distinguere due piani di lavoro differenti, ogni piano contiene diverse funzionalità che alla fine compongono tutta la funzionalità del router:

- **Data Plane**

- Questo piano controlla il processo di Forwarding.
- Ha il compito di incapsulare e decapsulare i datagram secondo il protocollo di livello 3 scelto.
- Ha il compito di applicare il NAT.
- Inoltre contiene funzioni per il “*pre-routing*”, ossia funzioni eseguite prima di determinare la route che dovrà seguire il datagram.

- **Control Plane**

- Questo piano controlla il processo di Routing.
- Questo piano può essere implementato in 2 modi:

- **Cooperativo - Traditional Routing Algorithm**

- Implementato nei normali Routers.
- Il routing è fatto in maniera distribuita e cooperativa, in cui tutti i routers si aiutano a vicenda.

- **Software-Defined networking (SDN)**

- Implementato in server remoti con cui i router si scambiano informazioni.
- Il routing è fatto in maniera centralizzata, dove un'unità centralizzata (tipo un server) che conosce la topologia della rete applica l'algoritmo di routing e calcola tutti i possibili percorsi di costo minimo.

Control Plane Cooperativo - “Traditional Routing Algorithm”

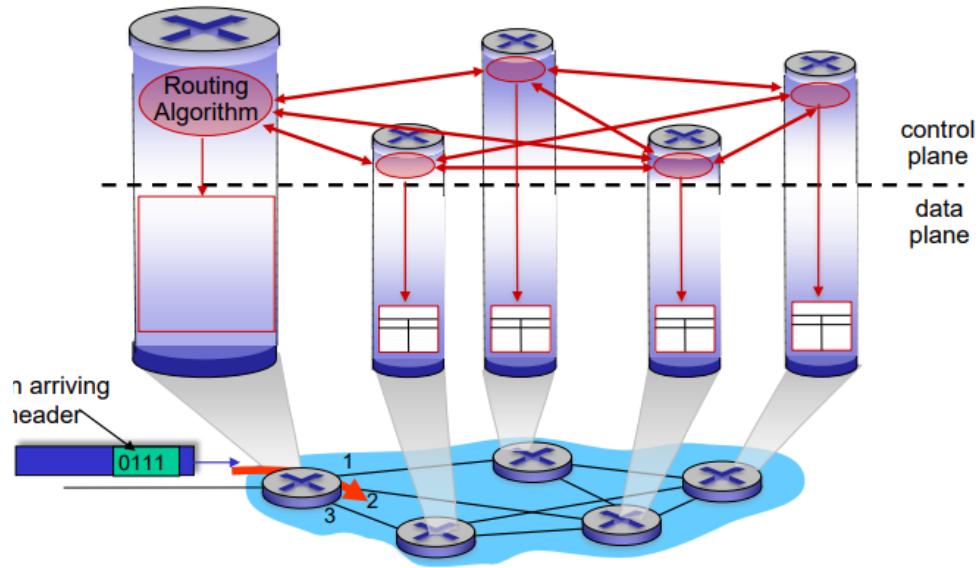
I router si scambiano informazioni tra loro per determinare i percorsi a costo minimo.

L'algoritmo viene eseguito in ogni router, quindi ogni router gestirà la sua tabella di forwarding.

Questo metodo è quello più classico ma anche quello più complesso poiché:

- Bisogna organizzare lo scambio di informazioni tra router.
- Porta maggiore carico sui singoli router, i quali oltre che a eseguire le normali funzionalità dovranno anche eseguire un algoritmo di routing.

Ma è anche quello più sicuro, infatti questa modalità non presenta un “*single point of failure*”.



Software-defined networking (SDN)

Il control plane viene staccato dai router.

I routers dovranno solo eseguire le funzioni di data plane.

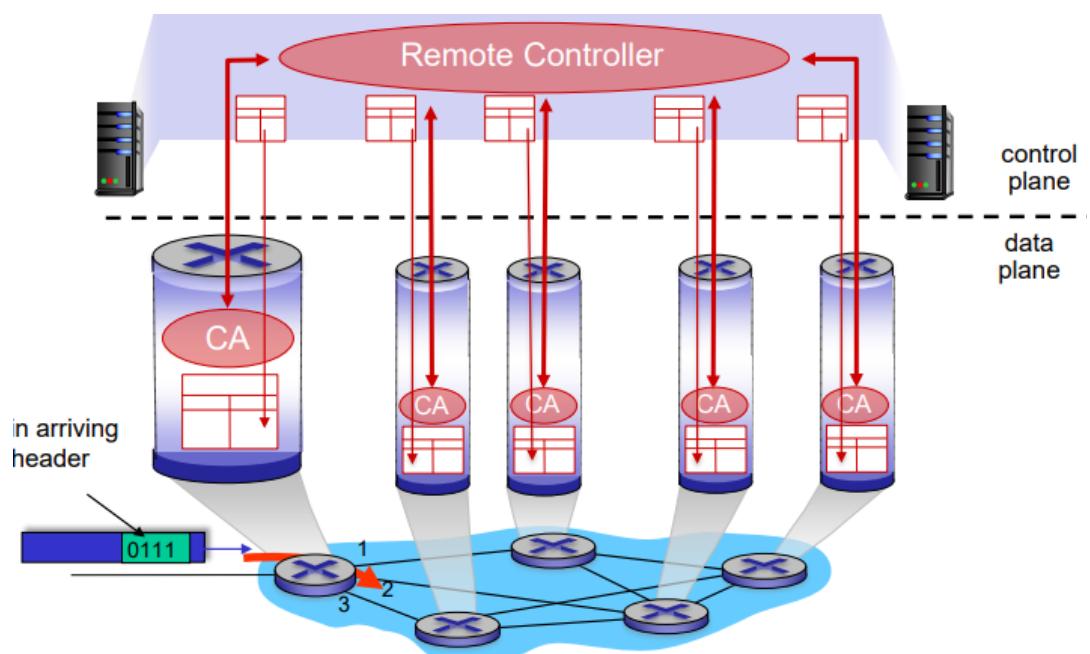
Un controllore remoto riceve tutte le info sulla rete, in base a queste costruisce l'albero dei cammini minimi per ogni evenienza.

Il controllore remoto ha il compito di fornire indicazione ai routers.

Le indicazioni vengono fornite in un modo molto semplice: il controllore remoto si occupa di aggiornare le tabelle di forwarding di ogni singolo router.

Il controller SDN potrebbe essere un dispositivo hardware fisico o una macchina virtuale.

Questa soluzione porta con sè grande flessibilità, poiché dato che il processo di routing è fatto via software da una unità centrale.



Network Service Model

Come vogliamo progettare l'invio dei datagram dal mittente al destinatario..

- **Individual Datagrams:**
 - L'utente vorrebbe la consegna garantita del singolo datagram.
 - L'utente vorrebbe che ogni consegna avvenga con un ritardo massimo di 40 msec.
- **Flow of Datagrams:**
 - L'utente vorrebbe la consegna dei datagram in ordine.
 - L'utente vorrebbe una banda minima garantita.

Modello “Best Effort”

- La consegna dei datagram non è garantita.
 - Alcuni datagram potrebbero non arrivare mai.
- Non è garantito un limite superiore al ritardo.
 - I tempi di consegna dipendono in maniera non prevedibile dal carico di rete in quel momento.
- Non è garantito il mantenimento dell'ordine dei datagram.
 - I pacchetti potrebbero arrivare a destinazione in disordine.
- Non si ha un throughput minimo garantito (*Banda end-end*).
- Molto semplice
 - Motivo del suo successo.

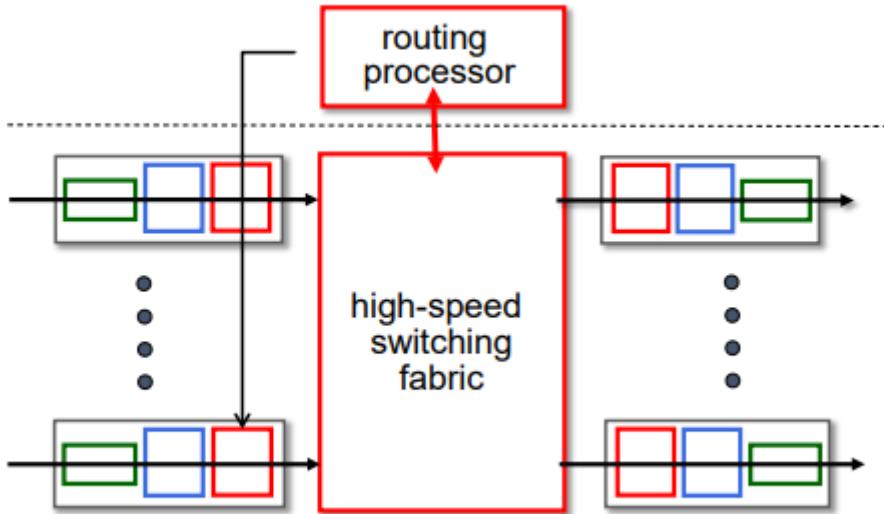
Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no
ATM	Constant Bit Rate	Constant rate	yes	yes	yes
ATM	Available Bit Rate	Guaranteed min	no	yes	no
Internet	Intserv Guaranteed (RFC 1633)	yes	yes	yes	yes
Internet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

Rinunciare a caratteristiche di controllo del traffico e della congestione garantisce d'altra parte dei vantaggi in termini di costi e prestazioni assolute, riuscendo comunque a mantenere una buona qualità se le capacità della rete eccede di buona misura le richieste effettive.

In caso di scarsa quantità di risorse, best effort diventa abbastanza inefficiente, ma in caso di risorse sopra la quantità minima comincia a difendersi bene.

Architettura dei Router

I router ricevono datagram da una porta di ingresso e la inoltrano in una porta di uscita. Le porte di uscita sono collegate a delle “reti di uscita”.



High-speed Switching Fabric

Implementazione del Data Plane.

Ha il compito di consultare le tabelle di forwarding e instradare i datagram provenienti dalle porte di ingresso nelle porte di uscita giuste.

Il forwarding deve essere veloce(nell'ordine di nano-secondi), perché i datagram in entrata sono molti e dovrebbero essere gestiti tutti, altrimenti l'overhead introdotto sarebbe troppo grande e problematico.

In genere è implementato in hardware per garantire la massima velocità possibile.

Routing Processor

Implementazione del Control Plane.

Ha il compito di scambiarsi informazioni riguardo la rete con altri router e riempire la tabella di forwarding applicando l'algoritmo di routing.

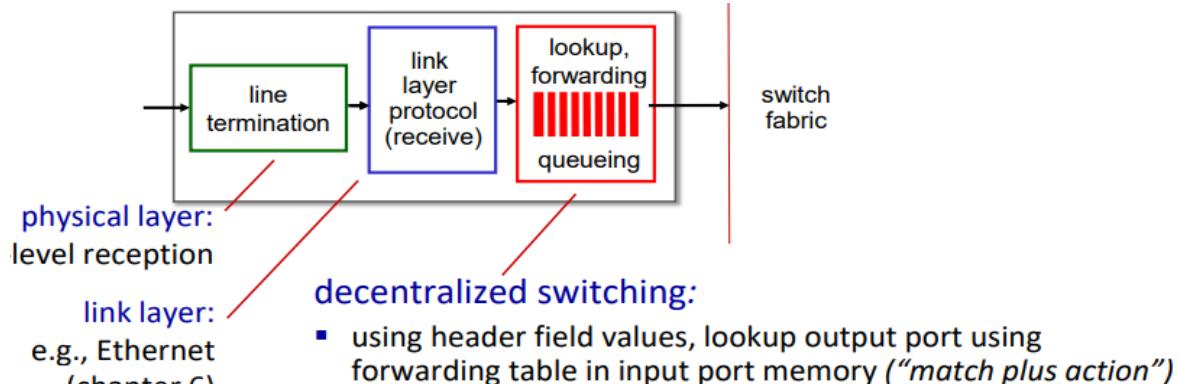
La parte di routing in genere è implementata in software.

In SDN al posto del routing processor abbiamo una macchina esterna che fa le stesse cose del Routing Processor (dal punto di vista del Data Plane).

Porte del Router

Ogni porta ha un suo “*mini-processore*”, quindi un router è di fatto un sistema multiprocessore.

Porte di Ingresso



La parte verde ha il compito di ricevere il segnale di ingresso (*bit-level reception*).

I datagram spesso sono molti e quindi serve memorizzarli (*store*) momentaneamente. Come nello switch, lo store porta con sé un possibile ritardo di accodamento. La politica di gestione della coda dipende dall'implementazione.

Quando il datagram arriva in testa alla coda verrà gestito dallo Switching Fabric e successivamente instradato verso la porta di uscita corretta.

Destination Based Forwarding

Il forwarding avviene guardando solamente l'indirizzo IP di destinazione.

Tabelle di Forwarding

Nella tabella non ho un record per ogni possibile host destinatario o uno per ogni rete destinataria, altrimenti (come nello switch) avrei tabelle enormi, quindi costose da mantenere e soprattutto da usare.

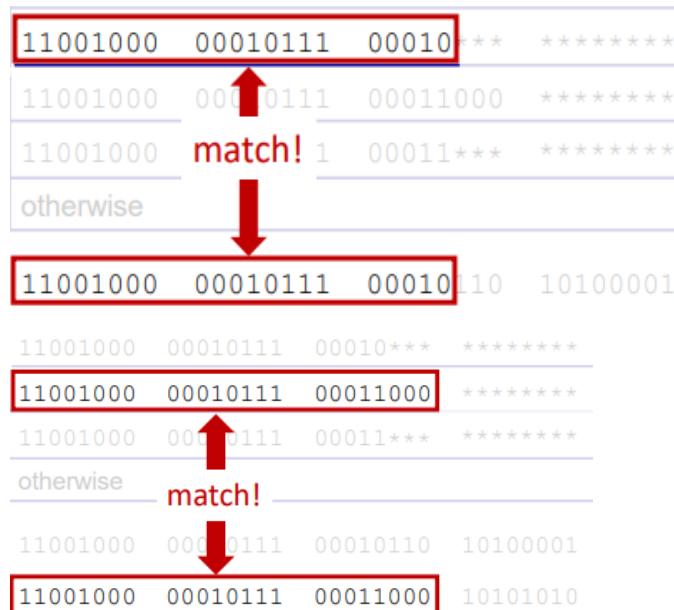
Per risolvere il problema, ho un record per ogni Range (*Forwarding Generalizzato*) di indirizzi IP.

Come nella tabella di switching, un record associa ad ogni range un'interfaccia di uscita.

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

Tabelle di Forwarding - Longest Prefix Matching

Quando si controlla la tabella di forwarding, si scorrono i record dall'alto verso il basso.
 Per ogni record si confronta l'indirizzo di destinazione con il range espresso nel record.
 Prendo il primo record che ha la più alta corrispondenza (il maggior numero di bit uguali) con l'indirizzo di destinazione.

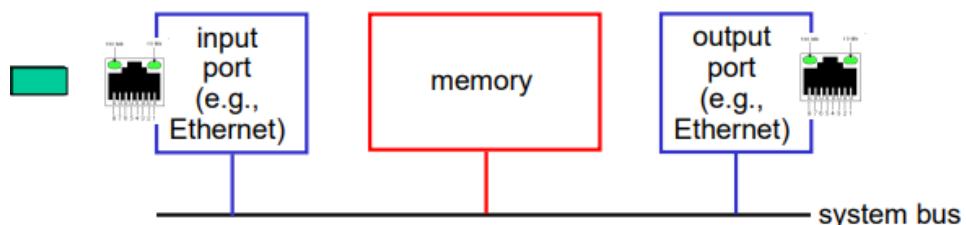


L'asterisco significa “qualsiasi bit”, è l'equivalente del “–” a Reti Logiche nelle mappe di Karnaugh.

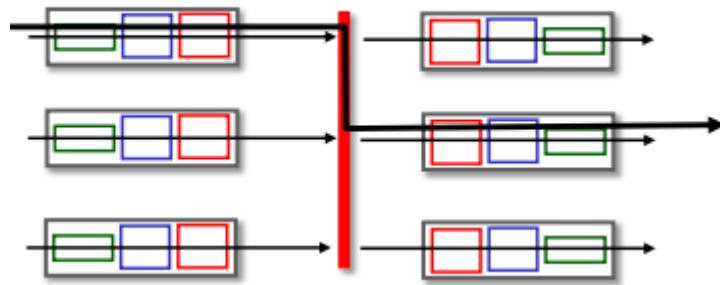
High-speed Switching Fabric di 1° Generazione (Memory)

Sostanzialmente un computer *general-purpose*:

1. La porta ethernet di entrata Riceve un Datagram.
2. La CPU del router controlla l'indirizzo IP di Destinazione nella tabella di forwarding contenuta nella propria memoria RAM.
3. Inoltra il datagram alla relativa porta ethernet di uscita.



Nel bus condiviso si trasferisce il datagram direttamente dalla porta di ingresso alla porta di uscita, senza avere una memoria.

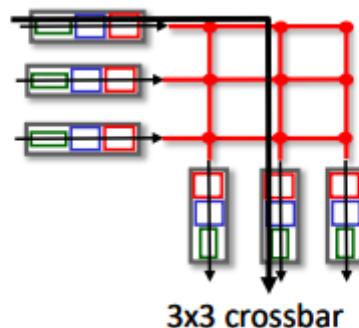


Throughput elevati ma limitati dal fatto che sono obbligato a gestire un datagram alla volta.

High-speed Switching Fabric di 3° Generazione (Crossbar)

Ho molteplici bus, quindi posso fare trasferimenti multipli simultanei ma non verso la stessa interfaccia di uscita contemporaneamente.

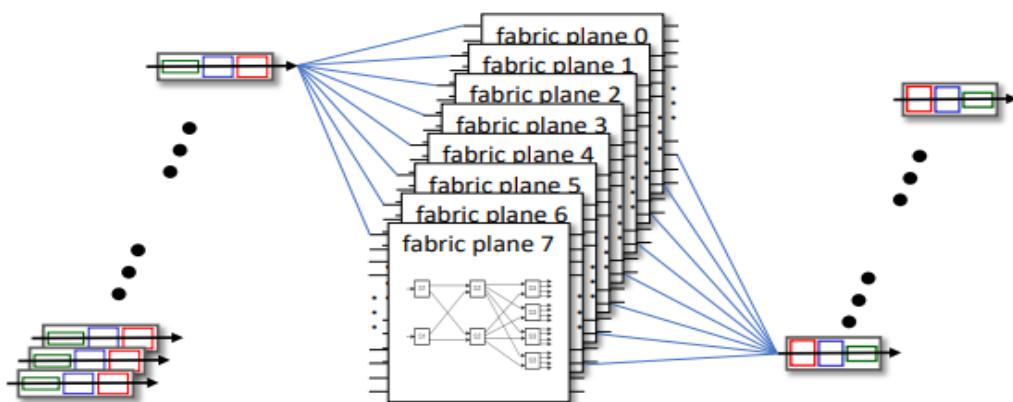
Però posso fare due trasferimenti da una singola interfaccia di entrata verso due interfacce di uscita differenti.



High-speed Switching Fabric di 4° Generazione (Interconnection Network)

E' l'architettura dei Cisco CRS.

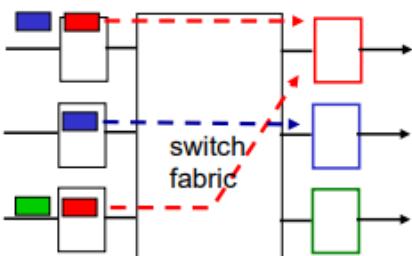
E' la soluzione alle limitazioni di banda imposte dall'uso di un bus condiviso.



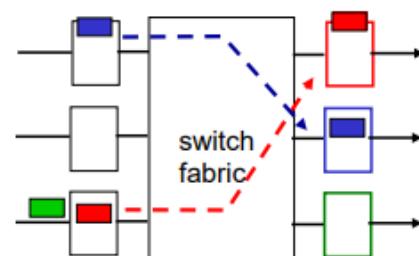
Ritardo di accodamento nelle porte di Ingresso

Se il fabric è più lento del rate di arrivo dei pacchetti nelle porte di ingresso comincerò a sviluppare un ritardo di accodamento, dovuto all'accodamento di pacchetti nelle code delle porte di entrata.

Si può verificare un *Head-of-Line-Blocking* (HOL): ossia i datagram accodati in cima alla coda non permettono a datagram in fondo alla coda di andare avanti.
HOL è un problema che si presenta quando la politica di gestione della coda è strettamente FIFO.



output port contention: only one red datagram can be transferred. lower red packet is *blocked*



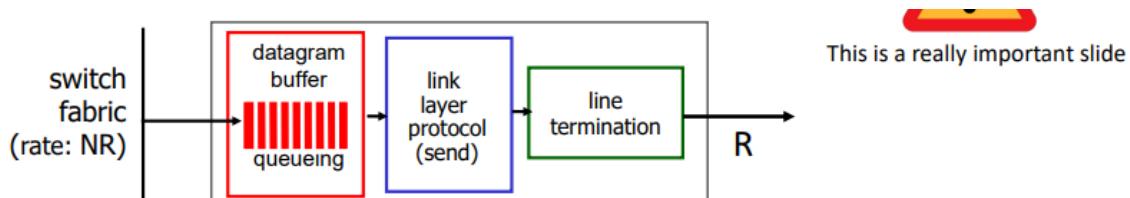
one packet time later: green packet experiences HOL blocking

In

Ritardo di Accodamento nelle porte di Uscita

Se il rateo di invio dei pacchetti dalle porte di uscita è più lento del fabric si svilupperà ritardo di accodamento, dovuto all'accodamento di pacchetti nelle code delle porte di uscita.

Serve una politica di scheduling nel caso in cui il buffer si riempia troppo.



- **Buffering** required when datagrams arrive from fabric faster than link transmission rate. **Drop policy:** which datagrams to drop if no free buffers?
- **Scheduling discipline** chooses among queued datagrams for transmission

- Datagrams can be lost due to congestion, lack of buffers
- Priority scheduling – who gets best performance, network neutrality

Internetworking - C_21

Algoritmi di Scarto dei Datagram nella Coda

Se la coda si riempie serve una politica che mi dica quale datagram va scartato a favore di quello appena arrivato.

- **Tail Drop**
 - Se il buffer è pieno, si butta l'ultimo datagram arrivato a prescindere da cosa contiene.

- **Priority**
 - Si **drops** il datagram a priorità minore nella coda.
 - Se quello appena arrivato è quello a priorità minore → Tail Drop.
- **Marking**
 - Approccio conservativo poiché il mio obiettivo è quello di **evitare di buttare via dei pacchetti**.
 - Il mio scopo sarà quello di **fare in modo che il buffer non si riempia mai al 100%**.
 - Dico ai mittenti che le code sono congestionate e quindi di **abbassare il rateo di arrivo dei pacchetti** altrimenti si è obbligati a scartare i pacchetti.
 - Il router invia una notifica ai mittenti e se capiscono abbasseranno il rateo di invio dei datagram (se sono benevoli, perché esistono attacchi con l'obiettivo di congestionare uno specifico router bombardandolo di pacchetti).
 - La notifica deve essere inviata preventivamente, ossia prima che il buffer si sia riempito.
 - Ad esempio quando il buffer ha raggiunto il 70% della capacità.
 - Le prestazioni ne risentiranno, inviare mark ai mittenti produce overhead.

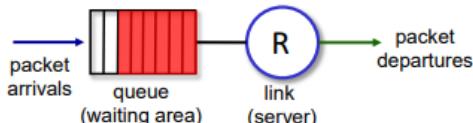
Algoritmi di Scheduling dei Datagram

Come la coda dei datagram viene gestita.

Packet Scheduling - FCFS

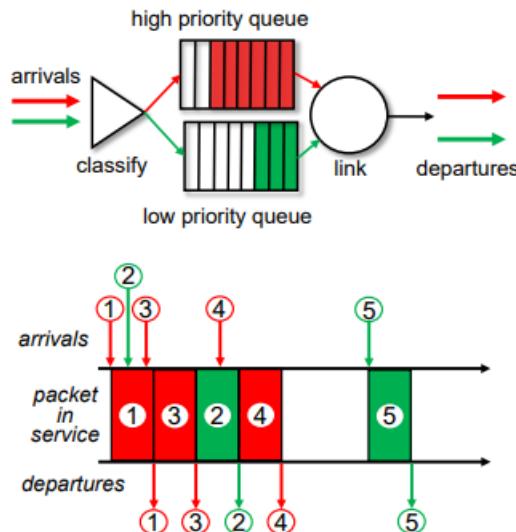
I datagram sono gestiti secondo l'ordine di arrivo, a prescindere da cosa contengono.

Ma i datagram spesso non sono tutti uguali (trasportano cose diverse che potrebbero avere esigenze riguardo al ritardo massimo) e potrebbe generare dei disservizi per certe applicazioni.



Scheduling Basato su Priorità

Il traffico in arrivo viene classificato (secondo i dettagli contenuti nel pacchetto) e accodato in opportune code di priorità.

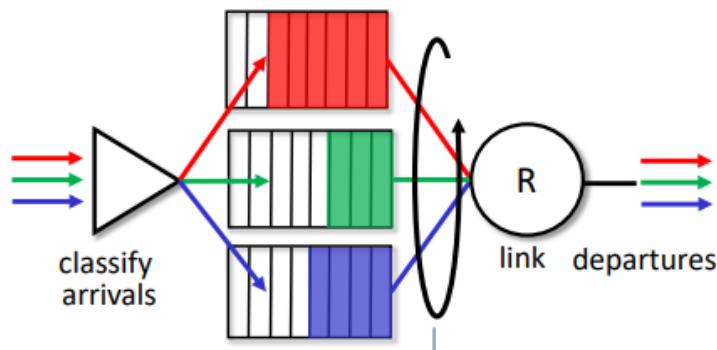


In questo caso i pacchetti rossi hanno un servizio privilegiato e verranno serviti per primi. I pacchetti verdi verranno gestiti se e solo se la coda dei pacchetti rossi è vuota.

Round Robin - Versione “Fair”

Divido i pacchetti in varie code (classi di servizio), ogni coda è servita con la politica round robin.

Quindi a turno verranno servite tutte.



Round Robin Generalizzato - “Weighted Fair Queuing”

Attribuisco a ogni classe di servizio un peso, in base al quale assegnerò una precisa quantità di servizio.

Ogni classe di servizio ha una sua Banda Minima Garantita.

Ad esempio:

- Coda 1 → 2 Turni quando ha il Token.
- Coda 2 → 1 Turni quando ha il Token.
- Coda 3 → 3 Turni quando ha il Token.

Network Neutrality

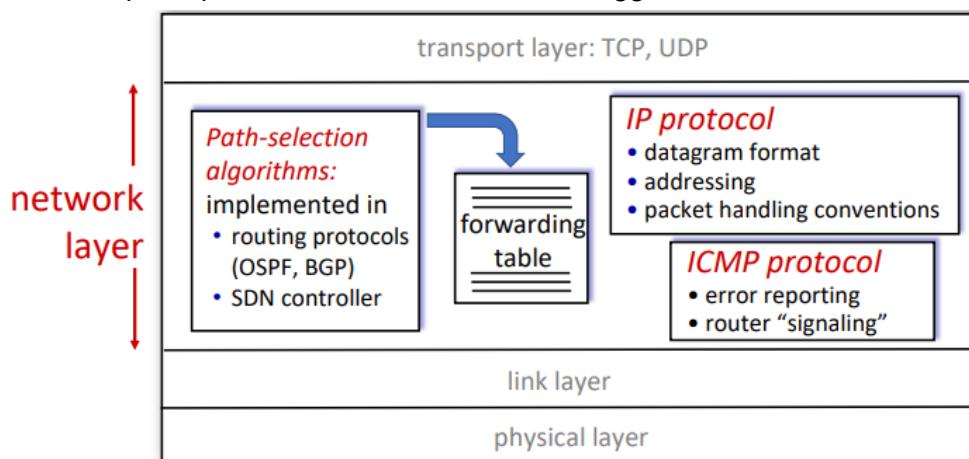
Ci sono varie definizioni, che potrebbero variare anche tra nazioni:

- **Definizione Tecnica**
 - Come gestire e come allocare le risorse quindi la Politica di Scheduling e il Buffer Management.

- FCFS rispetta la *Network Neutrality*, perché tutti i datagram hanno la stessa quantità di servizio, a prescindere da cosa contengono e da/a chi è mandato/destinato.
- Lo scheduling basato su priorità non rispetta la *Network Neutrality*, perché potrei avvantaggiare certi utenti (che magari pagano di più) a sfavore di altri.
- **Definizione Sociale / Economica**
 - Proteggere la libertà di parola, chi ha tante risorse economiche non deve poter impedire a chi non ne ha di parlare su internet.
 - Incoraggiare innovazione e competizione.
- **Definizione Legale**

Livello Network

Il suo scopo è quello di assicurarsi che i messaggi arrivino all'host destinatario.



Path-Selection Algorithms (Control Plane)

Definiscono i percorsi ottimali e installano le info necessarie nelle tabelle di forwarding. Possono essere Algoritmi di Routing Tradizionali o tramite un Controllore SDN.

Forwarding Table (Data Plane)

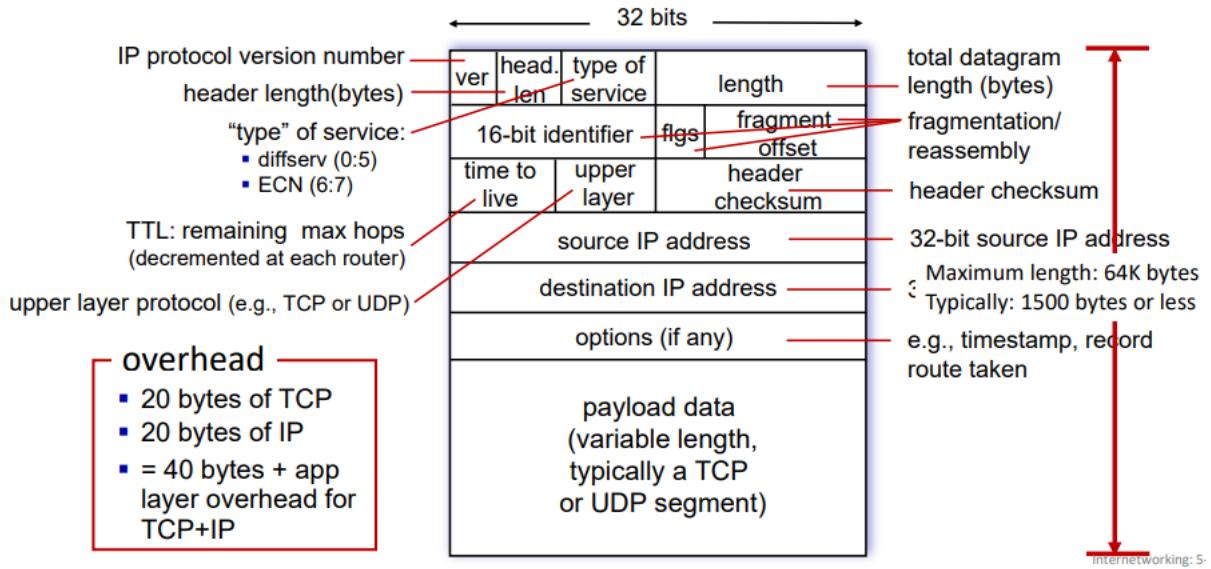
La forwarding table dice come devono essere inoltrati i datagram.

Protocollo IP

Ha vari obiettivi:

- Indirizzare gli host (*addressing*).
- Frammentazione.

Protocollo IP - Formato del Datagram



Intestazione del Datagram Formato IP

L'intestazione è allineata a parole di 32 bit.

L'intestazione è composta da **almeno 5 Parole da 32 bit**, la dimensione finale dipende dai campi opzionali che vengono considerati parte del payload.

- **1° Parola Lunga**
 - **Version - 4 Bit**
 - Specifica la versione di protocollo IP.
 - La più usata è IPv4, sta per essere soppiantata da IPv6.
 - **Head Length - 4 Bit**
 - Lunghezza dell'Header espressa in numero di Byte.
 - **Type of Service (ToS) - 1 Byte**
 - Specifico la classe di servizio (e quindi la priorità) a cui appartiene il datagram.
 - Specificare questa cosa non implica nulla, l'ultima parola ce l'ha la politica di gestione del router ,perché se il router applica FCFS questo campo è inutilizzato.
 - *diffserv* "Differenziato" → [0 : 5]
 - *ECN* → [6 : 7]
 - **Length - 2 Byte**
 - Dimensione espressa in byte complessiva (intestazione + payload) del datagram.
 - Dimensione massima del Payload → 64 Kb.
 - Di solito un pacchetto IP è lungo 1500 Byte.
- **2° Parola Lunga**
 - **Identificatore - 2 Byte**
 - Identifica il Datagram, serve a distinguerlo da tutti gli altri datagram.
 - **Flags - 3 Bit**
 - **Fragment Flag (fragflag) - 1 Bit**
 - 1 → allora c'è almeno un ulteriore frammento dopo di questo.

- 0 → allora questo è l'ultimo frammento della sequenza.
- **Fragment Offset - 13 Bit**
 - La posizione (espressa in numero di byte) del frammento nel datagram originale.
 - Serve a ricostruire il datagram originale.
- **3° Parola Lunga**
 - **TTL - 1 Byte**
 - Espresso in *Numero di Hop*, ossia il numero massimo di router rimanenti che il datagram può attraversare prima di essere scartato.
 - Il router lo decrementa al momento dell'instradamento, ossia subito dopo essere uscito dallo switching fabric.
 - Se il TTL (dopo il decremento) arriva a 0, il router invece che inoltrato lo scarta.
 - Il TTL evita che un datagram circoli all'infinito in caso di destinazione irraggiungibile.
 - **Upper Layer (Protocol) - 1 Byte**
 - Indica il protocollo di livello trasporto usato.
 - **Header Checksum - 2 Byte**
 - Informazioni del CRC sul blocco Header.
- **4° Parola Lunga - 4 Byte**
 - Indirizzo IP Sorgente.
- **5° Parola Lunga - 4 Byte**
 - Indirizzo IP Destinatario.
- **Campi Opzionali - ? Byte**
 - Informazioni che possono essere aggiunte se volute dal nodo sorgente o dall'applicazione.
 - Spesso sono considerate parte del payload a causa della loro dimensione variabile.
 - Ad esempio:
 - **Timestamp.**
 - **Record Route Taken.**
 - Dice il percorso di routers che il pacchetto deve percorrere.
 - Utile in caso di reti istituzionali.
 - In caso dell'esistenza di questo campo, i router non guarderanno la tabella di forwarding ma instradano a seconda del router voluto dalla route selezionata.
- **Payload - [0 , 64] KByte**
 - Contiene un pacchetto proveniente dal protocollo di trasporto.

MTU - Max Transfer Unit

Impone un limite massimo alla dimensione di un pacchetto in una specifica rete.

MTU dipende dalle caratteristiche della rete.

Tipicamente in una rete ethernet è di 1500 Byte.



Frammentazione

A causa del payload la dimensione del datagram non è prevedibile a priori.

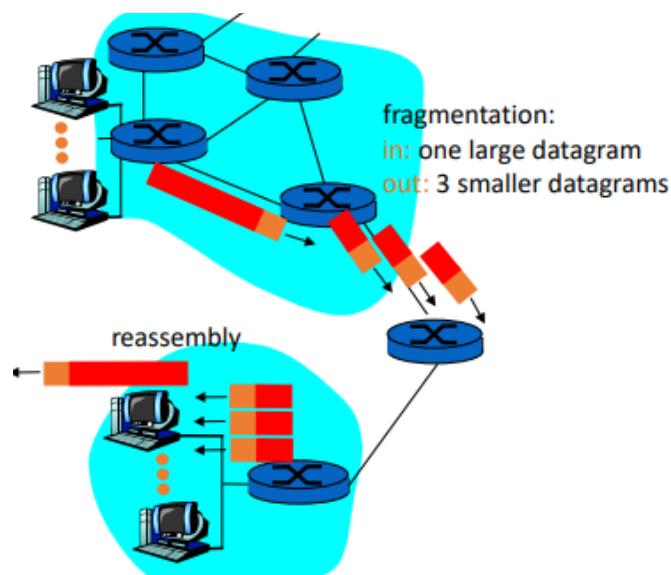
Il livello IP deve fare in modo che la lunghezza totale del datagram non superi:

- MTU.
- Le dimensioni massime del payload di un frame di livello data link.

La soluzione è quella di frammentare il payload del datagram e creare quindi molti piccoli datagram più piccoli e trattare ognuno di essi come un datagram indipendente dagli altri.

Verranno riassemblati nel protocollo IP dell'host destinazione.

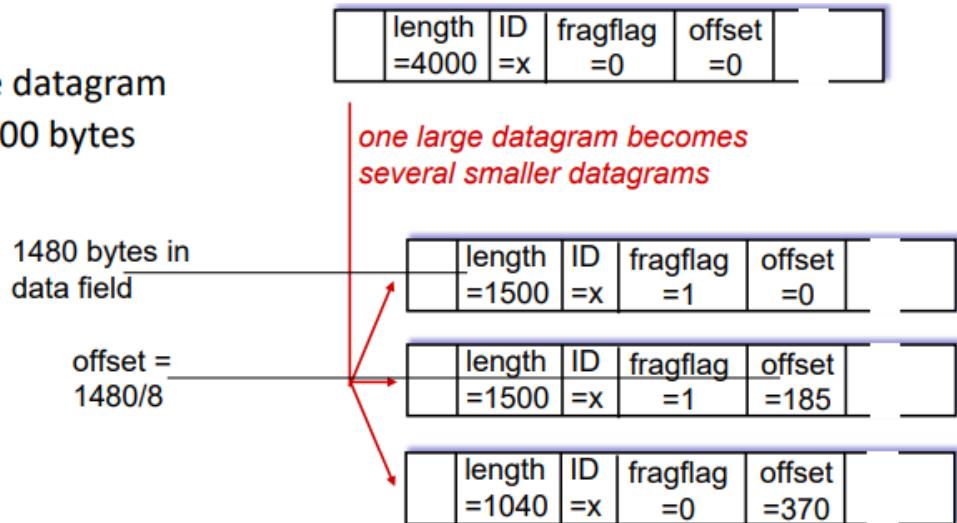
I pacchetti frammentati verranno memorizzati in un buffer in attesa che arrivino tutti.



Il datagram del protocollo IP avrà nel campo header un valore necessario a riassemblare il datagram una volta arrivato a destinazione, ossia la coppia `<offset , frag>`.

example:

- 4000 byte datagram
- MTU = 1500 bytes



$$MTU = \min\{ MTU_{link}, Dimensione_{max\ frame} \} = 1500\ Byte$$

Divido il payload del datagram in 3 frammenti, a ognuno applico una intestazione IP opportuna.

Poi invio ogni frammento al livello sottostante per trasformarli in frames.

La dimensione del frammento deve essere in modo tale: $Header + Payload \leq MTU$

Un Header IP è pari a 20 Byte (le opzioni sono considerate parte del payload).

Quindi il datagram originale va diviso in frammenti da massimo 1480 Byte, in modo da aggiungere poi l'intestazione IP del frammento e arrivare quindi a 1500 byte.

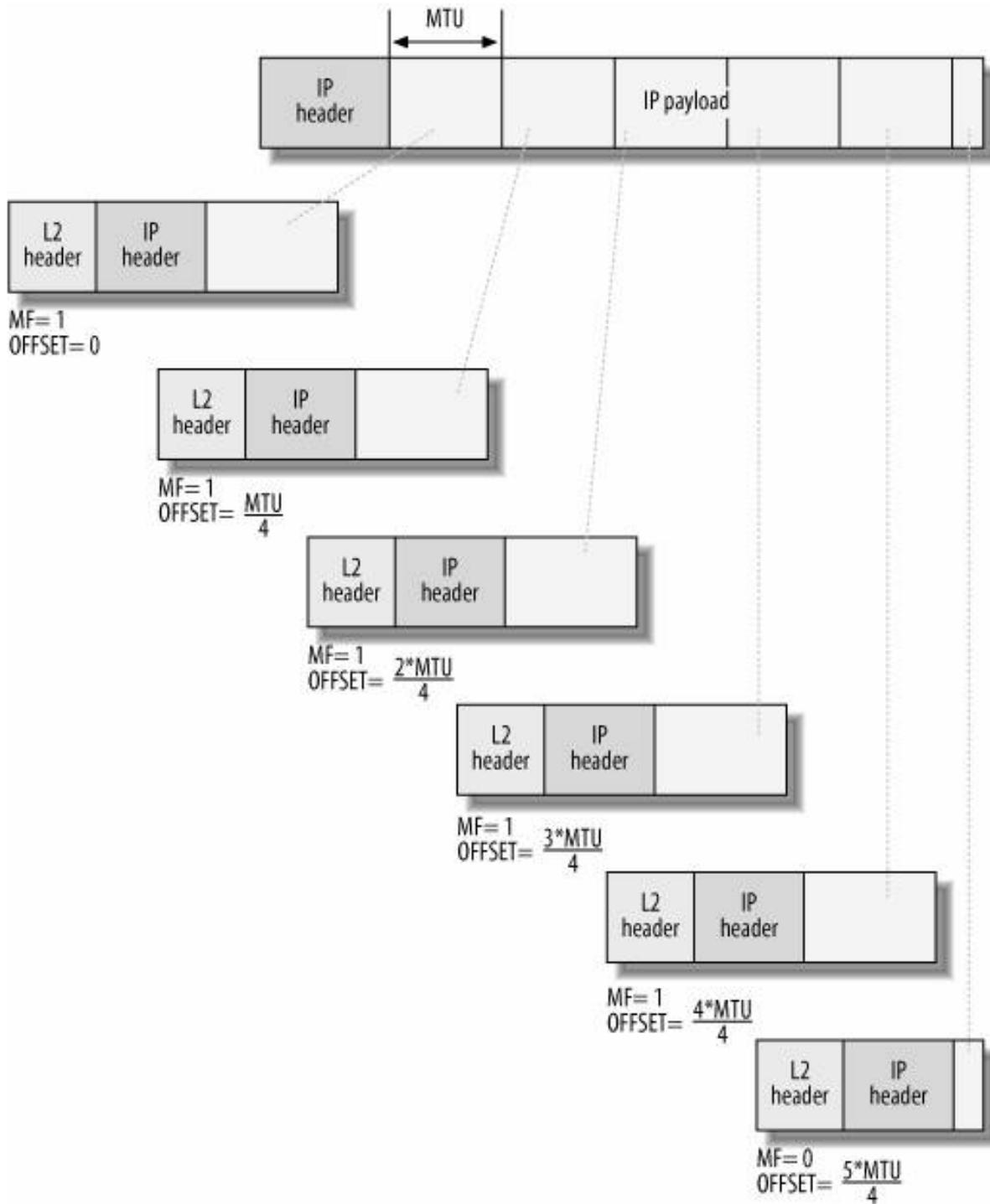
In questo caso, divido il datagram originale in: $\frac{Dim_{Datagram}}{MTU - 20} = \frac{4000}{1480} = 2.702 \rightarrow 3$ frammenti.

Al terzo frammento pongo *fragflag* a 0 per indicare che è l'ultimo frammento della sequenza.

A ogni frammento metto in offset opportuno, che indica la posizione del payload del frammento per ricostruire il datagram originale.

Il campo offset permette di riordinare i datagram a destinazione.

Se uno qualsiasi dei frammenti non arriva a destinazione butto via tutto e considero l'intero datagram mai arrivato a destinazione.



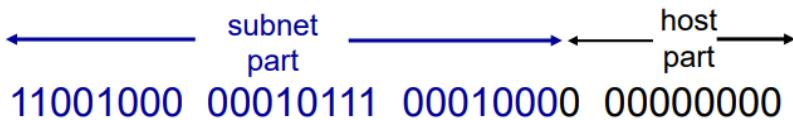
Indirizzamento IP - IPv4

Assegno ad ogni host una stringa di 32 bit, chiamata indirizzo IP.

In realtà l'indirizzo IP è proprio dell'Interfaccia di rete, un PC può avere multiple interfacce di rete ognuna con il suo indirizzo IP.

Un router è collegato ad almeno 2 reti, quindi ha almeno 2 interfacce di rete, e quindi almeno 2 indirizzi IP, uno per ogni rete.

L'indirizzo del router in una rete è chiamato *Default Gateway*.



200.23.16.0/23

L'indirizzo IP è costituito da due parti:

- **Subnet Part (Indirizzo di Sottorete | Prefisso):**
 - Identifica la particolare sottorete in cui si trova l'host.
 - La *subnet part* è la stessa per tutti gli host di una subnet.
 - *Indirizzo_ip / 24* → "I primi 24 bit indicano la sottorete a cui appartiene l'host".
 - Dato un indirizzo IP, un calcolatore estraе la *subnet part* eseguendo un **and bit a bit** con la *subnet mask*.
- **Host Part :**
 - Identifica il singolo host nella subnet indicata nella *subnet part*.
 - Dato un indirizzo IP, un calcolatore estraе la *host part* eseguendo un **and bit a bit** con la *subnet mask negata*.

Subnet

- 1° Definizione : Sotto-rete fisica di internet.
- 2° Definizione : L'insieme degli host con cui posso comunicare senza l'utilizzo di un router.

Classi di Indirizzi IP

Classe	Utilizzo dei Bit N → Network H → Host	Not. Dec. Puntata	Subnet Mask	Nº Host Disponibili
A	0NNNNNNN . HHHHHHHH . HHHHHHHH . HHHHHHHH	[0 - 127].H.H.H	255.0.0.0 / 8	16'777'214 = $(2^{24} - 2)$
B	10NNNNNN . NNNNNNNN . HHHHHHHH . HHHHHHHH	[128 - 191].N.H.H	255.255.0.0 / 16	65'534 = $(2^{16} - 2)$
C	110NNNNN . NNNNNNNN . NNNNNNNN . HHHHHHHH	[192 - 223].N.N.H	255.255.255.0 / 24	254 = $(2^8 - 2)$
D	1110XXXX . XXXXXXXX . XXXXXXXX . XXXXXXXX	[224 - 239].x.x.x	Non Definito	Non Definito
E	1111XXXX . XXXXXXXX . XXXXXXXX . XXXXXXXX	[240 - 255].x.x.x	Non Definito	Non Definito

Prima gli indirizzi erano divisi in 5 classi:

- Classe A → Parte Rete di 8 Bit , Parte di Host a 24 Bit.
- Classe B → Parte Rete di 16 Bit , Parte di Host a 16 Bit.
- Classe C → Parte Rete di 24 Bit , Parte di Host a 8 Bit.

- Classe D → Parte Rete di ? Bit , Parte di Host a ? Bit.
 - Riservata al MultiCast.
- Classe E → Parte Rete di ? Bit , Parte di Host a ? Bit.
 - Riservata a Usi Futuri e per ragioni sperimentali.

Ora gli indirizzi IP sono classless, quindi senza le regole espresse dalle 5 classi.

IP Addressing CIDR (Classless InterDomain Routing)

L'indirizzo IP non ha una classe e il numero di bit che indicano la rete possono essere scoperti solo tramite la subnet mask.

Configurazione di un Host

Come fa un host a prendere un indirizzo IP della propria subnet?

L'indirizzo IP non può essere dato a caso, perché deve essere unico nella subnet.

Ci sono 2 modi:

- **Hard-Coded (“All’Antica”)**
 - L'indirizzo IP viene impostato direttamente sull'host dall'amministratore in modo permanente.
 - Ad esempio in UNIX, questa cosa viene fatta scrivendo nel file di configurazione della scheda di rete.
- **DHCP (Dynamic Host Configuration Protocol)**
 - L'assegnamento degli indirizzi IP è in questo modo centralizzato e permette di evitare di dare indirizzi IP errati e di evitare collisioni di indirizzi IP.
 - Il server DHCP può essere configurato impostando il range di indirizzi che può gestire.
 - I dispositivi moderni spesso eseguono automaticamente la richiesta DHCP una volta che vengono accesi (i router moderni contengono anche un processo server DHCP), dando il bellissimo effetto chiamato “*Plug and Play*”.

DHCP - Dynamic Host Configuration Protocol

Gli host nella rete di casa mia sono configurati automaticamente grazie al protocollo DHCP.

Quale dispositivo fa questa cosa?

Nelle reti domestiche è il router, lì dentro c'è anche un processo server DHCP.

In teoria facciamo finta di avere una macchina server DHCP indipendente.

Il Protocollo DHCP conferisce agli host della rete una configurazione corretta per poter navigare, ossia DHCP invia agli host che ne fanno richiesta i 4 parametri essenziali per la navigazione.

L'host desideroso di configurazione invia la richiesta in broadcast una richiesta di DHCP nella speranza che un server DHCP nella rete gli assegni.

La richiesta di configurazione è fatta in *broadcast* perché:

- L'host richiedente non ha i 4 parametri.
- L'host richiedente non sa quale è l'IP del server DHCP.
- L'host richiedente non sa neanche come è fatta la rete e che tipo di indirizzo IP abbia

I 4 parametri che vengono passati dal server DHCP sono:

- Indirizzo IP.

- Mask.
- Default Gateway.
- DNS.

Un client si può connettere in DHCP solo se ha la possibilità di farlo (ad esempio un client potrebbe non avere un programma *client DHCP*) , altrimenti si deve configurare in modo manuale.

DHCP - Extra (No Lezione)

I server DHCP moderni non inviano solo i 4 parametri “classici” , ma ne inviano 8.

- Subnet Mask.
- Default Gateway.
- Indirizzo dei Server DNS.
- Nome di Dominio DNS di Default.
- Indirizzi Server WINS.
 - *Windows Internet Naming Service* → Un servizio di Name Server per NetBIOS
 - Servizio molto simile al DNS.
- Indirizzi Server NTP.
 - *Network Time Protocol (porta 123)*
 - Permette di sincronizzare gli orologi del computer.
- Indirizzi Server FTP.
- Parametri di Configurazione del Proxy WPAD

Protocollo DHCP

In genere tutti gli step del Protocollo DHCP avvengono in broadcast.

A volte, dipendentemente dal server DHCP, avviene anche in unicast.

Alla richiesta , dato che è broadcast , potrebbero rispondere più server DHCP (qualora ce ne fosse più di uno nella rete) , ognuno con la sua “offerta” , il client semplicemente continuerà l'esecuzione del protocollo con il server che desidera.

1. Discover :

- Il Client DHCP invia un pacchetto di tipo **DHCPDISCOVER** con:
 - src = 0.0.0.0 | 68
 - Indirizzo IP di “chi non ha un indirizzo”.
 - dst = 255.255.255.255 | 67
 - Indirizzo di Broadcast.

2. Offer :

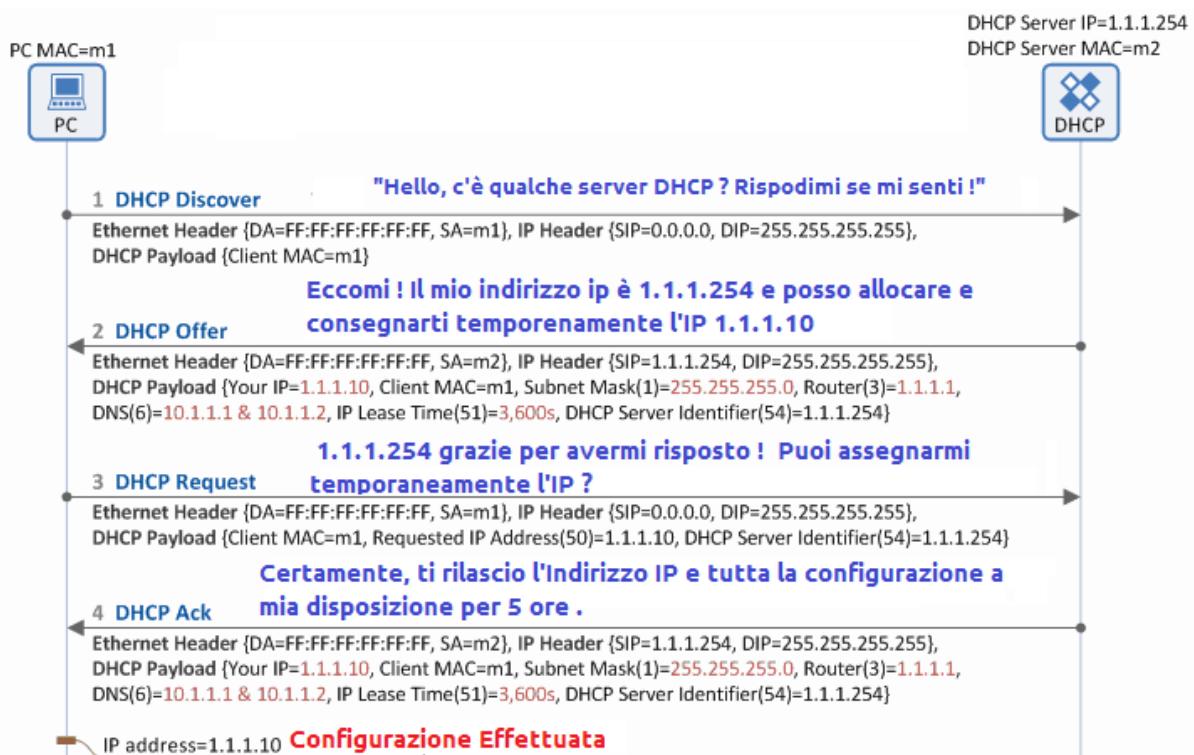
- Il Server DHCP risponde con un pacchetto di tipo **DHCPOFFER**:
 - src: 223.1.2.5 | 67
 - Indirizzo IP del Server DHCP.
 - dst: 255.255.255.255 | 68
 - yaddr : 223.1.2.9
 - Indirizzo IP Proposto al Client.
 - Lease Time
 - Scadenza della configurazione.

3. Request :

- Il Client DHCP accetta l'offerta inviando al server un messaggio di tipo **DHCPREQUEST**
 - src = 0.0.0.0 | 68
 - dst = 255.255.255.255 | 67
 - yaddr : 223.1.2.9 (*Il client chiede "formalmente" di poter usare questo indirizzo*)

4. Acknowledge :

- Il Server DHCP risponde con un messaggio di tipo **DHCPACK**, per autorizzare il client a usare yaddr.
 - src: 223.1.2.5 | 67
 - dst: 255.255.255.255 | 68
 - ID : x + 1
 - yaddr : 223.1.2.9



Note

- Porta = 68 (*Identifica il processo DHCP nel Client*).
- Porta = 67 (*Identifica il processo DHCP nel Server*).
- "Lease Time" è espressa in Secondi nella risposta del server DHCP.
- yaddr = "Your Address"

Quando nella rete è presente un server DHCP non è detto che tutti gli host siano configurati in DHCP

Ad esempio , ci sono macchine nella rete che potrebbero volere un indirizzo IP fisso.

Il server DHCP nella rete locale ha un range di indirizzi IP assegnabili , gli indirizzi IP che sono stati assegnati staticamente sono stati tolti da quel range di indirizzi.

Se un indirizzo è assegnato dal DHCP non può essere assegnato staticamente e viceversa.

Quindi di solito in ogni rete esiste un range di indirizzi assegnabili staticamente e un range di indirizzi assegnabili dinamicamente.

Domande DHCP

Specificare i messaggi scambiati

Una sessione DHCP è composta da 4 messaggi:

1. Il client invia in broadcast una DHCPDiscover dove avverte i Server DHCp presenti nella rete che ha bisogno di una configurazione.
 - Come indirizzo IP sorgente ha 0.0.0.0, perché l'host in questione non ha una configurazione di rete quindi è come se fosse nessuno.
 - Come IP destinatario ha 255.255.255.255 (broadcast) perché non sapendo gli indirizzi IP della rete non può sapere neanche quelli dei server DHCP.
2. I Server DHCP in ascolto vedono la richiesta e ognuno di essi invia una offerta di configurazione, questo messaggio è chiamato DHCPOffer.
 - Come destinatario è ancora indicato il broadcast, ma stavolta come sorgente si ha l'indirizzo IP del Server DHCP.
3. Il Client vede le offerte, ne sceglie una e invia in broadcast un DHCPRequest (indicando in un apposito campo l'indirizzo IP del server DHCP da cui prenderà la configurazione, in modo che il server DHCp in questione possa capire che è stata accettata la sua offerta).
4. Il server DHCP invia un DHCPAck per confermare la cosa e indicando la durata della configurazione.

Come avviene la comunicazione

Tutta in broadcast, infatti i messaggi DHCP hanno come destinatario 255.255.255.255.

Livello in cui opera tale protocollo

DHCP è una applicazione che risiede a livello applicativo, quindi il livello 7.

ISP - Internet Service Provider

A noi fruitori, gli indirizzi IP sono forniti dai Provider.

I provider offrono il servizio di connettività, ed esso è essenzialmente l'assegnazione di un blocco di indirizzi IP da usare nella propria subnet.

Esempio

Fly-By-Night (*Flai Bai Nai*) ha un blocco di indirizzi IP che partono da: 200.23.16.0 / 20

ISP's block	<u>11001000 00010111 00010000 00000000</u> 200.23.16.0/20
-------------	---

Fly-By-Night può dividere questo blocco in varie reti da assegnare ad ogni organizzazione a cui offre il servizio di connettività.

Supponiamo di dividere il blocco in 8 sottoblocchi da distribuire a 8 organizzazioni (la parte sottolineata è la subnet part):

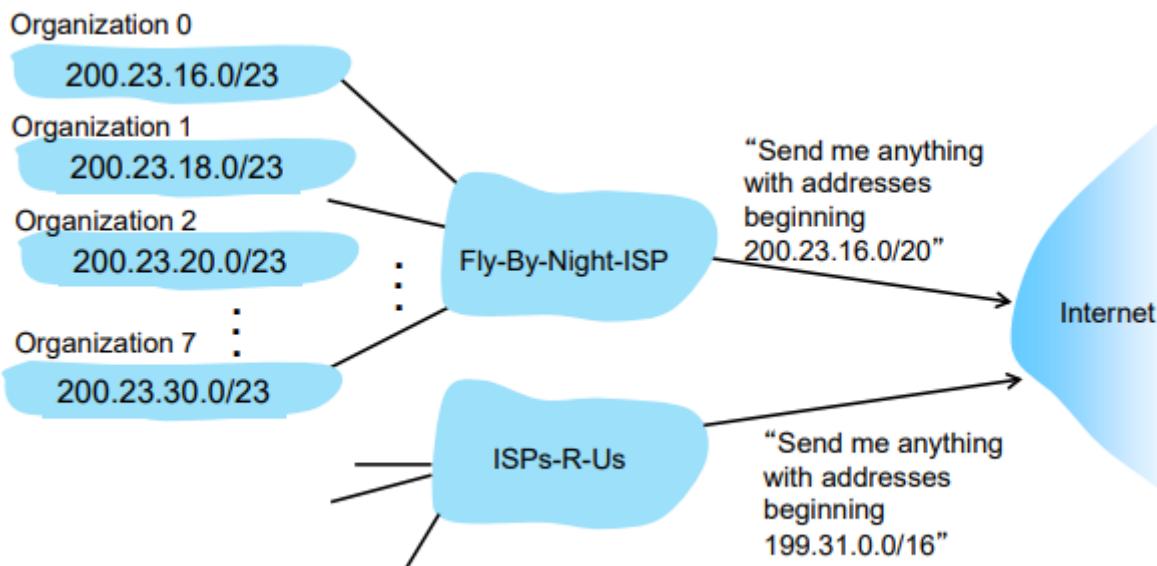
Organization 0	<u>11001000</u>	<u>00010111</u>	<u>00010000</u>	00000000	200.23.16.0/23
Organization 1	<u>11001000</u>	<u>00010111</u>	<u>00010010</u>	00000000	200.23.18.0/23
Organization 2	<u>11001000</u>	<u>00010111</u>	<u>00010100</u>	00000000	200.23.20.0/23
...
Organization 7	<u>11001000</u>	<u>00010111</u>	<u>00011110</u>	00000000	200.23.30.0/23

Le sottoreti delle 8 organizzazioni saranno connesse tutte alla rete di Fly-By-Night. Fly-By-Night avrà ovviamente un router di frontiera con la forwarding table che contiene tutte le informazioni che riguardano le sottoreti clienti di Fly-By-Night.

I core router di internet non necessitano di tutte le informazioni sulle caratteristiche delle reti delle organizzazioni.

Semplicemente sanno il range di indirizzi che gestisce il router di *Fly-By-Night*.

Poi penserà lui a smistare i pacchetti tra le reti delle organizzazioni.





Nella figura a Sinistra: Prima del "Fuck By Night"

Nella figura a Destra: Dopo il "Fuck By Night"

ICANN - Internet Corporation for Assigned Name and Numbers

Assegna gli indirizzi ai provider di altissimo livello e gestisce anche la zona root dei DNS inclusa la gestione dei TLD.

ICANN collocò l'ultimo blocco di indirizzi IPv4 in un RRs nel 2011.

IPv6 ha uno spazio di indirizzamento di 128 bit invece che 32.

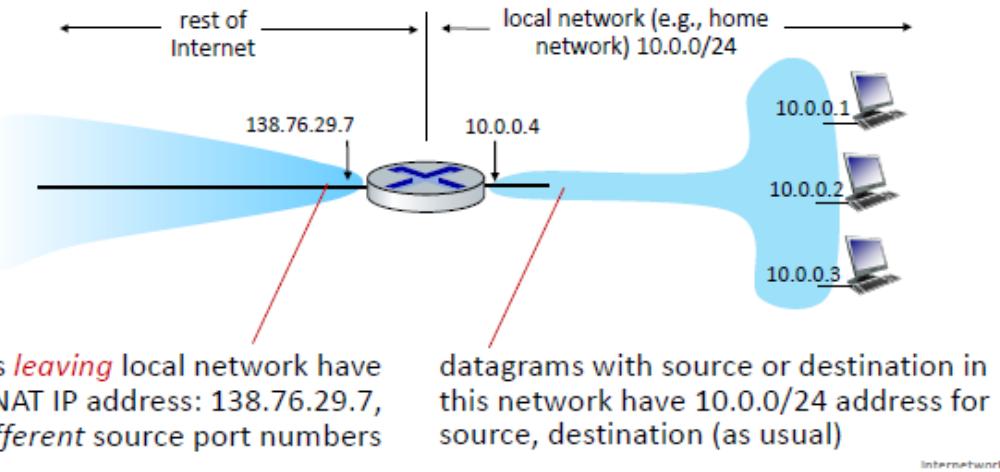
ICANN alloca gli indirizzi IP mediante 5 Registri Regionali (RRs), i quali possono a loro volta allocare dei Registri Locali.

Protocollo NAT - Network Address Translation

Tecnica per risparmiare indirizzi complementare e utilizzabile congiuntamente al DHCP.

Gli indirizzi IPv4 sono pochi e vanno sfruttati al meglio.

Assegno un unico indirizzo IP ad un gruppo di host appartenenti alla stessa sottorete.



Supponiamo che la rete locale abbia indirizzo di rete pari a $10.0.0.x / 24$

Con il protocollo NAT, gli host privati avranno *Indirizzi IP Privati*, i quali saranno univoci soltanto all'interno della subnet.

Gli host di una rete privata non possono comunicare direttamente con l'esterno perché il loro indirizzo IP non è valido all'esterno (non è univoco) della rete e quindi i loro datagram non sono instradabili all'esterno della rete locale.

Uscita - Source NAT

Quando un datagram proveniente da un host privato deve oltrepassare il gateway e quindi andare in un'altra rete, il router della rete privata sostituisce l'indirizzo IP Privato dell'host ($10.0.0.2 / 24$) con l'indirizzo IP pubblico del Router della rete privata ($138.76.29.7 / 24$) e quindi il datagram infine risulta instradabile.

$< \text{IP Privato , N Porta Sorg} > \rightarrow < \text{IP Pubblico Router , N Porta Est} >$

Questa cosa viene fatta per tutti gli host della rete.

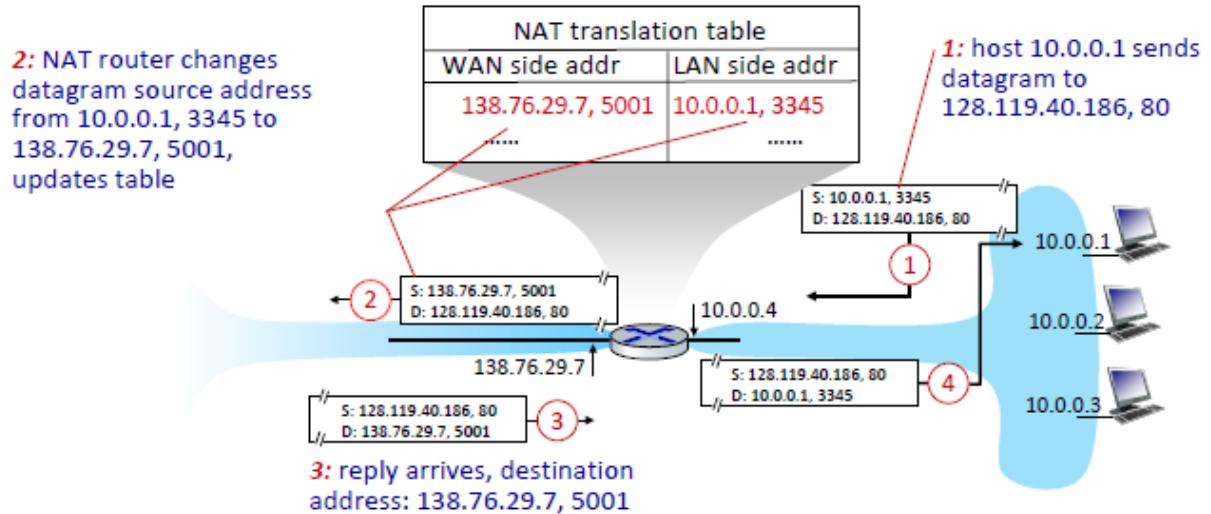
Gli host delle reti esterne vedranno solo l'indirizzo IP "esterno" della nostra rete privata (L'indirizzo IP "esterno" è dato dallo ISP e risulta effettivamente unico nel mondo) che in questo caso pari a $138.76.29.7 / 24$.

Gli host della rete privata possono inviare un datagram all'esterno inviando un pacchetto al default gateway (il quale in questo esempio ha indirizzo pari a $10.0.0.4 / 24$).

Come fa il router a capire a chi appartiene la eventuale risposta al datagram precedentemente inviato?

Un router che esegue il NAT è detto "Router-NAT".

Nota che il NAT nonostante stia nel router è in grado di leggere e cambiare i numeri di porta, rendendo di fatto il router un dispositivo di livello 4.



1. Supponiamo che 10.0.0.1 con porta sorgente 3345, debba inviare un datagram a 128.119.20.186 con porta destinazione 80.
2. Il datagram non trova destinatario all'interno della rete e quindi viene inviato al Default Gateway e quindi al Router della rete privata.
3. Il Router riceve il Datagram ed esegue il *Modulo NAT / PAT*.
 - a. Prima di inviarlo sostituisce l'indirizzo IP privato del mittente con l'indirizzo IP pubblico della Rete ossia 138.76.29.7.
 - b. La corrispondenza viene segnata nella Tabella di NAT.
 - c. **Per evitare ambiguità, a ogni host privato si assegna un nuovo numero di porta sorgente, detto numero di porta esterna** (in questo caso 5001) che lo andrà a differenziare da tutti gli altri host della rete privata nella colonna a sinistra della tabella di NAT.
4. Il Server esterno elabora la richiesta e invia una risposta con indirizzo destinatario pari a quello del Router e con porta 5001 (*quindi il numero di porta sorgente e l'indirizzo ip privato dell'host rimangono ignoti al server*).
5. Quando la risposta arriva al Router della rete privata quest'ultimo consulterà la tabella di NAT e quindi sostituisce il numero di porta 5001 e l'indirizzo IP pubblico con il numero di porta sorgente e l'indirizzo IP privato dell'host.
6. Infine inoltrerà la risposta all'host privato..

Gli host della rete non si accorgono dell'esistenza del NAT e non sanno che il loro indirizzo IP non è utilizzabile all'esterno.

Effettivamente il NAT cambiando il numero di porta esegue una modifica di livello 4 (livello trasporto) violando la stratificazione a livelli, quindi il NAT teoricamente non è un protocollo di livello 3.

Il NAT viola il end-to-end argument.

Il problema più grosso è : “Posso mettere un server nella mia rete privata con un Router-NAT?”.

I server devono essere accessibili dall'esterno e con il NAT nudo e crudo non si può fare , perché gli host dentro alla rete sono nascosti (non hanno un indirizzo IP pubblico) dall'esterno.

L'unico indirizzo IP valido all'esterno è quello del router.

La soluzione a questo problema la vedremo dopo.

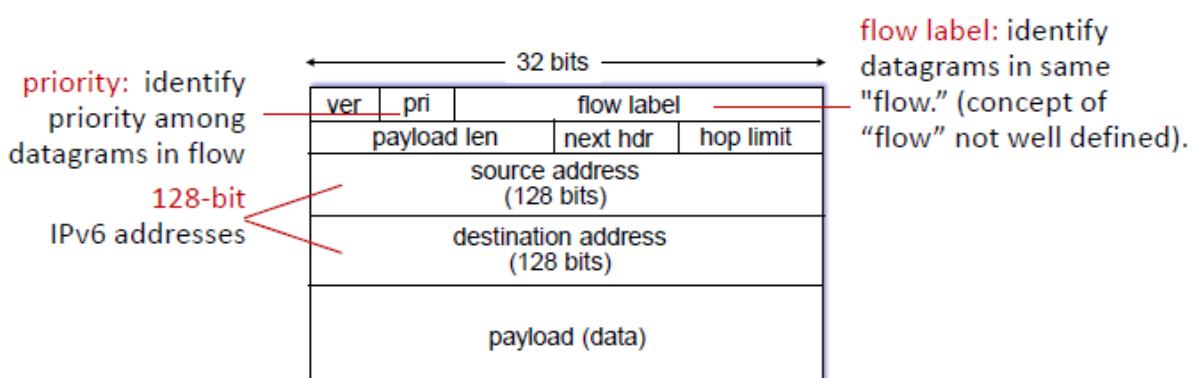
Domande NAT

- A cosa serve?
 - Il suo scopo è quello di permettere la navigazione ad un insieme di Host appartenenti ad una rete privata utilizzando un unico Indirizzo IP pubblico.
 - Gli host all'interno della rete useranno indirizzi IP non validi all'esterno e unici SOLO all'interno della subnet.
- Per quale scopo è stato pensato?
 - Il motivo per cui è nato era per risparmiare indirizzi IPv4.
- Limiti di NAT?
 - Il router che effettua al NAT ha limite massimo connessioni dovuto alle porte del router, è un router nel mezzo per cui violata connessione end-to-end, la soluzione migliore è usare IPv6.
- Come si implementa?
 - In genere il NAT è implementato nel router di frontiera, il quale oltre che a fare le normali mansioni di routing e forwarding dovrà anche applicare il NAT.
 - Si implementa mediante appositi comandi Linux.
 - Oppure si può implementare mediante OpenFlow sul router di frontiera.
- Situazione particolare in cui questo meccanismo potrebbe non bastare e come si risolve?
 - Se un Host esterno vede solo l'indirizzo IP del router esterno, ossia l'indirizzo pubblico, se volesse collegarsi ad esempio con un server Web interno alla mia rete non potrebbe.
 - Si può risolvere aggiungendo una regola specifica per quel determinato host della mia rete interna, ossia il router-nat esegue un **Destination NAT** e quindi inoltrerà i datagram verso l'host privato effettuando opportuni cambiamenti al datagram.

IPv6 - Protocollo IP Versione 6

IPv5 è stato un fallimento assoluto e infatti nessuno se lo ricorda.

IPv6 non ancora del tutto implementato.



- ver
 - Dice la versione del protocollo, in questo caso ci sarà il valore 6.
- flow label
 - Identifica i datagram nello stesso flusso, ossia 2 datagram con lo stesso flow label appartengono allo stesso flusso.
- pri
 - Specifica la priorità del datagram ricevuto.
 - Serve per applicare politiche di gestione delle code nel router.
- payload len
 - Rappresenta la lunghezza del payload in numero di byte.
- next hdr
 - Protocollo di livello superiore a cui è destinato il datagram.
- hop limit
 - Numero massimo di hop che può attraversare il pacchetto , equivalente del TTL in IPv4.

In confronto a IPv4:

- Non c'è il checksum.
 - Risultava un processo lentissimo per il router, perché anche solo decrementare il TTL comportava ricalcolare il checksum a ogni router.
 - Il checksum è affidato a protocolli superiori, lasciando quindi il controllo del checksum alla macchina destinataria.
- Il frammento di IPv6 è più lungo di quello di IPv4 ma ci sono molti meno campi.
- No Frammentazione (Più o Meno).
 - Frammentare fa perdere tempo, se il payload è troppo grande il router si rifiuta di instradare.
 - Quindi la frammentazione, se serve, avviene a monte.
 - Purtroppo con l'avvento dello IoT questa cosa risulta un limite molto scomodo, infatti è stato introdotto un protocollo IPv6 che le reintroduce.
- Non ci sono i campi Options.
 - Non si usavano mai e causano un grosso problema, ossia la lunghezza variabile dell'intestazione, la quale comportava una lettura del router nel campo field e quindi ulteriore perdita di tempo.

Transizione da IPv4 a IPv6

Non tutti i router possono essere aggiornati simultaneamente.

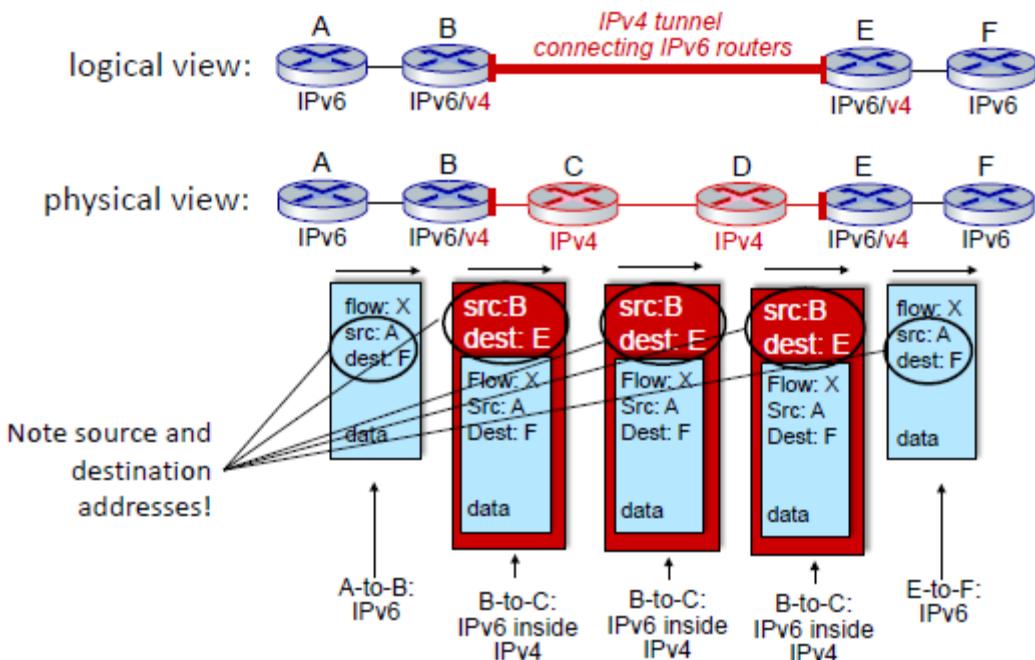
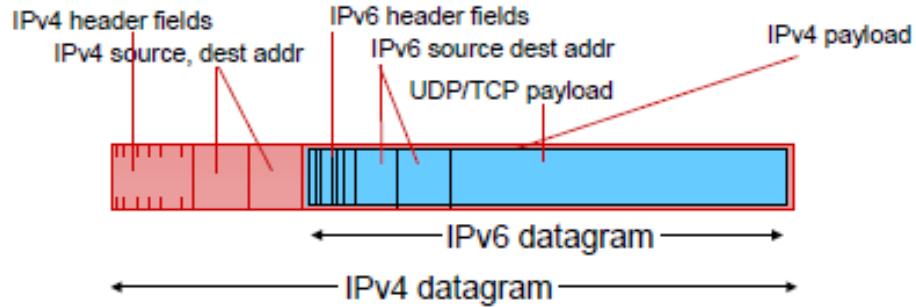
Quindi ci sarà un periodo transitorio in cui vengono usati sia IPv4 che IPv6.

Ci sono router che riescono a gestire IPv4 e IPv6 contemporaneamente (quando ricevono un datagram riconoscono se è IPv6 o IPv4), ma non tutti i router sono in grado di farlo, alcuni possono gestire solo IPv4 o solo IPv6.

Tunneling

Tecnica che mi permette di utilizzare il protocollo IPv6 attraverso una catena di router che usano solo il protocollo IPv4.

Il datagram IPv6 viene inserito nel payload di un datagram IPv4, il quale potrà essere gestito dai router che usano solo IPv4.



1. Il router B (che gestisce IPv4 e IPv6 contemporaneamente) ha a destra dei router che usano solo IPv4, quindi il router B (per instradare un datagram IPv6 lungo questi router) effettua il *tunneling* e incapsula il datagram IPv6 in un datagram IPv4.
 - a. Ossia il datagram IPv6 diventa il payload del datagram IPv4.
2. Il datagram IPv4 ha come mittente il router B e come destinatario il router E, il quale può gestire IPv4 e IPv6 contemporaneamente.
3. Il datagram IPv4 attraversa la catena di router che usano esclusivamente IPv4.
4. Quando il datagram arriva al router E verrà eseguita una *decapsulation* e dal datagram IPv4 verrà estratto il datagram IPv6.
5. A quel punto il router E instrada il datagram IPv6 verso la destinazione originale.

Notifiche del Router

Il Router invia notifiche agli host tramite il protocollo ICMP.

Protocollo di supporto a IP - ARP - Address Resolution Protocol

ARP è un protocollo che sta tra il livello 3 e il livello 2, perché può gestire sia indirizzi IP che indirizzi MAC.

Il Problema risolto da ARP: Per inviare un frame all'interno di una rete ethernet (classica) occorre sapere l'indirizzo MAC del destinatario, ma io so solo l'indirizzo IP.

Ogni nodo IP (host e routers, ossia dispositivi in grado di lavorare a livello 3) ha una tabella ARP.

La tabella ARP ha record del tipo → <Indirizzo IP , Indirizzo MAC , TTL>

Come per la switching table, i nodi creano e gestiscono la tabella ARP in maniera autonoma.

Come ogni tabella vista finora i record hanno un TTL, il quale in ARP è nell'ordine delle decine di minuti.

Supponiamo di avere due host A e B in una rete Ethernet.

A conosce l'indirizzo IP di B ma non l'indirizzo MAC.

1. A vuole mandare a B un datagram, ma il MAC di B non è nella sua Tabella ARP.
2. A manda un messaggio ARP in broadcast (quindi con l'indirizzo MAC di broadcast → FFFF..FF) contenente l'indirizzo IP di B.
 - a. "Di chi è questo indirizzo IP?"
3. B riceve il pacchetto ARP e vede che l'indirizzo IP contenuto è il suo.
4. B risponde ad A con il suo indirizzo MAC.
 - a. "Sono B, quello è il mio indirizzo IP, ecco il mio indirizzo MAC!"
5. A si segna il MAC di B nella sua Tabella ARP.

ARP in IPv6

In IPv6 non si usa il protocollo ARP, ma si usa una tecnica che ci permette di ricavare il MAC direttamente dall'indirizzo IPv6 (come una funzione Hash).

Il protocollo ARP risolve anche il routing tra LAN diverse.

Il router che connette le due LAN possiede 2 tabelle ARP , una per ogni LAN dove (per ogni host delle LAN) ho le associazioni IP - MAC.

Protocollo di supporto a IP - ICMP - Internet Control Message Protocol

Usato per inviare notifiche verso altri nodi della rete.

Queste notifiche possono riguardare molte cose:

- Malfunzionamenti.
- Informazioni di Controllo.
- echo reply
- echo request.
- Host irraggiungibile.
- Porta irraggiungibile.
- etc.

E' di livello network, ma funziona sopra il protocollo IP, perché può riferirsi anche ad errori di livello trasporto (porta irraggiungibile) o applicazione.

Struttura di un Messaggio ICMP

Sia P il datagram IP che ha causato l'errore.

Un messaggio ICMP è composto da:

- Tipo.
- Codice.
- Intestazione di P.
- I primi 8 bytes di P.

In base al tipo e codice ho una notifica diversa , ne mostriamo solo alcuni:

- Tipo = 0 | Codice = 0 → echo reply (ping)
- Tipo = 3 | Codice = 0 → Rete destinataria irraggiungibile.
 - Si butta via il datagram e si invia un messaggio ICMP di questo genere.
- Tipo = 3 | Codice = 2 → Protocollo destinatario irraggiungibile.
 - Si butta via il datagram e si invia un messaggio ICMP di questo genere.
- Tipo = 11 | Codice = 0 → TTL scaduto.

Forwarding Classico - “Destination Based”

Il processo di match viene fatto solo in base all'indirizzo di destinazione contenuto nell'intestazione del protocollo IP.

Simile al funzionamento del forward dello switch.

Questo è il metodo minimo e indispensabile per le funzionalità di un router.

Forwarding Generalizzato - “Match Plus Action”

Il processo di match viene fatto in base ai campi di intestazione del protocollo di livello 2,3 e 4

Quando arriva un datagram il router controlla la tabella di forwarding fino a trovare una entrata che corrisponde all'indirizzo di destinazione.

Prende il “*next hop*” (interfaccia di uscita) dalla riga della tabella e inoltra il datagram verso quella interfaccia.

La “*action*” esprime cosa bisogna fare con il datagram.

Questa tecnica si chiama “*Matching Plus Action*” (prima controllo e poi agisco).

L'azione da svolgere si basa su varie condizioni che si applicano a svariati campi non solo all'indirizzo di destinazione.

Poi le azioni possono essere varie :

- *Drop*
 - Scarto il pacchetto.
- *Copy*
- *Forward*
 - Inoltra il pacchetto verso una certa interfaccia di uscita.
- *Modify*
 - Modificare campi del pacchetto.

Ora la tabella di forwarding è chiamata “Flow Table”.

Ogni riga è del tipo < *Match* , *Action* >.

Viene specificata l'azione da fare (e non solo la porta che implica un forwarding).

La parte matching contiene vari campi.

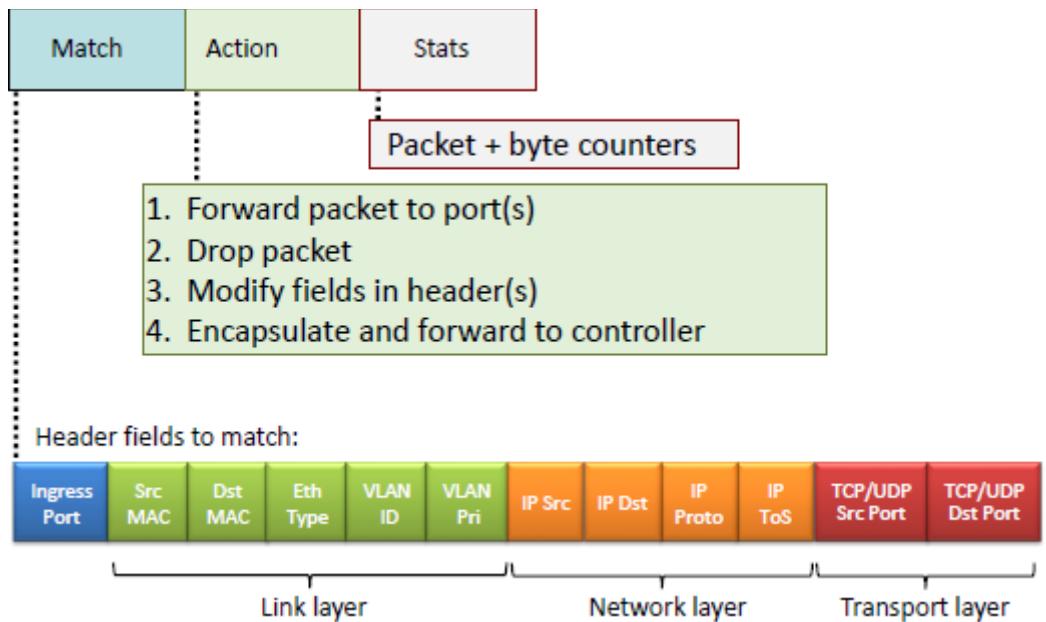
Ogni router ora ha una tabella dei flussi che permette di fare molte più cose.

src = *.*.*.* , dest=3.4.*.*	forward(2)
src=1.2.*.* , dest=*.**.*	drop
src=10.1.2.3 , dest=*.**.*	send to controller

Openflow

Openflow è un Framework con lo scopo di consentire la definizione di tabelle (dette tabelle dei flussi) che permettono di effettuare un forwarding generalizzato.

Spesso openflow è utilizzato in combinazione a SDN.



Esempi di OpenFlow

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
-------------	---------	---------	----------	---------	----------	--------	--------	---------	--------	------------	------------	--------

* * 22:A7:23:
11:E1:02 * * * * * * * * * * * * port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

Openflow è talmente flessibile che posso unificare (nel router) 4 dispositivi diversi in uno solo.

Router

- **match:** longest destination IP prefix
- **action:** forward out a link

Switch

- **match:** destination MAC address
- **action:** forward or flood

Firewall

- **match:** IP addresses and TCP/UDP port numbers
- **action:** permit or deny

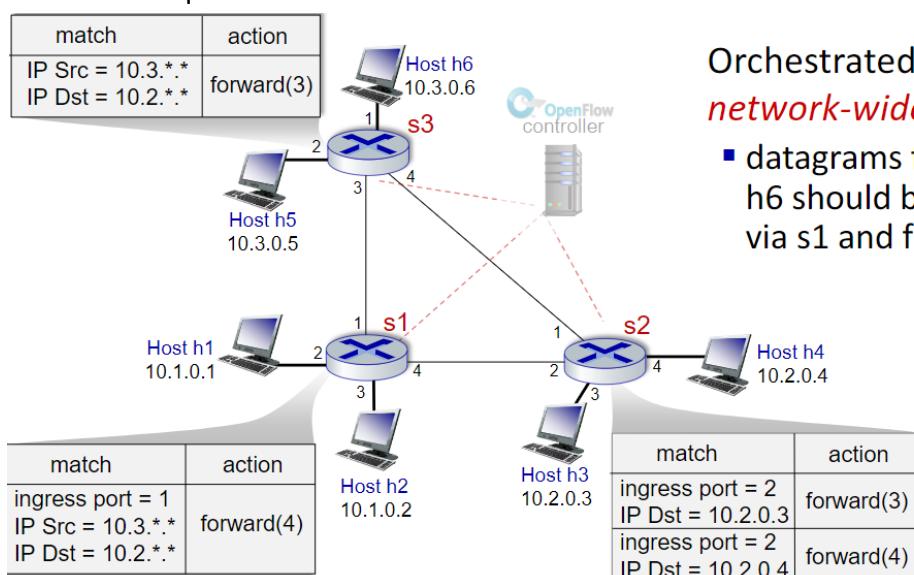
NAT

- **match:** IP address and port
- **action:** rewrite address and port

Comportamenti Network-Wide con Openflow

Tramite openflow posso creare dei comportamenti "Network-Wide", ossia comportamenti configurati in una specifica rete, ad esempio "*tutti i pacchetti provenienti da un certo host devono passare da questo specifico router e poi essere droppati*".

Basta solo programmare le tabelle di ogni router della rete in modo che il comportamento risultante sia quello voluto.



Orchestrated tables can create **network-wide** behavior, e.g.:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

Middleboxes

Qualsiasi destinatario intermedio (quindi tra un sorgente e un destinatario) che svolge qualsiasi funzione che non sia tipica del router.

Esempi:

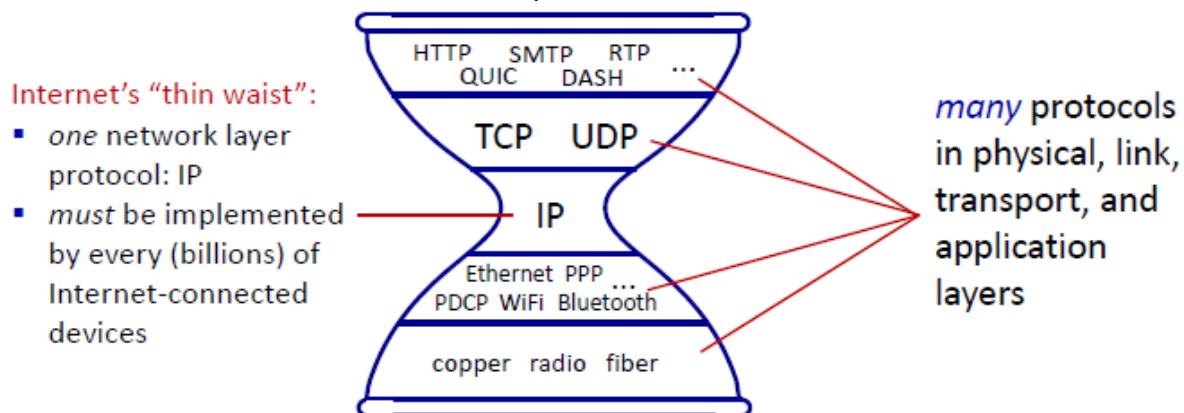
- Firewall
- IDS - Intruder Detection System
- NAT
- Cache

Inizialmente erano soluzioni hardware proprietarie.

Ora si usano soluzioni più programmabili (e quindi anche software) implementati tramite API.

IP Hourglass

Dall'invenzione di internet sono nati tanti protocolli.



Ci sono molti protocolli di livello rete (ossia livello 2) e moltissimi protocolli di livello applicazione.

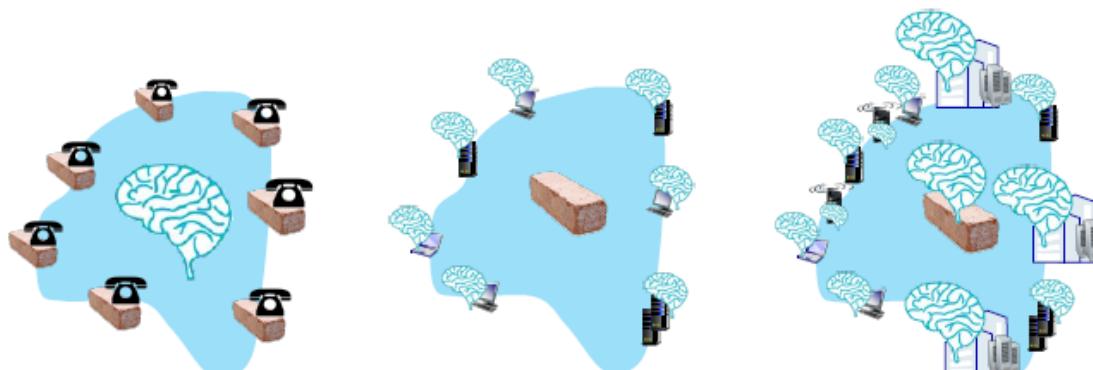
Ma a livello 3 c'è stata una stagnazione, il protocollo IP è sempre rimasto da solo.

Ma in realtà insieme al protocollo IP possiamo annoverare:

- NAT
- Caching
- Firewall
- NFV

Aggiunte che non modificano l'essenza di IP ma vanno a supporto del protocollo IP.

Distribuzione dell'Intelligenza (complessità)



20th century phone net:

- intelligence/computing at network switches

Internet (pre-2005)

- intelligence, computing at edge

Internet (post-2005)

- programmable network devices
- intelligence, computing, massive application-level infrastructure at edge

Nella rete telefonica i telefoni sono stupidi, l'intelligenza è concentrata nella centralina. Dopo, con internet, meno roba fanno i router/switch meglio è, perché gli permette di essere più veloci.

Livello Trasporto

Ha il compito di fornire una comunicazione logica tra 2 processi che risiedono in due macchine diverse.

Le caratteristiche di un protocollo di livello trasporto possono essere le seguenti:

- Multiplexing e Demultiplexing.
 - Effettuato da TCP e da UDP.
- Trasferimento di Dati Affidabile.
 - Effettuato da TCP
- Controllo del Flusso.
 - Effettuato da TCP.
- Controllo della Congestione.
 - Effettuato da TCP.

Logical Communication

Il livello 4 offre una comunicazione tra 2 processi che risiedono in 2 macchine diverse ma comunicanti

In questo livello i pacchetti sono chiamati *segmenti*.

Il livello applicazione divide i messaggi del livello applicazione in segmenti e li consegna a livello trasporto.

A livello di trasporto del ricevitore:

1. Verranno ricevuti i segmenti dal livello IP.
2. Si controllano i valori di check nell'header.
3. Si estrae il messaggio del livello applicazione scartando l'intestazione del livello trasporto.
4. Dopo aver ricevuto tutti i segmenti si riassemblano per ricomporre il messaggio originale.
5. Si esegue il *Demultiplexing* :
 - a. Si invia il messaggio all'applicazione destinataria attraverso il relativo socket associato alla porta destinataria.

Socket

Strumento fornito ai processi dal sistema operativo per utilizzare la rete.

Ogni socket è identificato dal numero di porta.

Multiplexing

E' il processo di prendere dati da un socket sorgente e aggiungere le intestazioni di livello trasporto.

Quindi il multiplexing è un'operazione che ha come attore principale l'insieme dei processi che sono in esecuzione sulla macchina sender (e che vogliono inviare o ricevere messaggi nella rete) e non la macchina sender stessa.

Demultiplexing

E' il processo di prendere un certo segmento, togliere l'intestazione di livello trasporto e inviarlo nel socket corrispondente al numero di porta destinatario.

Il demultiplexing è un'operazione che consiste nell'inoltrare un messaggio del livello applicazione all'opportuno socket di destinazione caratterizzato dal numero di porta.

L'operazione di demultiplexing cambia a seconda del protocollo di livello trasporto usato:

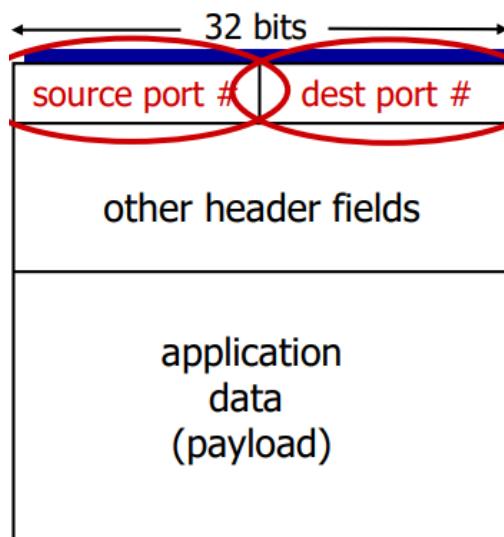
- UDP → viene fatto solo controllando il Numero di Porta Destinatario.
- TCP → viene fatto controllando tutti e 4 i parametri citati sopra.

Formato del Pacchetto di livello Trasporto

Il livello trasporto dell'host ricevitore riceverà dei datagram IP.

Ogni segmento ha 4 valori molto importanti che identificano una connessione TCP:

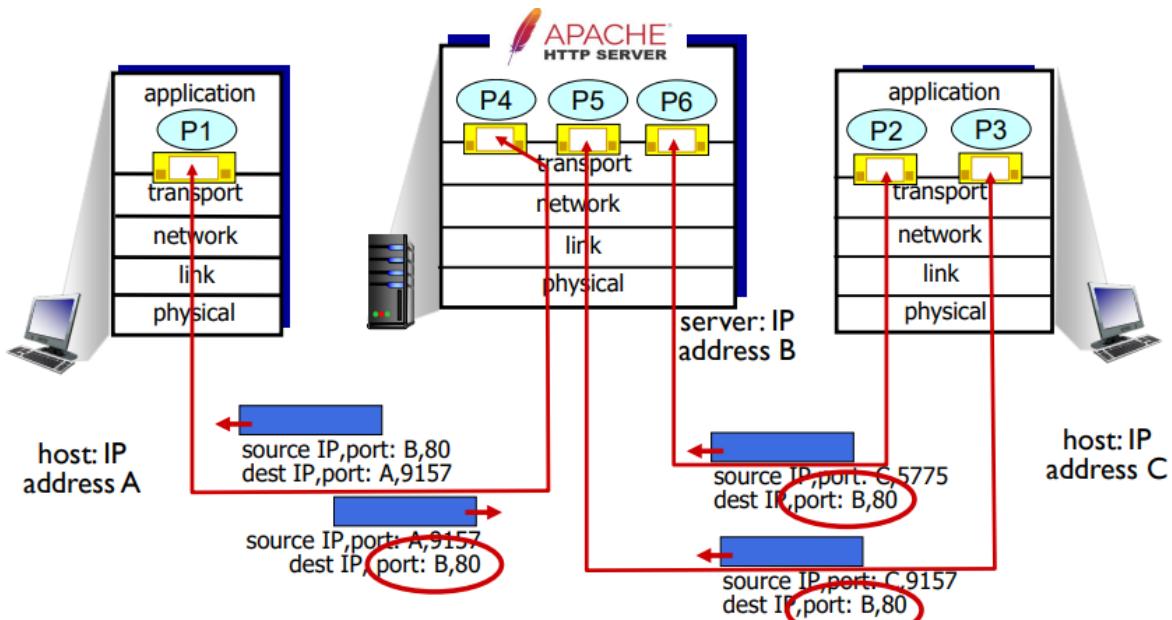
- Intestazione di Livello Network.
 - **IP Sorgente** e **IP Destinatario**.
- Intestazione del Livello Trasporto.
 - **Numero di Porta Sorgente** e **Numero di Porta Destinatario**.



Demultiplexing Corretto

Il receiver userà i 2 indirizzi IP e i 2 Numeri di porta per eseguire il demultiplexing.

Una macchina ricevitore può supportare molteplici socket TCP/UDP contemporaneamente. Quindi deve per forza considerare le 4 informazioni insieme per garantire un demultiplexing corretto, poiché in questo caso ogni socket TCP/UDP della macchina receiver è identificato dalla quadrupla sopracitata.



UDP - User Datagram Protocol

Servizio di tipo “*Best Effort*”, quindi nessuna garanzia, si spedisce il segmento e basta.
Manca ogni tipo di sicurezza ma la differenza di prestazioni in confronto al TCP è grande.

UDP si occupa del minimo indispensabile: multiplexing, demultiplexing e una “leggera” error detection

UDP non prevede la creazione di una connessione (*connectionless*).

La mancanza di una connessione diminuisce l’overhead spaziale e temporale poiché la creazione e la gestione di una connessione occupa memoria (a causa delle strutture dati necessarie) e anche tempo prezioso.

Dove è usato UDP?

UDP è usato in vari protocollti:

- **DNS**
 - DNS prevede messaggi molto piccoli, quindi eventuali ritrasmissioni non saranno un problema.
 - Grazie alla leggerezza di UDP un server DNS può soddisfare molte più richieste per unità di tempo.
- **SNMP (Simple Network Management Protocol)**
 - Non fatto a Lezione.
 - Protocollo per ricevere e organizzare informazioni dai dispositivi IP gestiti.
- **HTTP / 3**
 - Si usa UDP perché negli ultimi anni si è preferito delegare il controllo sull'affidabilità e il controllo della congestione al livello di applicazione, in modo da poter dare più flessibilità agli sviluppatori i quali conoscono bene di cosa ha bisogno l'applicazione.
- **Applicazioni di streaming di contenuti multimediali.**
 - In questi casi UDP è utile perchè servizi di questo tipo sono:
 - “*loss tolerant*”

- Un servizio è tale quando a fronte di perdite di alcuni pacchetti il servizio può continuare ad essere erogato.
- “**rate sensitive**”
- Un servizio è tale quando la qualità del servizio dipende fortemente dal rateo di arrivo dei pacchetti.
 - Quindi è imperativo che il throughput sia il più alto possibile.

Formato Pacchetto UDP

UDP prevede un header più piccolo rispetto a quello di TCP (poiché deve fare molti meno controlli) di dimensione fissa pari a 64 bit.

- **Porta Sorgente & Porta Destinataria – 4 Byte**
- **Length – 2 Byte**
 - Lunghezza del segmento UDP (intestazione + payload) espressa in bytes.
 - Il payload ha dimensione massima di 8192 bytes (65'536 bit).
 - Il payload ha dimensione (length - 8) bytes.
- **Checksum – 2 Byte**
- **Payload**
 - Il Pacchetto ottenuto dal livello superiore (livello applicazione).

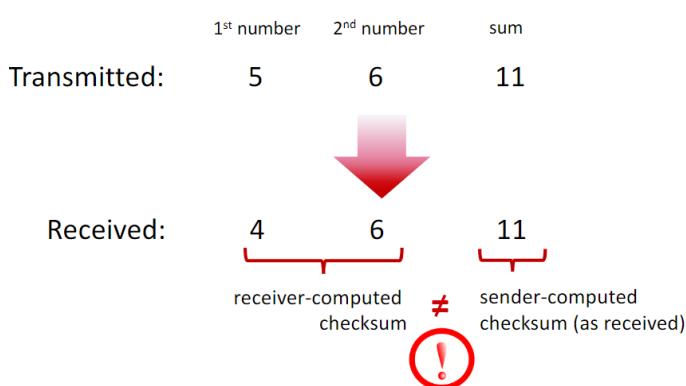
Checksum - Internet Checksum

Verifica l'esistenza di errori (detection) ossia la presenza di bit “*flippati*”.

Consideriamo tutto il Segmento come una sequenza di interi a 16 bit.

Il Sender:

1. Esegue una *checksum addition* della sequenza di interi a 16 bit.
 - *checksum addition* → Somma in modulo su 16 bit
2. Mette il risultato negato nel campo checksum del segmento.
 - Neanche il professore sa perché si usa il risultato negato invece che il risultato vero e proprio...non facciamoci domande e accettiamo il fatto che è stato fatto così.



Il Receiver:

1. Riceve il segmento dal Sender.
2. Esegue la *checksum addition* e confronta il risultato (negato) con il campo checksum del segmento appena ricevuto.
 - Se sono Uguali:
 - Non sono sicuro al 100% che sia integro.

- Infatti basta dati due numeri a 16 bit , che 2 coppie di bit posizionate nello stessa posizione in entrambi i numeri fippino che il risultato non cambia e quindi l'error detection non interviene.

$ \begin{array}{r} 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1 \\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \end{array} $ $ \begin{array}{r} 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0 \end{array} $ $ \begin{array}{r} 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1 \end{array} $	<p>Even though numbers have changed (bit flips), no change in checksum!</p>
---	--

- **Se sono Diversi:**

- Sono sicuro al 100% che NON è integro.

$ \begin{array}{r} 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1 \end{array} $ wraparound $ \begin{array}{r} 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1 \end{array} $ sum $ \begin{array}{r} 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0 \end{array} $ checksum $ \begin{array}{r} 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1 \end{array} $	
---	--

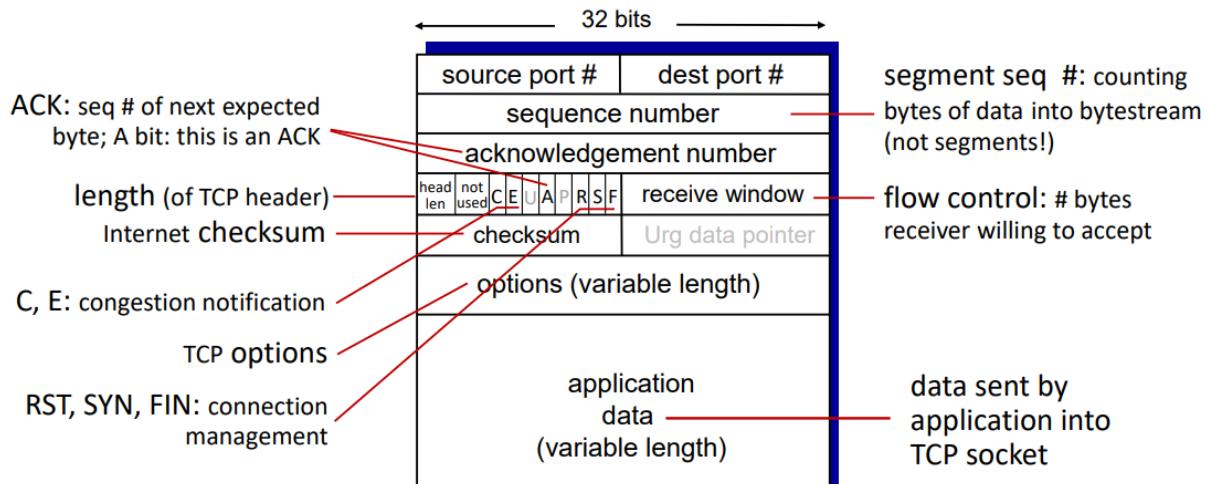
Protocollo TCP

TCP si pone come obiettivo di ricreare (virtualmente) una comunicazione di tipo **point-to-point**

punto-punto → “Come se i due host fossero direttamente collegati senza intermediari di mezzo”.

- **TCP è Affidabile.**
 - I pacchetti arriveranno tutti, se si perde qualche pacchetto o se è corrotto si trasmette di nuovo.
- **TCP mantiene l'ordine dei pacchetti.**
- **TCP ha un flusso di dati “full-duplex”**
 - In una connessione TCP i segmenti possono essere inviati contemporaneamente da entrambe le direzioni.
 - I segmenti hanno una MSS (*Maximum Segment Size*).
- **TCP usa il Pipelining con ACK cumulativi.**
 - Più pacchetti inviati in modo consecutivo senza aspettare i relativi ACK in una connessione.
- **TCP è connection-oriented.**
 - Sender e receiver si scambiano informazioni di controllo prima di iniziare la comunicazione (eseguono un handshake).
- **TCP controlla il flusso**
 - Il sender non manderà più pacchetti di quanti il receiver riesca a gestire in modo da evitare di sovraccaricare il receiver e quindi perdere pacchetti.

Struttura del Segmento TCP



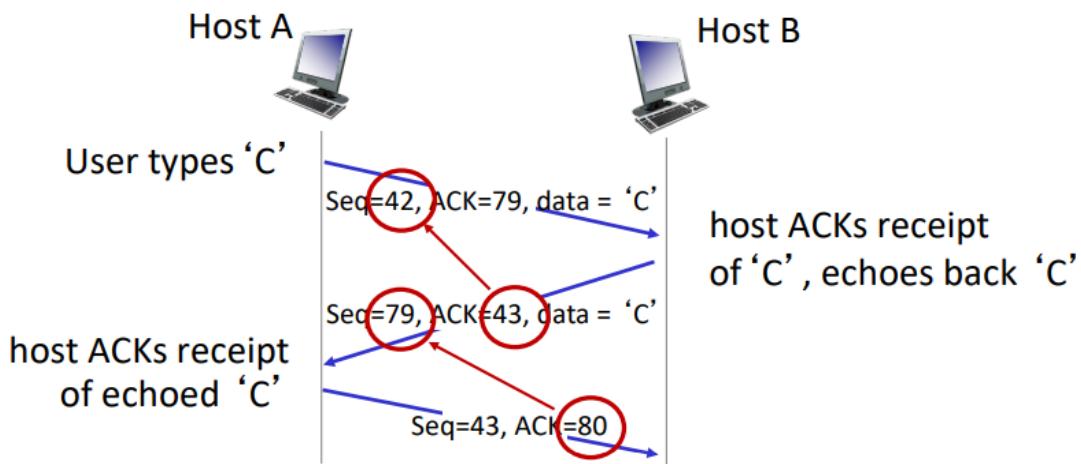
Numeri di Sequenza in TCP

Il numero di sequenza (*sequence number*) indica il numero di sequenza relativo al segmento e serve al receiver per scartare i doppioni e per capire quali segmenti mancano in modo da mandare ACK cumulativi in modo opportuno.

Il numero di sequenza indica lo scostamento (espresso in numero di byte) del pacchetto corrente rispetto all'inizio del segmento TCP all'interno del flusso.

Se un segmento contiene i byte { 92 , 93 , 94 , 95 } del flusso, allora il numero di sequenza di questo segmento sarà pari a 92.

Il relativo ACK sarà “ACK 96”, perché ora il ricevitore si aspetta il 96° byte.

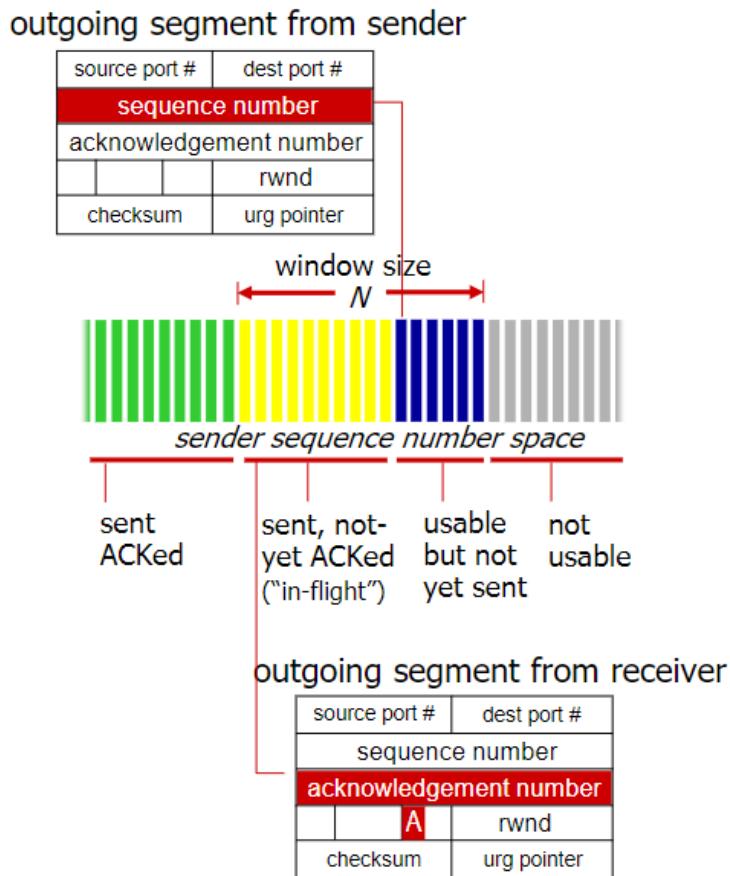


ACK Cumulativi

In base ai numeri di sequenza ricevuti il receiver comporrà gli ACK cumulativi dove specificherà al sender l'ultimo pacchetto *in-order* (ossia il numero di sequenza n tale che il receiver ha ricevuto tutti i pacchetti con numero di sequenza strettamente minore di n) che ha ricevuto.

Se il ricevitore spedisce ACK 100, significa che nel prossimo messaggio che gli verrà spedito si aspetta un segmento che contiene il 100° byte del flusso.

La gestione dei pacchetti fuori ordine non è specificata da TCP, dipende dal ricevitore.



Protocollo Telnet

Protocollo basato su TCP (porta 23) nasce inizialmente per consentire all'utente di eseguire il login in una macchina da un'altra macchina.

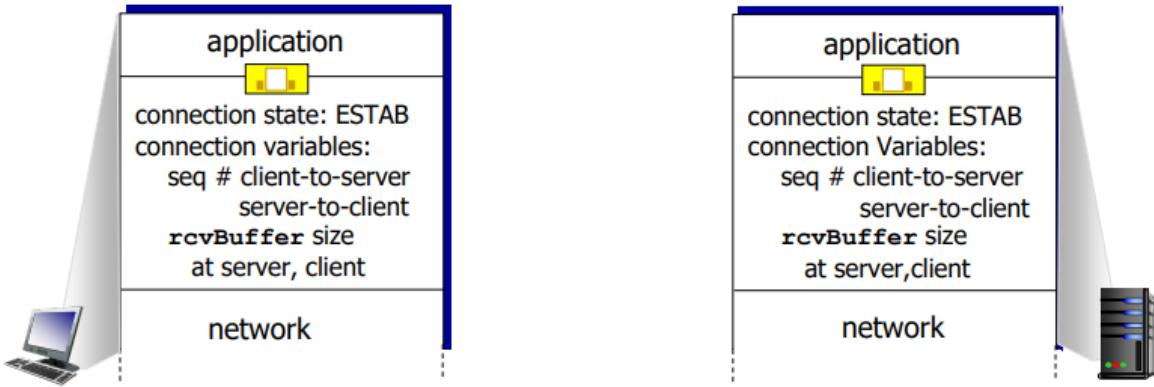
In parte a causa della progettazione del protocollo e in parte per la grande flessibilità tipicamente fornita dai programmi Telnet, è possibile utilizzare un programma Telnet per stabilire una connessione interattiva ad un qualche altro servizio su un server internet.

Un utilizzo classico è collegarsi col Telnet alla porta 25 (sulla quale tipicamente si trova un server SMTP) per effettuare il debugging di un server di posta.

Strutture dati della Connessione TCP

Il livello TCP del lato A e il livello TCP del lato B allocano strutture dati utili a tenere informazioni sulla connessione e sui segmenti inviati (ad esempio n°sequenza , acknowledgment number etc..).

Le strutture vengono deallocate alla chiusura della connessione.



Apertura della Connessione TCP

Quando il client esegue la `connect()`, a livello TCP parte la procedura di richiesta di apertura di una connessione.

La richiesta una volta arrivata al server viene accodata nella coda delle richieste.

Il server esegue `accept()` per accettare richieste di connessione da parte dei client.

Three Way Handshake - Inizio della Connessione TCP

Stretta di mano a "3 vie" perché vengono inviati 3 segmenti di ACK.

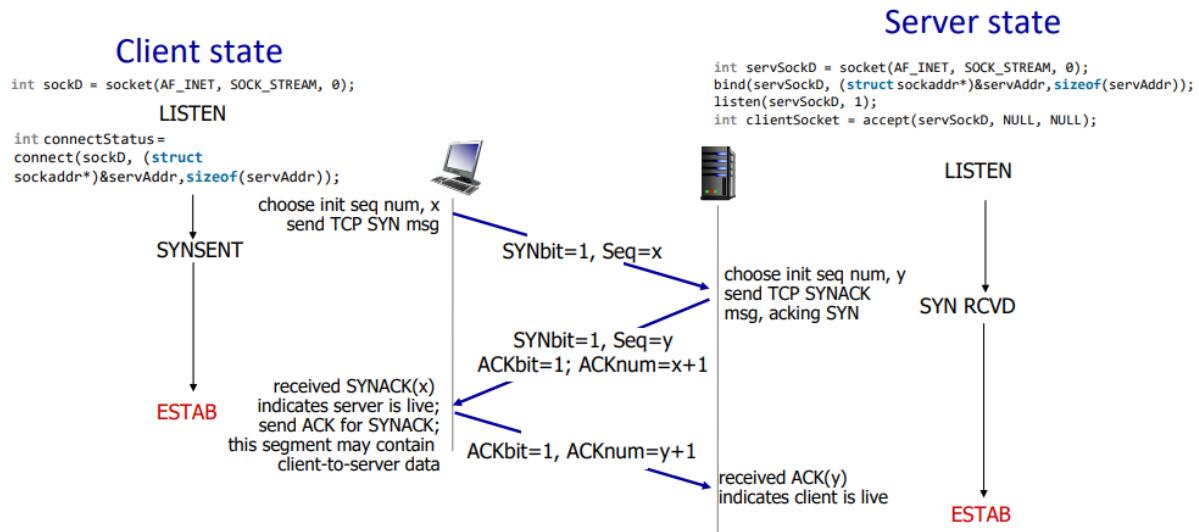
I messaggi del Three-Way handshake hanno il bit `SYNbit = 1`, il quale indica che quel segmento riguarda l'instaurazione di una connessione TCP.

1. Il Server crea il socket con `socket()`, fa la `bind()` e pone il socket in ascolto con la `listen()`.
 - Il Server pone lo stato della connessione a `LISTEN`.
2. Il Client crea il socket con `socket()` e fa la `bind()`, oppure il sistema operativo del client assegna una porta effimera al socket.
 - Il Client pone lo stato della connessione a `LISTEN`
3. Il Client esegue la `connect()`, quindi invia una richiesta di apertura di connessione (essa sarà un segmento TCP con numero di sequenza x) al Server.
 - Come numero di sequenza mette `Seq = x`, con x un numero casuale.
 - Il Client pone lo stato della Connessione a "SYN SNT".
4. Il Server riceve la richiesta di connessione del client, la mette nella coda delle richieste e successivamente accetta la richiesta con `accept()`.
 - Il Server pone lo stato della Connessione a "SYN RCVD".
5. Il Server alloca tutte le strutture dati necessarie per gestire la connessione TCP.
6. Il Server invia la risposta, ossia un segmento TCP con:
 - `SYNbit = 1 ; Seq = y ; ACKbit = 1; ACKnum = (x + 1);`
 - `ACKnum = (x + 1)` perché è l'ACK relativo al segmento TCP del punto 3.
7. Il Client riceve la risposta e capisce che la connessione è stata stabilita.
8. Il Client alloca tutte le strutture dati necessarie per gestire la connessione TCP.
 - La `connect()` ritorna con successo.
 - Il Client pone lo stato della Connessione a "`ESTAB`".
9. Il Client invia la risposta, un segmento TCP con:
 - `ACKbit = 1 ; ACKnum = (y + 1);`

- Questa risposta serve ad accettare il server che il client è ancora presente e che il primo messaggio non era una richiesta di connessione vecchia.

10. Il Server riceve.

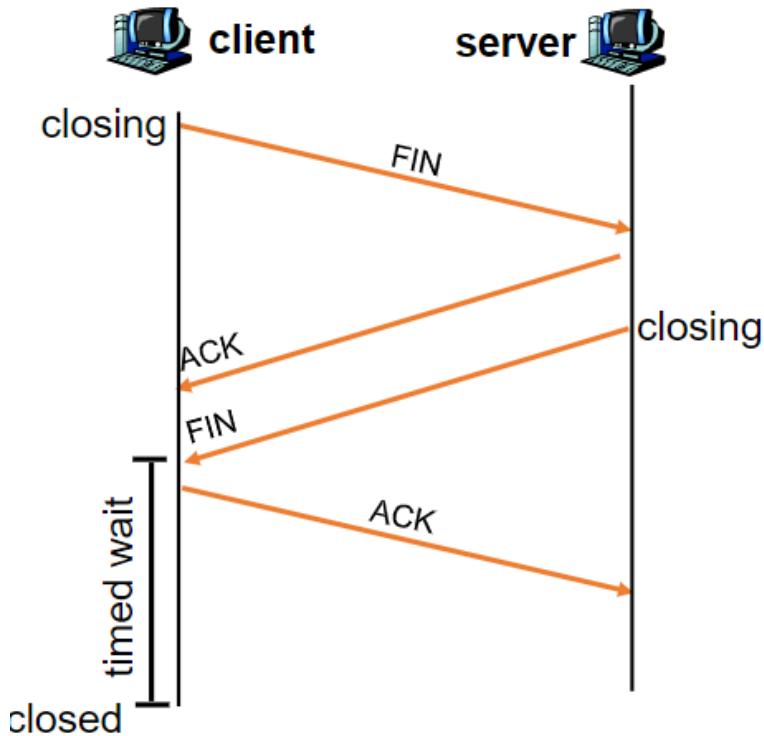
- Comprende che il client ha capito che la sua richiesta di connessione è stata accettata.
- Il Server pone lo stato della Connessione a “ESTAB”.



Fine Vita della Connessione TCP - “Tear Down”

La procedura di chiusura viene eseguita secondo uno standard.

- Il Client invia un segmento di “TCP FIN” al server.
 - Ossia con il flag di chiusura alzato.
- Il Server riceve il segmento, vede il flag FIN alzato e invia un ACK con lo stesso flag alzato.
- Il Client riceve l'ACK del server, vede il flag FIN alzato e invia un ACK.
- Il Client rimane in attesa per un periodo di tempo determinato.
 - Il Client non chiude subito ma rimane aperto per un certo periodo di tempo “Timed Wait”, perché non è sicuro che il server riceva il suddetto ACK e che di conseguenza il server potrebbe rimanere con la connessione aperta.
 - Se il server non riceve ACK: allora il server rimanda il segmento del punto 2 e il client dato che è rimasto in attesa ha modo di ricevere il segmento e quindi di rinviare il segmento di ACK del punto 3.
- Il Server riceve l'ACK ed elimina tutte le strutture dati della connessione.
- Allo scadere di “Timed Wait” il Client elimina le strutture dati della connessione.



Riassunto - Apertura Connessione TCP

1. Il Client sceglie randomicamente x e invia al Server un segmento.
 - SYNbit = 1; Seq = x ;
 - *“Vorrei una connessione TCP con te”*
2. Il Server riceve il segmento, sceglie randomicamente y e invia una risposta
 - SYNbit = 1; Seq = y ; ACKnum = $x+1$;
 - Il Server alloca le strutture dati necessarie.
 - *“Va bene, mandami la conferma”*
3. Il Client riceve la risposta del server e invia la risposta
 - SYNbit = 1; Seq = $x + 1$; ACKnum = $y + 1$;
 - Il Client alloca le strutture dati necessarie.
 - *“Ok, ecco la conferma”*

Riassunto - Chiusura Connessione TCP

1. Il Client invia al Server un segmento.
 - FINbit = 1; Seq = x ;
 - *“Vorrei chiudere la connessione TCP con te”*
2. Il Server riceve, sceglie randomicamente y e invia una risposta
 - FINbit = 1; Seq = y ; ACKnum = $x+1$;
 - *“Va bene, mandami la conferma”*
3. Il Client riceve e invia la risposta
 - FINbit = 1; Seq = $x + 1$; ACKnum = $y + 1$;
 - Il Server appena ottenuta la risposta dealloca le strutture dati.
 - *“Ok, ecco la conferma”*
4. Il Client attende un pò di tempo.

- Questa attesa serve nel caso in cui il server non abbia ricevuto la risposta del punto 3, in tal caso il Server invierà di nuovo il segmento del punto 2, dato che il client è in attesa sarà in grado di riceverlo e nel caso inviare di nuovo il segmento del punto 3.
- Il Client, allo scadere del timer dealloca le strutture dati.

Trasferimento dei Dati Affidabile nel TCP

TCP lato client e TCP lato server sono collegati come se fossero in una connessione **punto-punto**.

Con la differenza che ora il collegamento è virtuale.

Lo schema è “*window-based ARQ*”.

- **Acknowledgements.**
- **Timeouts.**
- **Retransmission.**

ARQ - Automatic Repeat ReQuest

Strategia di controllo di errore, che svolge il compito di rivelare un errore (ma non di correggerlo).

I pacchetti corrotti vengono scartati e viene richiesta la loro ritrasmissione.

La questione del Timeout

Se troppo corto il timer scatta troppo presto e spesso.

Ciò provocherebbe ritrasmissioni inutili, quindi più doppioni e quindi più traffico in rete.

Se troppo lungo il *throughput* cala a picco.

Occorre dare un valore opportuno al timeout e che magari si adattasse dinamicamente alla connessione in cui viene applicato.

Round Trip Time

Il Round Trip Time (RTT) è il tempo misurato partendo dal primo segmento di trasmissione fino all'

arrivo del relativo ACK.

Ignorando eventuali ritrasmissioni, nella misurazione si parte sempre dal primissimo segmento.

Infatti spesso quando si ha un RTT molto alto è perché la rete è congestionata o disturbata e che quindi ha richiesto delle ritrasmissioni.

Stima del Round Trip Time

Cerchiamo di “indovinare” il Round Trip Time del segmento che stiamo per spedire basandosi sui RTT dei segmenti inviati precedentemente nella stessa connessione TCP.

Nella stima del Round Trip Time successivo io considero solo una parte degli RTT passati, per la precisione considero gli ultimi ‘n’ RTT, dove n è deciso a priori.

Dove n deve avere un valore equilibrato, né troppo grande e né troppo piccolo.

Algoritmo della Media Esponenziale Mobile

La stima fatta con questo metodo non oscilla e con opportuni parametri posso impostare la reattività e la stabilità del valore di uscita.

Per applicare questo metodo occorre misurare i RTT dei segmenti precedenti:

- Le misure le chiamo *SampleRTT* → RTT
- Le stime le chiamo *EstimatedRTT* → ERTT

$$ERTT_1 = RTT_0$$

$$ERTT_2 = a * RTT_1 + (1 - a) * RTT_0$$

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

$$ERTT_{n+2} = a * RTT_{n+1} + a * (1 - a) * RTT_n + a * (1 - a)^2 * RTT_{n-1} + \dots + (1 - a)^n * RTT_1$$

Regolazione della Reattività e della Stabilità

Con $a < 1$, di solito $a = 0.125$.

Questo è il parametro di cui si parlava prima, esprime il peso che devono avere le misure recenti rispetto a quelle vecchie.

Più l'uscita è “reattiva” e più è sensibile ai valori recenti rispetto a quelli passati.

Più l'uscita è “stabile” o “stazionaria” e meno è sensibile ai valori recenti rispetto a quelli passati.

A seconda del valore di a posso modificare la reattività o la stabilità.

- Se $a \sim 1 \rightarrow$ La Media è più reattiva perché la stima si avvicina all'ultimo valore di RTT.
- Se $a \sim 0 \rightarrow$ La Media è più stabile perché si avvicina e resta vicina molto ai primi valori di RTT.

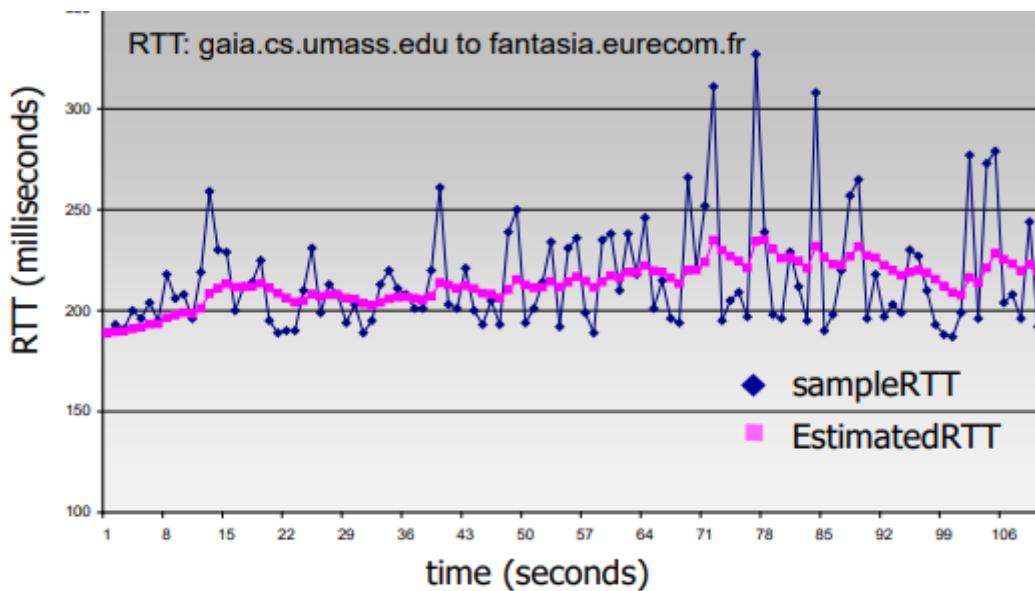
Così come è fatto l'algoritmo va bene a livello matematico, ma nel concreto non posso tenere in memoria milioni di SampleRTT e soprattutto non posso fare altrettante somme e moltiplicazioni per ogni pacchetto.

Ciò è tranquillamente evitabile, perché se svolgo qualche semplice passaggio matematico:

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

$$ERTT_{n+1} = a * RTT_n + (1 - a) * ERTT_n$$

La stima al passo $n+1$ dipende dalla stima del passo n -esimo e il SampleRTT n -esimo.



Calcolo del Timeout e Margine di Sicurezza

L'unico vincolo di cui dobbiamo preoccuparci è → $\text{TimeOut}_n \geq \text{ERTT}_n$

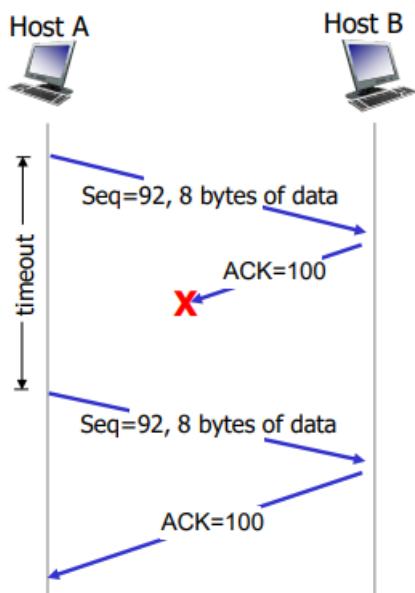
Un modo comune di calcolarlo → $\text{TimeOut}_n = \text{ERTT}_n + 4 * \text{DevRTT}_n$

DevRTT è detto “*Margine di Sicurezza*”, è direttamente proporzionale alla variabilità dei SampleRTT.

$$\text{DevRTT}_{n+1} = (1 - b) * \text{DevRTT}_n + b * |\text{RTT}_n - \text{ERTT}_n|$$

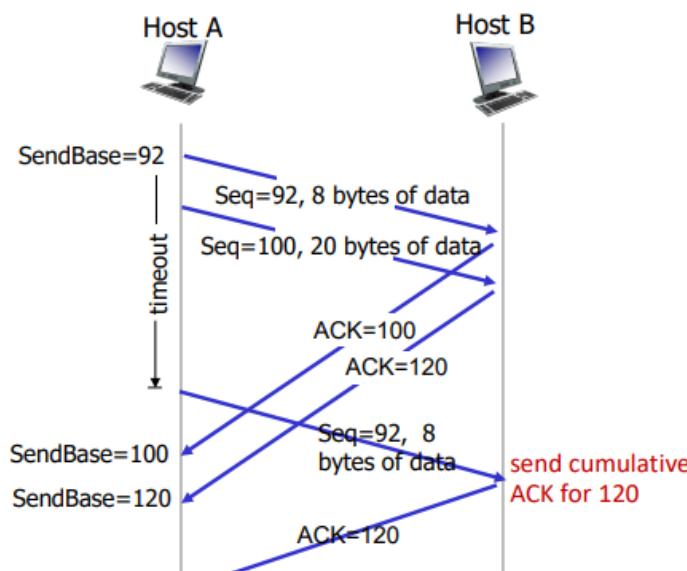
Di solito $b = 0.25$.

Possibili Casi di Ritrasmissione nel TCP



lost ACK scenario

Caso semplice, il primo pacchetto di ACK viene perso, il mittente allo scadere del timeout rimanda il pacchetto e adesso il pacchetto di ACK viene effettivamente recapitato al mittente.

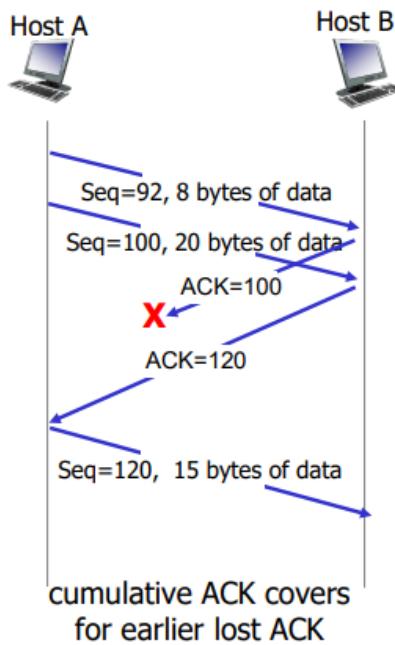


premature timeout

I pacchetti di ACK arrivano tardi, il mittente allo scadere del timeout rimanda il primo pacchetto (seq# 92).

Riceve gli ACK arrivati in ritardo e registra che sono stati ricevuti, quindi non ritrasmette nulla.

Dopo un pò arriva lo ACK per il pacchetto rimandato, il mittente riconosce che è un doppione e lo ignora.



Il primo ACK viene perso, il secondo (con sequenza maggiore) invece arriva.

Il mittente capisce che il ricevitore ha ricevuto 120 byte, quindi implicitamente il ricevitore ha ricevuto entrambi i pacchetti, quindi nonostante il mittente non abbia ricevuto il 1° ACK non ritrasmette nulla perché il 2° ACK prova il fatto che il ricevitore ha ricevuto ANCHE il 1° pacchetto (quello con seq# 92)..

TCP Fast Retransmit

Aspettare ogni volta lo scadere del timeout è oneroso e quando la rete ha un throughput abbastanza alto la cosa diventa uno spreco.

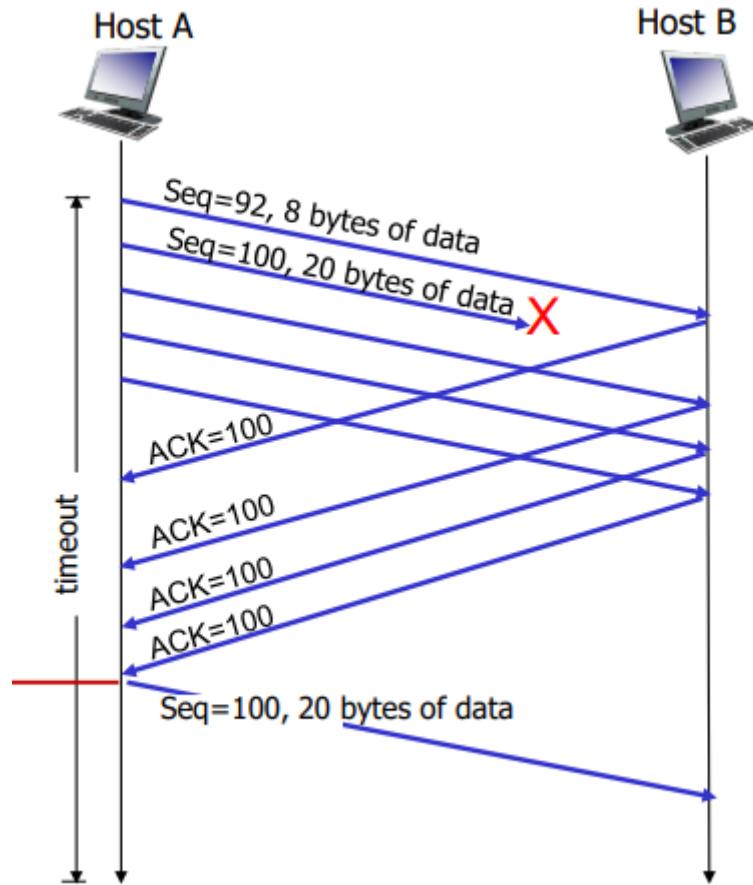
Vorrei che il pacchetto perso venga ritrasmesso PRIMA dello scadere del timeout.

Supponiamo che un pacchetto con un numero di sequenza x non arrivi mai.

Il ricevitore (se abbastanza veloce) invierà periodicamente ACK x in modo da avvertire il trasmittitore che il pacchetto con numero di sequenza x non è arrivato.

Se il trasmittitore riceve 3 doppioni di "ACK x", allora ritrasmetterà il pacchetto con numero di sequenza x senza aspettare lo scadere del timeout.

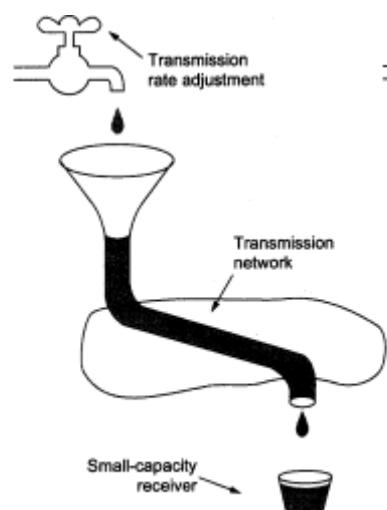
In questo modo se la rete ha un throughput elevato non mi serve aspettare il timeout e quindi rallentare inutilmente la comunicazione.



TCP Flow Control

Problema : Il livello network invia dati con un rateo di invio superiore al rateo con cui il livello applicazione legge dati dal buffer del socket.

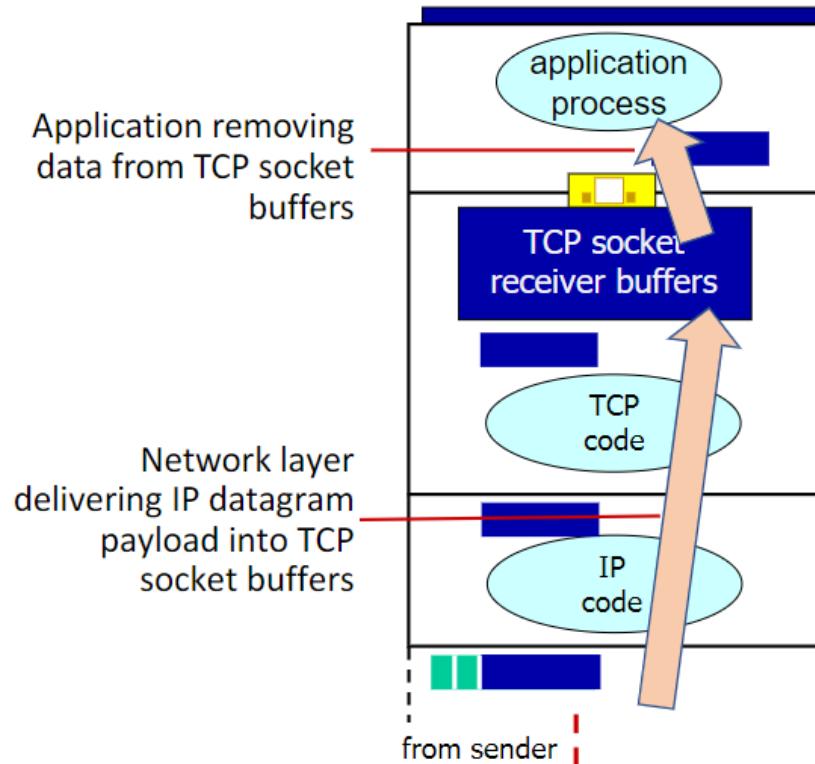
Il livello di applicazione del ricevitore non riesce a stare dietro alla velocità di arrivo dei pacchetti.



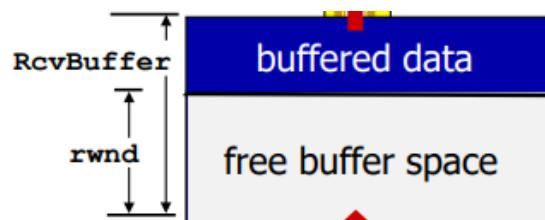
Il buffer di ricezione del socket è la zona di memoria del ricevitore dove vengono messi i dati dopo il demultiplexing, i dati rimangono lì in attesa che l'applicazione esegua la receive.

Se l'applicazione risulta troppo lenta il buffer di ricezione del socket rischia di riempirsi e alcuni messaggi potrebbero essere persi a causa dell'overflow.

Il sender deve avere modo di capire i limiti del ricevitore e quindi evitare di inviare troppa roba rischiando un overflow del buffer del socket del trasmettitore.



Il livello TCP del ricevitore dice al TCP sender quando spazio ha ancora libero nel buffer del socket.



La dimensione del buffer ossia RcvBuffer viene impostata al momento della creazione del socket, tipicamente è di 4096 bytes.

Sia rwnd la quantità di byte liberi nel buffer dei dati.

Il sender fa in modo che la dimensione dei pacchetti "unACKed" che stanno circolando nella rete (quindi pacchetti nello stato "*in-fight*") non superino mai rwnd, ossia la quantità di byte liberi nel buffer del ricevitore.

Per fare questa cosa usiamo un campo dell'intestazione del segmento TCP, chiamato "receive window", dove il receiver specifica il valore di rwnd nei segmenti di ACK cumulativi.

Usando questo metodo, non si avrà mai un overflow del buffer del socket ricevitore.

Problema - rwnd troppo basso

Può capitare che il buffer del ricevitore si riempia completamente (quindi rwnd talmente piccolo da non poter inviare nulla), in questo caso il trasmettitore rimarrebbe bloccato e non potrebbe trasmettere ulteriori dati.

Per risolvere il problema, il Sender continua ad inviare segmenti costituiti da un solo byte, composto da soli zeri.

Questi pacchetti vengono chiamati “*WindowProbe*”.

Questi segmenti verranno scartati se il buffer del ricevitore è pieno.

Se invece il buffer del ricevitore non è pieno, il trasmettitore riceverà degli ACK dal trasmettitore con specificato il valore di rwnd, se rwnd è abbastanza grande invierà altra roba.

TCP Congestion Control

L'effetto finale agli occhi degli utenti è lo stesso del flow control, però la congestione si verifica sul canale e non sul buffer del ricevitore.

La congestione è quando i buffer (di entrata o di uscita) dei router si riempiono troppo velocemente e i router non riescono a gestirli in tempo..

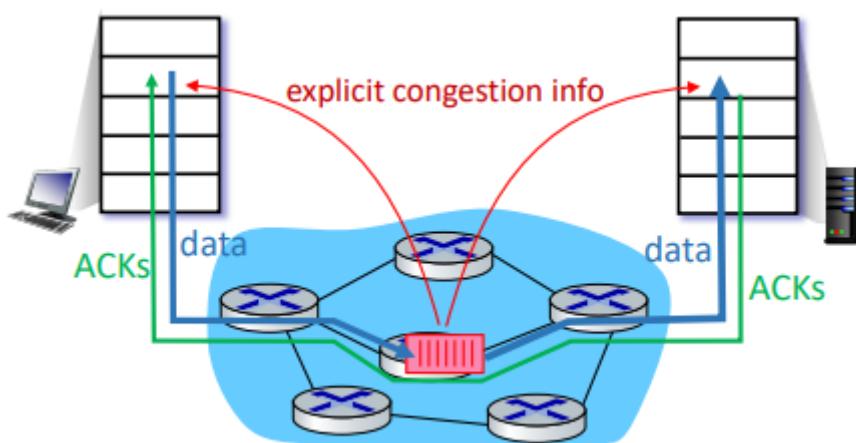
Nel caso migliore avrà un ritardo di accodamento, in caso di grave congestione i datagram vengono scartati.

Network-Assisted Congestion Control

La rilevazione della congestione viene fatta dai routers lungo la route.

La rilevazione della congestione è certa, perché sono proprio i router a farla.

I router forniscono direttamente dei feedback sia al ricevitore che al trasmettitore sul livello di congestione.



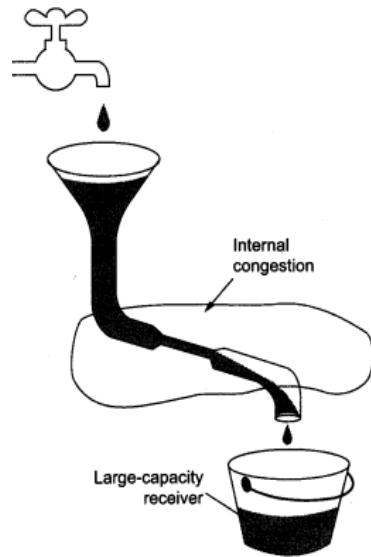
Il router non fa altro che inviare pacchetti dove viene esplicitato il livello di congestione che sta sopportando.

I trasmettitori si adeguano di conseguenza.

End-End Congestion Control

La rilevazione della congestione viene fatta dal ricevitore e dal trasmettitore.

La rilevazione della congestione è probabilistica, viene eseguita osservandone i sintomi.



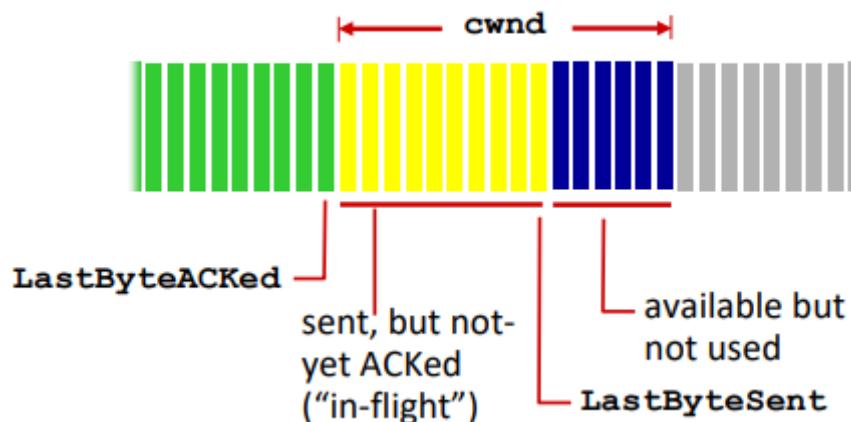
Consideriamo il canale “congestionato” quando:

- Il trasmettitore riceve un triplice ACK duplicato relativo ad un segmento precedentemente inviato.
- Se scade il timeout relativo ad un segmento precedentemente inviato.

Ossia → Il canale è congestionato quando perdo almeno un pacchetto.

Il mio obiettivo è trovare (aggiustando il rate di trasmissione) la velocità massima di trasmissione per cui non intaso la rete.

Per fare ciò il sender quindi si occupa di limitare il numero di bytes unACKed nella pipeline.
Sia cwnd il numero che esprime il massimo numero di bytes unACKed che posso avere.



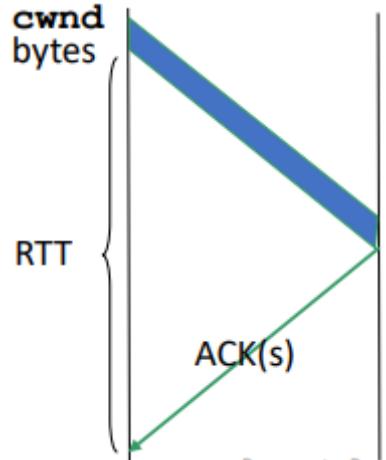
Supponendo di mettere cwnd bytes in pipeline e che per ricevere la risposta debba aspettare RTT.

Posso definire il rate di trasmissione del sender come:

$$\text{Rate}_{\text{sender}} = \frac{\text{cwnd}}{\text{RTT}}$$

Byte per Sec

Quindi il Sender può regolare il suo rate di trasmissione modificando cwnd.



Non scordiamoci del controllo del Flusso

Ricordiamo che nel lato ricevitore abbiamo **rwnd** del flow control, che indica quanti byte può ricevere.

Il sender deve fare in modo che il numero di byte “in-flight” sia minore di $\min\{ rwnd, cwnd \}$

$$\text{LastByteSent} - \text{LastByteACKed} \leq \min\{ rwnd, cwnd \}$$

MSS - Maximum Segment Size

MSS (*Maximum Segment Size*) rappresenta la massima dimensione del *payload* in un segmento TCP, da non confondersi con la massima dimensione di quest'ultimo, poiché questo è composto infatti da una parte di intestazione (header) e una appunto di corpo dati. Ogni connessione TCP ha il suo MSS, negoziato nella fase di setup di una connessione TCP.

In IPv4, la MSS è tipicamente 1460 Byte, perché la MTU tipica è 1500 byte, a cui vanno sottratti 20 Byte di Header IP e 20 Byte di Header TCP.

In genere la MSS minima è di 536 Byte.

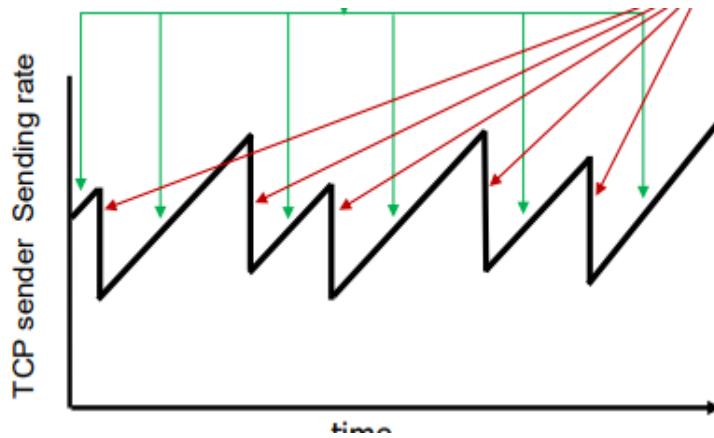
AIMD - Additive Increase Multiplicative Decrease

Il sender aumenta il rate di invio fino a che non riceve un sintomo di congestione.

Se rileva un sintomo di congestione diminuisce il rateo di invio.

Il mio obiettivo NON è evitare la congestione, ma limitarla.

Per essere sicuro di evitarla al 100% dovrei mettere il rate del sender a 0.



AIMD - Fase di Slow Start

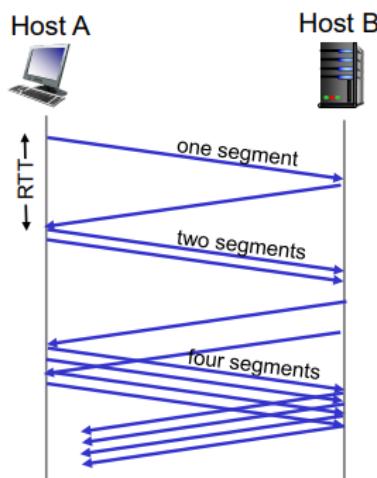
$$cwnd_0 = MSS$$

In questa fase l'aumento di cwnd è di tipo esponenziale.

Il mittente inizia trasmettendo il primo segmento dati, attendendo un riscontro.

La cwnd aumenta se dopo aver trasmesso un segmento ottiene il relativo ACK di risposta.

$$\text{A ogni RTT la cwnd raddoppia di dimensioni} \rightarrow cwnd_{i+1} = 2 * cwnd_i$$



AIMD - Valore di Soglia

Chiamato *ssthresh* → “Threshold” - “Valore di Soglia”.

Inizialmente *ssthresh* = 64 KByte.

AIMD - Fase di Congestion Avoidance

Si entra in questa fase quando: $cwnd \geq ssthresh$

In questa fase si ha un incremento additivo lineare:

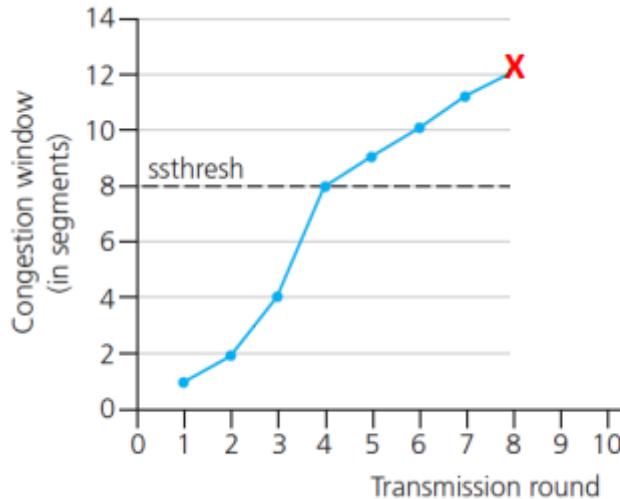
$$cwnd_{i+1} = cwnd_i + MSS * \left(\frac{MSS}{cwnd_i} \right)$$

AIMD - Fase di Fast Recovery

Fase in cui entro in caso di triplice ACK duplicato.

In questa fase si ha un incremento additivo lineare:

$$cwnd_{i+1} = cwnd_i + MSS * \left(\frac{MSS}{cwnd_i} \right)$$



AIMD - Sintomi di Congestione

I sintomi di congestione sono 2, non gravi allo stesso modo e noi vorremmo che AIMD si comportasse in modo diverso a seconda del sintomo rilevato.

AIMD - Triplice ACK Duplicato

Se ricevo un Triplice ACK duplicato, è il sintomo di congestione più lieve.

In questo caso:

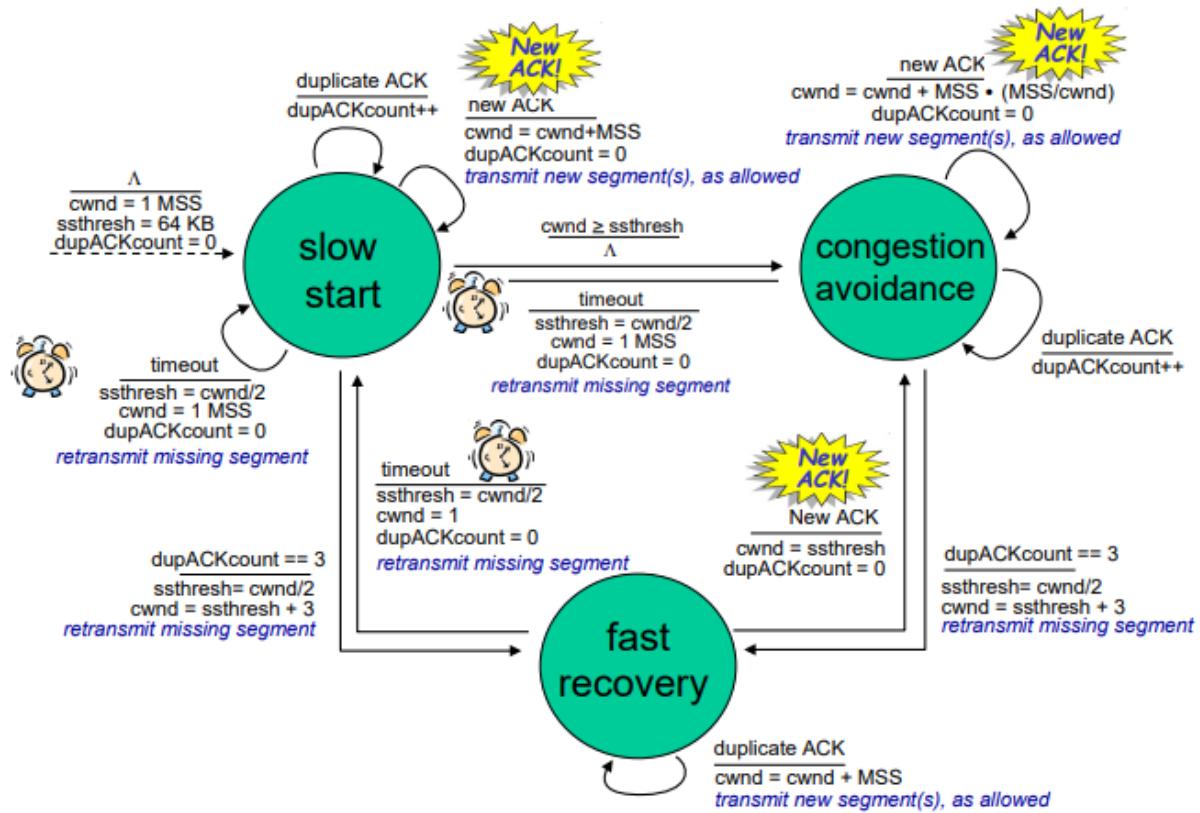
- $ssthresh = \frac{cwnd}{2}$
- $cwnd = \frac{cwnd}{2} + (3 * MSS)$
- Resetto il contatore di ACKs → $dubACKcount = 0$.
- Vado in “Fast Recovery”.

AIMD - Timeout Scaduto

Sintomo considerato più grave del triplice ACK.

In questo caso:

- $ssthresh = \frac{cwnd}{2}$
- $cwnd = MSS$
- Resetto il contatore di ACKs → $dubACKcount = 0$.
- Vado in “Slow Start”.

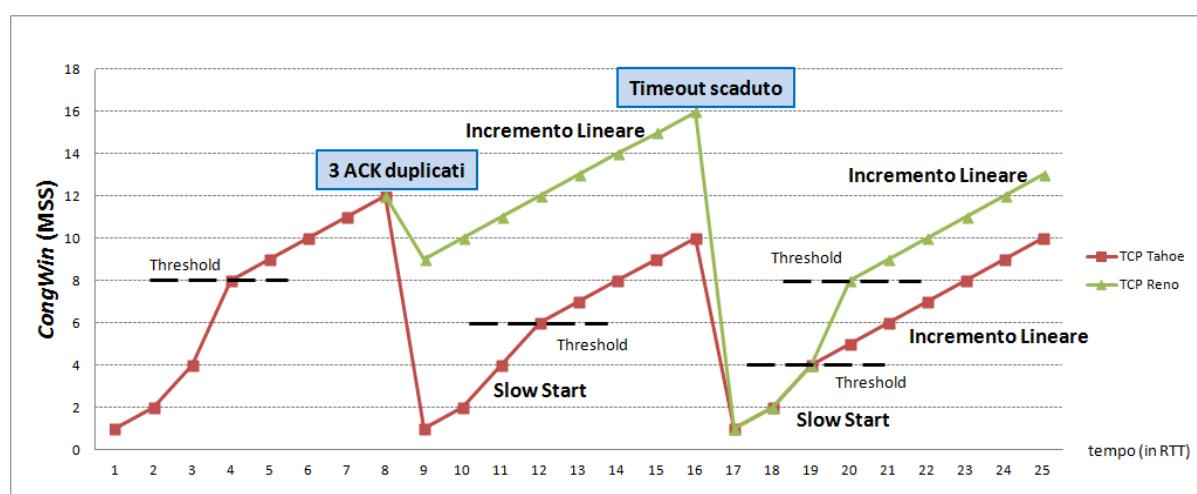


TCP Reno e TCP Tahoe

In TCP Reno si differenzia tra i due sintomi di congestione.

In TCP Tahoe invece non si fa nessuna differenza.

Le due versioni si differenziano per la reazione al triplice ACK duplicato.



TCP Fairness

Supponiamo di avere K connessioni TCP che condividono lo stesso canale di comunicazione con banda massima pari a R.

Un sistema è fair se tutte le connessioni prendono R/K .

Un sistema è efficiente se tutte le connessioni prendono R.

La fairness è bella ma può andare a discapito dell'efficienza.

Il TCP tende alla fairness, ma "gira intorno" al punto di massima fairness, cioè cerca un equilibrio tra efficienza e fairness.

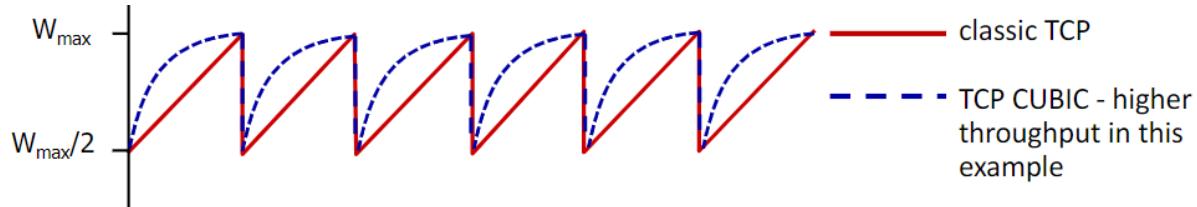
TCP Cubic

Versione alternativa di TCP, versione di default in LINUX.

Differisce dal TCP classico nella Congestion Avoidance.

In TCP Cubic l'aumento della finestra di congestione viene fatto in altro modo.

In questa versione si ha un approccio più greedy.

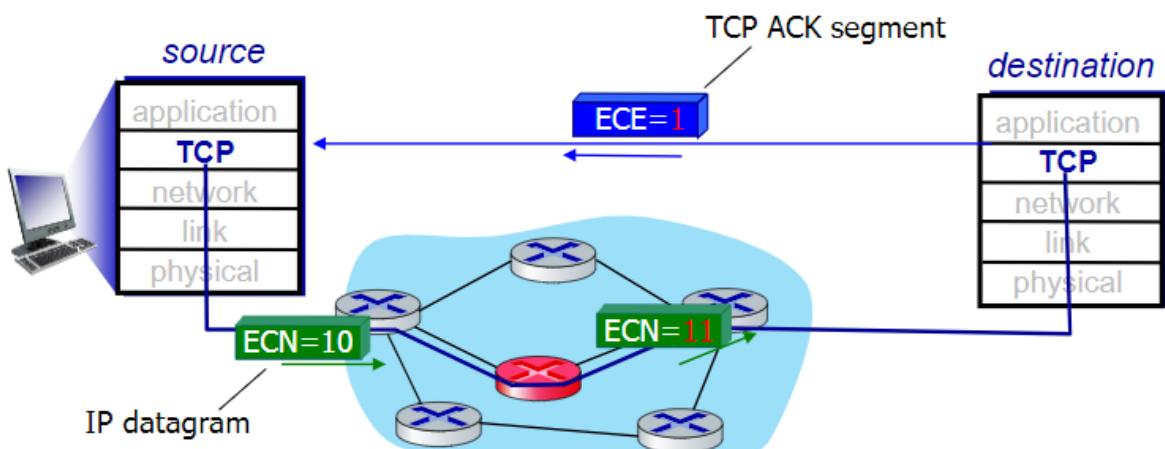


ECN - Explicit Congestion Notification

Metodo opzionale per rendere la congestion control di TCP più efficace.

Spesso le implementazioni di TCP possiedono anche una parte di *network-assisted congestion control*.

Ossia nel controllo della congestione anche i router collaborano, inviando notifiche esplicite.



Quando ECN viene negoziato con successo, un router *ECN-aware* può impostare un contrassegno nell'intestazione IP invece di eliminare un pacchetto per segnalare una congestione imminente.

Il destinatario del pacchetto invia l'indicazione di congestione al mittente, che riduce la velocità di trasmissione come se rilevasse un pacchetto scartato.

ECN sfrutta due bit <ECN , ECE> dell'intestazione di IP (il campo ToS).

- 00 → Trasporto non abilitato con ECN.
- 10 → Trasporto abilitato con ECN.
- 11 → Congestione incontrata.

TCP vs UDP

Su internet circola anche il traffico UDP oltre al traffico TCP.

UDP è best effort, non ha meccanismi di controllo o affidabilità, ergo la sorgente può spedire alla velocità che vuole.

Può capitare che il traffico UDP sia in numero estremamente superiore a quello TCP:

“UDP ruba la banda al TCP”

Chi progetta applicazioni TCP può aprire tante connessioni TCP e usarle contemporaneamente per recuperare più banda possibile, con questo metodo è dimostrabile che il rate aumenta.

Alcuni sistemi proibiscono o limitano questa pratica.

■ Come funziona l'apertura di una connessione TCP?

L'apertura di una connessione avviene tramite il Three Way Handshake.

1. Il client invia al server un segmento TCP con:
 - SYNbit = 1; e Seq = x;
 - Con x scelto randomicamente.
2. Il Server riceve la richiesta, dopo un pò la accetta e invia al client un segmento con:
 - SYNbit = 1; Seq = y; ACKnum = x+1;
 - Con y scelto randomicamente.
3. Il Client riceve la risposta e invia un ACK:
 - SYNbit = 1; Seq = x+1; ACKnum = y+1;

■ Come funziona la chiusura di una connessione TCP?

La chiusura di una connessione avviene tramite la procedura di teardown.

1. Il client invia al server un segmento TCP con:
 - FINbit = 1; e Seq = x;
 - Con x scelto randomicamente.
2. Il Server riceve la richiesta, dopo un pò la accetta e invia al client un segmento con:
 - FINbit = 1; Seq = y; ACKnum = x+1;
 - Con y scelto randomicamente.
 - Il Server dealloca le strutture dati.
3. Il Client riceve la risposta e invia un ACK:
 - FINbit = 1; Seq = x+1; ACKnum = y+1;
4. Il Client aspetta un intervallo di tempo.
5. Il Client dealloca le strutture dati.

■ Affidabilità protocollo TCP - Come viene garantita?

In TCP si applica la politica ARQ, in caso di errore in un pacchetto lo si scarta.

Il trasmettitore ha un timeout per ogni pacchetto inviato, allo scadere del quale se non si ricevono ACK avverrà l'immediata ritrasmissione.

■ Timeout TCP - Come si calcola?

Il timeout viene calcolato mediante una procedura che utilizza i Round Trip Time misurati per i pacchetti precedenti in quella stessa connessione.

Una di queste procedure potrebbe essere AIMD.

■ Timeout TCP - Perché la stima potrebbe essere troppo grande o troppo piccola?

Un timer troppo piccolo scatterebbe subito e spesso inondando la rete di pacchetti trasmessi e molto probabilmente duplicati.

Un timer troppo grande scatterebbe troppo tardi e quindi diminuirebbe la velocità complessiva della rete.

Controllo di flusso nel protocollo TCP- A cosa serve?

Il controllo del flusso serve quando il rate di invio dei segmenti lato sender è superiore al rate di prelievo dal buffer del socket di destinazione da parte dell'applicazione lato receiver.

Se questa cosa non venisse regolamentata prima o poi il buffer in questione si riempirà e si perderebbero dei pacchetti.

Cos'è la congestione?

Il fenomeno di congestione è quando i trasmettitori in una rete inviano troppi segmenti e i nodi intermedi della rete non riescono a gestirli in tempo, causando ritardi o perdite di pacchetto.

Congestione - Come e Dove si verifica?

La congestione avviene nei router, più precisamente nei buffer di entrata e nei buffer di uscita dei router.

Un router è congestionato quando il rateo di arrivo dei pacchetti è superiore al rateo di processazione e di inoltro.

Questo causa un ritardo di accodamento e nel peggiore dei casi perdite di pacchetti.

Congestione - Cosa comporta per gli host?

Ossia le prestazioni della rete calano.

Congestione TCP - Quanto vale MSS nella Congestion Window? come si calcola?

MSS (*Maximum Segment Size*) rappresenta la massima dimensione del corpo dati (*payload*) in un segmento TCP, da non confondersi con la massima dimensione di quest'ultimo, poiché questo è composto infatti da una parte di intestazione (header) e una appunto di corpo dati.

Ogni connessione TCP ha il suo MSS, negoziato nella fase di setup di una connessione TCP.

In IPv4, la MSS è tipicamente 1460 Byte, perché la MTU tipica è 1500 byte, a cui vanno sottratti 20 Byte di Header IP e 20 Byte di Header TCP.

In genere la MSS minima è di 536 Byte.

Differenza tra controllo di flusso e controllo di congestione?

Il controllo del flusso è un'operazione che fa il sender per "dare respiro" all'applicazione destinataria lato receiver.

Il controllo della congestione è un'operazione che fa il sender per "dare respiro" ai router della rete.

Perché si fa un trattamento diverso tra il timeout e il triplice ack?

Il triplice ACK nonostante sia considerato un sintomo di congestione è considerato più lieve. Questo perché se la rete ha prestazioni alte e il receiver pure non conviene aspettare lo scadere del timeout, infatti spesso prima che scada il timeout il receiver ha già inviato più di 3 ACK duplicati.

L'arrivo di un triplice ACK testimonia il fatto che comunque le prestazioni della rete non sono pessime e quindi non occorre prendere misure drastiche come quelle che si prenderebbero dopo un timeout scaduto.

Network Security

- **Confidenzialità.**
 - Se un mittente invia un messaggio solo il destinatario può leggerlo.
 - Il messaggio non può essere inviato in chiaro, quindi il mittente cifra il messaggio e il destinatario lo decifra.
 - Se le informazioni non sono critiche (di pubblico dominio come un IBAN o non sensibili tipo un ping) non serve cifrare il messaggio.
- **Autenticazione.**
 - Il mittente e il destinatario devono essere sicuri dell'identità dell'interlocutore.
 - Nessuno dovrebbe potersi fingere qualcun'altro.
- **Integrità del Messaggio.**
 - Un messaggio non deve poter essere alterato da malintenzionati.
 - L'alterazione in caso di errori di rete viene evitata dai protocolli di rete ma un'alterazione dovuta a manomissione di crittoanalisti invece no.
- **Accesso e Affidabilità**
 - Il sistema deve richiedere un controllo degli accessi e della disponibilità dei servizi.

Terminologia della Comunicazione

- **Alice**
 - Amante di Bob.
- **Bob**
 - Sposato con Trudi.
 - Amante di Alice.
- **Trudi**
 - Ha il ruolo di "Attaccante".
 - Moglie di Bob.
 - Incazzata **nera** con Alice.
 - Vuole provare che Bob è uno sporco peccatore che si è macchiato del crimine di adulterio.
 - Trudi può fare svariate cose cattive:
 - Può intercettare i messaggi scambiati → “*eavesdrop*”.
 - Può cancellare o modificare i messaggi.
 - Può Inserire messaggi falsi nella connessione → “*insert*”.
 - Può fingersi uno dei due → “*hijacking*”.
- **m**
 - Il messaggio che Alice e Bob vogliono scambiarsi, ma non può circolare così come è.
 - m è detto “messaggio in chiaro”.
- **c = K_A(m)**
 - c viene detto “critogramma” o “messaggio cifrato”, ed è effettivamente ciò che circola sul canale.

- Ottenuto cifrando il messaggio m con la “chiave di cifratura” K_A , usando un algoritmo di crittografia standard e quindi pubblici.
 - EXTRA → Gli algoritmi di cifratura militari non sono pubblici.
- $m = K_B(K_A(m))$, dove K_B è la “chiave di decifratura”.
- Se Trudi riceve c , per leggere m deve applicare tecniche di attacchi:

Approccio “Brute Force” o “Esaustivo”

In linea teorica funziona sempre.

Il crittoanalista (Eve) prende il crittogramma e prova tutte le chiavi possibili una dopo l'altra, prima o poi troverà quella giusta e otterrà il messaggio.

E' la tecnica più semplice ma anche quella più inefficiente perché l'attacco richiede di scorrere tutto lo spazio delle chiavi.

il “prima o poi” può essere un tempo molto lungo, anche nell'ordine di giorni o anni.

Approccio di Analisi

Posso applicare tecniche più sofisticate che vanno a colpire i punti deboli del cifrario usato.

Algoritmi di Crittografia a Chiave Simmetrica

La chiave di cifratura e la chiave di decifratura sono uguali, detta “K”.

L'algoritmo per cifrare il messaggio può essere diverso da quello per decifrare il crittogramma, ma la chiave è la stessa.

Il problema è *lo scambio della chiave*, che deve avvenire per forza lungo il canale insicuro.

$$c = C(m, K_{simm}) \quad m = D(c, K_{simm})$$

Cifrario a Sostituzione - “Cifrario di Cesare”

Sostituisco una lettera dell'alfabeto romano con un'altra lettera dello stesso alfabeto.

L'alfabeto romano ha 26 lettere.

La chiave è il numero di posizioni shiftate, ha 26 possibili valori.

Un computer general purpose con un algoritmo brute force lo rompe in pochi millisecondi.

Concatenare più volte il processo di sostituzione non aumenta la sicurezza.

Quando un Cifrario si dice “Rotto”

Un cifrario è detto “rotto” se è stato trovato un algoritmo o una tecnica che permette di decifrare un crittogramma in meno tempo di un attacco esaustivo.

DES - Data Encryption Standard

Chiave a 56 bit.

Blocchi di 64 bit, ogni blocco è cifrato/decifrato in modo individuale.

DES è considerato Rotto, qualsiasi messaggio cifrato con DES può essere decifrato con un attacco brute force in meno di un giorno.

Non ci sono attacchi analitici.

3DES - Triple DES

Concateno 3 cifrature DES con 3 chiavi diverse.

Più potente di DES.

AES - Advanced Encryption Standard

Blocchi a 128 bit.

Chiavi a 128, 192 o 256 bit.

AES non è ancora stato rotto.

Se per decifrare un messaggio DES usando un attacco brute force ci mettessi un secondo , in AES ci metterei svariati miliardi di anni.

Key Distribution Center - KDC

Strumento usato per lo scambio della chiave simmetrica da usare per cifrare i messaggi nella comunicazione tra due interlocutori.

Il problema dei cifrari a chiave simmetrica è la comunicazione della chiave.

KDC è una soluzione a questo problema.

Un KDC è un'entità fidata a cui si rivolgono sia Alice che Bob.

Mediando con il KDC entrambi ottengono in maniera sicura una chiave simmetrica.

KDC è un'organizzazione con diverse filiali nelle varie città/province.

Supponiamo che R1 sia la chiave simmetrica per la comunicazione tra Bob e Alice.

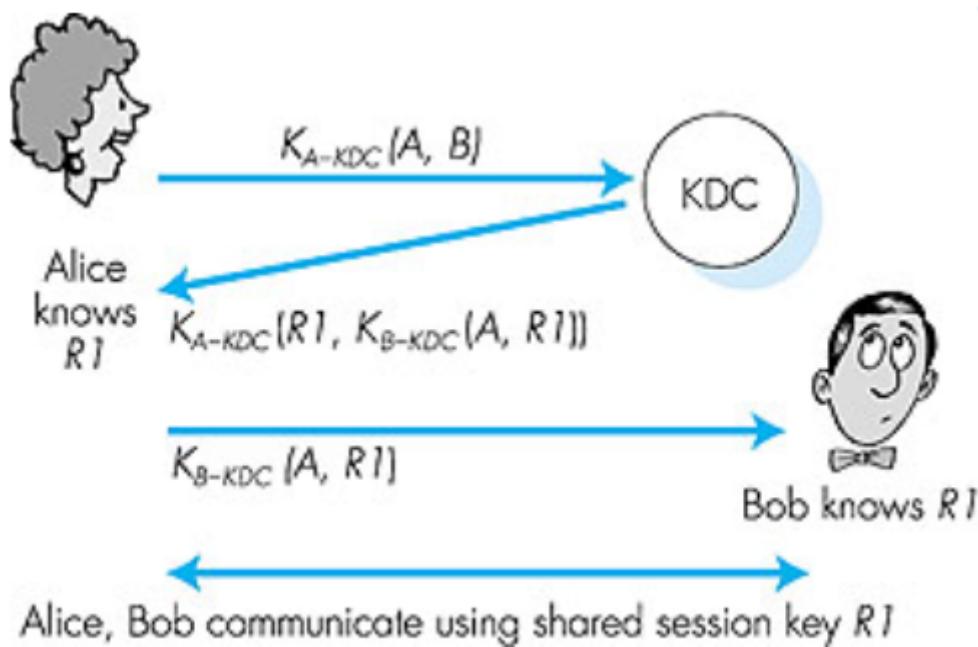
1. Alice si registra al KDC più vicino andandoci di persona.
2. Il KDC le rilascia una chiave simmetrica per la comunicazione tra Alice e KDC.
3. Bob si registra al KDC più vicino andandoci di persona.
4. KDC gli rilascia una chiave simmetrica per la comunicazione tra Bob e KDC.

Le chiavi tra KDC-Alice e KDC-Bob non hanno mai circolato nella rete, dato che ci sono andati di persona.

Sia R1 la chiave per la comunicazione tra Alice-Bob.

5. Alice vuole comunicare con Bob, ma le serve una chiave R1.
6. Alice si rivolge al KDC, gli manda un messaggio (con il quale dice che vuole comunicare con Bob) cifrato con la chiave KDC-Alice.
7. Il KDC decifra il messaggio e vede che Bob è registrato.
8. KDC prepara un messaggio dove mette:
 - a. La chiave R1.
 - b. Un Ticket, contenente R1 cifrata con la chiave KDC-Bob.
9. Il KDC cifra il messaggio con la chiave KDC-Alice.
10. Alice riceve il messaggio, lo decifra e ottiene < R1 , Ticket >
11. Invia il Ticket a Bob.
12. Bob riceve il messaggio, lo decifra e ottiene R1.

Ora sia Bob che Alice possiedono la chiave di sessione R1.



Chiave di Sessione

Chiave simmetrica usata per una sola sessione di comunicazione.

Il fatto che la si usi per una sola sessione di comunicazione ha a che fare con la sicurezza. Supponendo che due interlocutori usassero la stessa chiave simmetrica per tutte le loro comunicazioni, allora Trudi ha più tempo per scoprire la chiave e ha la possibilità di attuare alcune tipologie di attacchi.

Invece se la chiave simmetrica cambia ad ogni sessione, Trudi avrà a disposizione meno tempo e anche meno tipologie di attacchi perché i crittogrammi prodotti con la stessa chiave saranno in numero minore.

Quindi usare questo stratagemma rende tutto più sicuro ma più oneroso dal punto vista delle risorse, perché ogni volta devo generare una chiave nuova.

Crittografia a Chiave Asimmetrica

Detti anche Cifrari “a chiave pubblica”.

Non prevedono lo scambio della chiave.

Ogni utente ha una coppia di chiavi:

- Chiave Pubblica K_{Bob}^+
 - Nota a Tutti.
 - Serve per cifrare un messaggio destinato a Bob.
- Chiave Privata K_{Bob}^-
 - Nota solo a Bob.
 - Serve a Bob per decifrare un messaggio destinato a lui.

$$K_{Bob}^-(K_{Bob}^+(m)) = m = K_{Bob}^+(K_{Bob}^-(m))$$

Data una chiave pubblica, dovrebbe essere impossibile (o comunque molto difficile) ricavare la relativa chiave privata.

Il cifrario a chiave pubblica più famoso e usato è RSA.

Il Problema dei Cifrari Asimmetrici

I cfrari a chiave asimmetrica sono molto più lenti dei cfrari a chiave simmetrica.

RSA è cento volte più lento del DES a parità di lunghezza del messaggio.

Approccio Irido - Crittografia Asimmetrica per lo Scambio della Chiave Simmetrica

Data l'onerosità degli algoritmi a chiave asimmetrica si è pensato ad un trucchetto.

Posso usare un algoritmo a chiave pubblica per effettuare lo scambio della chiave di sessione.

La chiave è relativamente piccola (al massimo 256 bit) e quindi il tempo di esecuzione di RSA su tale stringa è trascurabile.

1. Alice produce la chiave di Sessione K.
2. Alice produce il crittogramma $c = K_{Bob}^+(K)$ e lo invia a Bob.
3. Bob riceve e decifra $K = K_{Bob}^-(K_{Bob}^+(K))$.
4. Bob e Alice hanno K.

Invece il messaggio può avere lunghezza pari a decine di migliaia di byte.

Integrità del Messaggio e della Sequenza

Spesso non serve che le informazioni rimangano segrete, ma è importante garantire che il contenuto non venga alterato.

Esempio Classico:

Bob vuole inviare il suo IBAN ad Alice, non serve cifrare il messaggio poiché l'IBAN è un'informazione di dominio pubblico.

Ma è assolutamente necessario che nessuno possa intercettare il messaggio e modificare l'IBAN, magari per mettere il suo al posto di quello di Bob.

L'Integrità del Messaggio è soddisfatta se l'algoritmo garantisce questi 4 punti:

- **Identità del mittente verificabile.**
 - Bob vorrebbe essere sicuro che il messaggio sia stato mandato da Alice.
- **Contenuto del messaggio non alterabile**
 - Vorrei che il messaggio arrivasse nelle stesse condizioni nelle quali era quando è stato inviato.
- **Messaggio non replicabile**
 - L'attaccante invia a Bob lo stesso messaggio che Alice inviò in passato.
- **Sequenza non alterabile**
 - In caso di sequenza di messaggi, l'ordine non deve essere scambiato, poiché potrebbe causare il fallimento nell'esecuzione di un applicativo.

Message Digest - “Impronta del Messaggio”

Un digest è una “sintesi” del messaggio.

Si ottiene facendo passare il messaggio in una funzione *Hash* : digest → H(m).

Di solito il digest ha lunghezza decisamente inferiore rispetto al messaggio originale.

Caratteristiche della funzione Hash:

- **Deterministica**
 - Dato un m, H(m) deve restituire sempre lo stesso risultato.
- **H(m) deve essere una funzione “one-way”**
 - **Facile da calcolare**
 - Deve essere computazionalmente facile calcolare H(m) partendo da m.
 - **Difficile da invertire**
 - Deve essere computazionalmente difficile calcolare m partendo da H(m).
- **Resistente alle Collisioni - “Collision Resistant”**
 - Deve essere difficile trovare 2 messaggi diversi con lo stesso digest.
 - Dati 2 messaggi diversi ma molto simili tra loro (magari che differiscono per un solo carattere), H(m) restituisce 2 digest completamente diversi.
- **Il Digest deve sembrare una sequenza randomica.**
 - H(m) è comunque deterministica e quindi il digest non può essere effettivamente una sequenza randomica, però dovrebbe sembrarlo.

Funzioni Hash più popolari

Tra le funzioni hash più usate abbiamo:

- MD5 → Digest a 128 Bit.
- SHA → Digest a 160 Bit.

MAC - Message Authentication Code

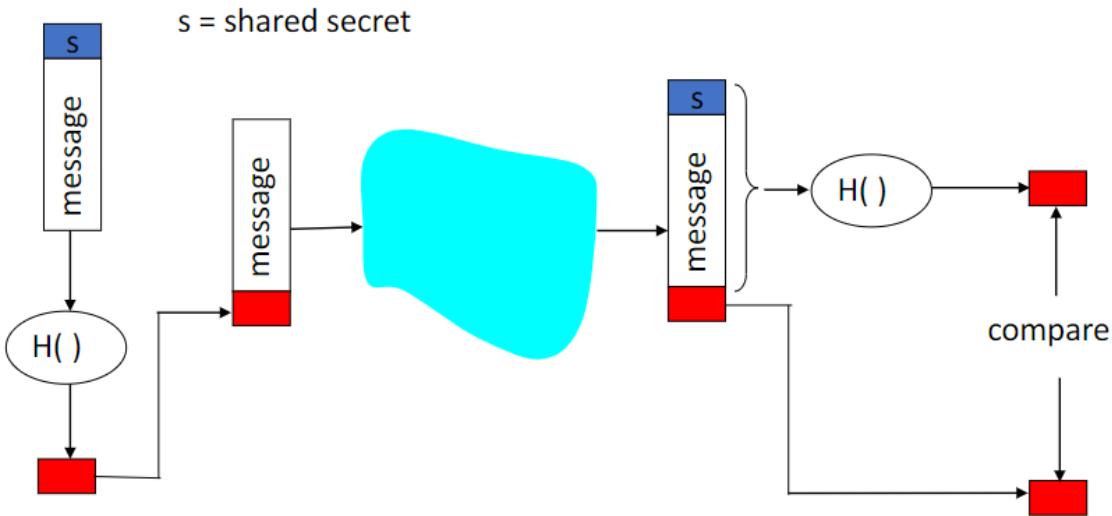
Il MAC è una tecnica che sfrutta le funzioni Hash che mi permette di:

- **Verificare l'integrità del messaggio.**
- **Autenticare il Mittente del messaggio.**
 - Attenzione! → Non permette al Mittente di autenticare il Destinatario.

Alice e Bob concordano un segreto (che solo loro due conoscono) e una funzione di Hash.

1. Alice concatena messaggio e segreto → < m , s >
 - s viene può essere considerata come la “*Chiave del MAC*”.
2. Alice calcola $MAC = H(\langle m , s \rangle)$
3. Alice invia < m , MAC > a Bob.
4. Bob riceve < m' , MAC' >.
5. Se $H(\langle m' , s \rangle) = MAC'$ → Test Superato!

Il messaggio m può essere anche cifrato se siamo interessati anche alla confidenzialità.



Protocollo HMAC - RFC 2104

Tecnica molto popolare per produrre il MAC.

1. Alice concatena il segreto e il messaggio.
a. $\langle s, m \rangle$
2. Alice calcola il Digest.
a. $\text{Digest} = H_1(\langle s, m \rangle)$
3. Alice concatena il segreto con il Digest.
a. $\langle \text{Digest}, s \rangle$
4. Si calcola l'Hash e ottengo il MAC.
a. $\text{MAC} = H_2(\langle \text{Digest}, s \rangle)$

$$\text{MAC} = H_2(\langle H_1(\langle m, s \rangle), s \rangle)$$

La funzione di Hash usata al punto 2 può differire da quella usata al punto 4.

Attacco “Record and Playback”

1. Trudi vede passare il messaggio che Bob invia ad Alice.
2. Trudi copia il messaggio e se lo salva in memoria.
3. Dopo un pò di tempo Trudi replica il messaggio e lo invia a Bob.
4. Bob riceve il messaggio di trudi pensando che venga da Alice.

Bob viene raggiunto, perché non sa che è stata Trudi a inviare il Messaggio.

Esempio Classico di “Record and Playback”

Trudi intercetta un messaggio dove Bob invia un bonifico ad Alice, Trudi lo memorizza e in un secondo momento lo invia ad Alice, causando un ulteriore bonifico dal conto di Bob al conto di Alice.

MAC vs Attacco “Record and Playback”

MAC non protegge da questo tipo di attacchi.

Bob controlla il MAC.

Il MAC è stato creato da Alice e dato che Trudi non ha alterato il messaggio (lo ha solo trasmesso di nuovo) il messaggio passa il controllo.

Perché il messaggio non è “fresco” ma è un messaggio passato e Bob non ha modo di capire se il messaggio è nuovo o vecchio.

Soluzione al “Record and Playback” - Nonce

Bob invia (in chiaro o cifrato) ad Alice un “Nonce” chiamato anche “*Token di Sessione*”.

Tipicamente il Nonce è un numero Intero generato randomicamente e valido solo per quella sessione.

Il Nonce (R) entra in gioco nel calcolo del MAC insieme a m e ad s.

MAC = H(< msg , s , R >)

Questa tecnica ci permette di evitare che Trudi effettui questo tipo di attacchi dopo lo scadere della sessione tra Bob e Alice.

Non ci protegge da un attacco che avviene mentre la sessione di comunicazione è ancora in corso.

OTP - “OneTime Password”

Metodo moderno per garantire la freschezza del messaggio.

Una OTP è una password (generata con un algoritmo pseudo randomico) valida solo per una singola sessione di accesso o una singola transazione.

La OTP evita una serie di carenze associate all'uso della tradizionale password (statica).

Il più importante problema che viene risolto da OTP è che, al contrario della password statica, essa non è vulnerabile agli attacchi “Record and Playback”.

Ciò significa che, se un potenziale intruso riesce a intercettare una OTP che è stata già utilizzata per accedere a un servizio o eseguire una transazione, non sarà in grado di utilizzarla, in quanto non sarà più valida.

D'altra parte, una OTP non può essere memorizzata da una persona in quanto dopo l'utilizzo sarebbe inutile.

Essa richiede quindi una tecnologia supplementare per poter essere utilizzata (un dispositivo fisico con la calcolatrice OTP incorporata o un numero di cellulare specifico o una app installata sul dispositivo certificato).

Mantenere la Sequenza di Messaggi

Voglio evitare che Trudi esegua uno swapping dei messaggi (cambiarne l'ordine) e poter rendere impossibile ricostruire il messaggio originale al destinatario.

Nota, se per scambiare l'ordine dei messaggi l'attaccante semplicemente modifichasse il numero di sequenza allora questa modifica verrebbe facilmente rilevata dal MAC, dato che quest'ultimo può essere calcolato su tutto il messaggio (intestazione e payload).

Firma Digitale - “Digital Signature”.

Un algoritmo di firma digitale deve avere le seguenti proprietà:

- **Non Falsificabile (Verifiable)**
 - Bob può provare che un documento sia stato firmato da Alice e da nessun altro.

- **Non Replicabile (Non Forgeable)**
 - Bob può provare che qualcun'altro ha firmato il messaggio e che non è stata Alice.
 - Nessuno può falsificare la firma di qualcun'altro.
- **Non Ripudiabile (Non Repudiation)**
 - Alice non può negare di aver inviato un messaggio inviato effettivamente da lei.
 - Allo stesso modo Alice può provare di non aver mai inviato un messaggio che non è stato effettivamente inviato da lei.
- **Integrità del Messaggio (Message Integrity)**
 - Alice può provare che lei ha firmato un certo messaggio m e non m' (supponendo che m' sia stato ottenuto modificando m).

Firma Digitale con MAC

Il MAC può fungere da firma digitale? **No, soddisfa 1/4 requisiti.**

Supponiamo che il MAC venga calcolato concatenando un segreto comune.

1. Non è Verificabile.
 - Se Alice e Bob inviassero lo stesso messaggio m , il MAC prodotto da Bob e Alice sarebbe lo stesso.
 - In una sessione tra Bob e Alice, un messaggio di Bob ha la stessa firma di Alice, quindi non è possibile provare che un certo messaggio è stato prodotto da Alice o da Bob.
2. È Replicabile.
 - Per il punto 1, Bob può replicare la firma di Alice.
3. È Ripudiabile
 - Per il punto 1, il MAC è facilmente replicabile, quindi dato che non è possibile associare una certa firma univocamente ad una certa persona, automaticamente il messaggio in questione diventa ripudiabile.
4. Il MAC garantisce l'integrità del messaggio.
 - Se il messaggio viene modificato il controllo del digest fatto dal destinatario fallirà.

Firma Digitale con Crittografia Asimmetrica

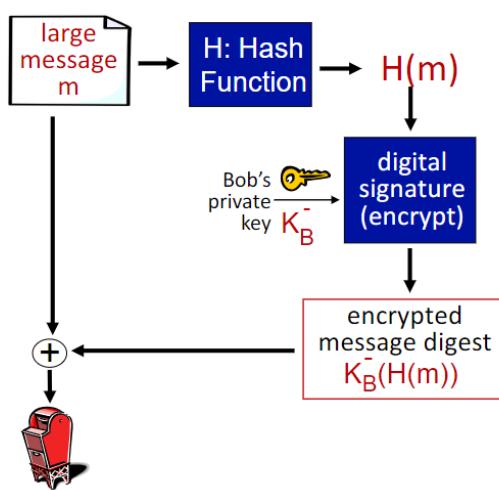
1. Bob calcola la firma.
 - $f = K_{Bob}^-(m)$
2. Bob spedisce ad Alice
 - $< m, f >$, con m in chiaro o cifrato.
3. Alice riceve.
 - se m è cifrato lo decifra.
4. Alice usa la chiave pubblica di Bob per decifrare la firma.
 - $m' = K_{Bob}^+(f)$
5. Se $m = m' \rightarrow$ Firma verificata correttamente.

Firma Digitale con Crittografia Asimmetrica - Ottimizzata

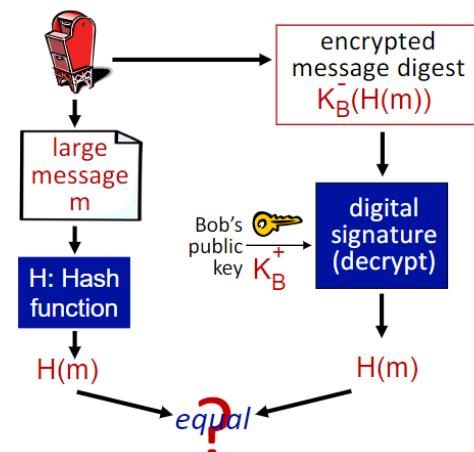
Dato che RSA è lentissimo, per ottimizzare potrei firmare qualcosa di più piccolo del messaggio, ad esempio un digest calcolato con una funzione di Hash.

1. Bob calcola $H(m)$
 - o La quale avrà dimensione molto minori rispetto a m.
2. Bob calcola la firma.
 - o $f = K_{Bob}^-(H(m))$
3. Bob spedisce ad Alice
 - o $< m, f >$, con m in chiaro o cifrato.
4. Alice riceve:
 - o $< m', f' >$, se m' è cifrato lo decifra.
5. Se $H(m') = K_{Bob}^+(f')$ → Firma verificata correttamente.

Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:



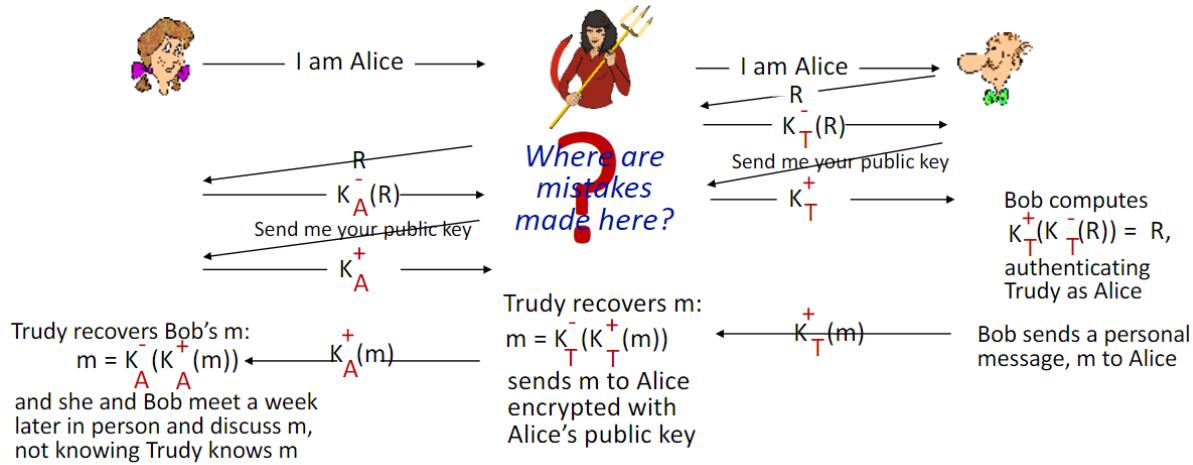
Il meccanismo appena presentato può fungere da firma digitale? **Si, soddisfa 4/4 requisiti.**

- 1. E' Verificabile.**
 - o Il destinatario può provare che una firma appartiene ad un certo utente o meno.
- 2. Non è Replicabile.**
 - o La firma è prodotta usando la chiave privata dell'utente, dato che solo lui la conosce, automaticamente solo lui può produrla.
- 3. Non è Ripudiabile.**
 - o Per il punto 2.
- 4. Garantisce l'integrità del messaggio.**
 - o La firma è calcolata usando un digest del messaggio come base, se il messaggio o la firma venissero modificati in qualche modo allora la verifica fatta al punto 5 fallirà.

Certificato Digitale

Supponiamo che Alice voglia ricevere la chiave pubblica di Bob.

Trudi potrebbe fingersi Bob inviando la sua chiave pubblica invece che quella di Bob, e attuare un attacco di tipo “*Man in the Middle*”.



Si potrebbe risolvere il dilemma tramite un certificato, garantito da terzi (*Certification Authority*), il quale assicura che una certa chiave pubblica appartiene ad un certo utente o meno.

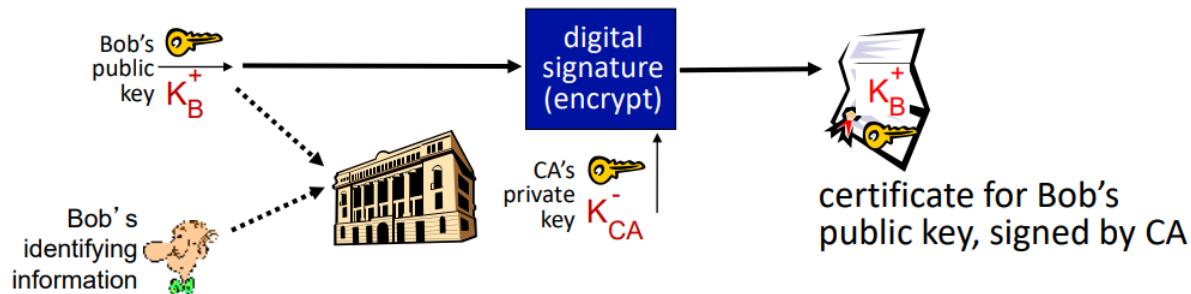
In questo modo l'autenticazione di Bob (quindi il fatto che una certa chiave pubblica appartenga effettivamente a Bob) è garantita dalla CA.

Un certificato digitale è un messaggio che contiene varie informazioni su uno specifico soggetto (Bob):

- Certificato
 - Versione del Certificato.
 - Numero seriale del Certificato.
 - ID dell'algoritmo.
 - Ente emittitore (*La Certification Authority*)
 - Validità:
 - Non prima di [timestamp]
 - Non dopo di [timestamp]
 - Soggetto (*Bob*)
 - Informazioni sulla chiave pubblica del soggetto
 - Algoritmo per l'utilizzo della chiave pubblica.
 - **Chiave Pubblica del Soggetto.**
 - Codice identificativo univoco dell'emittente (*facoltativo*)
 - Codice identificativo univoco del soggetto (*facoltativo*)
 - Estensioni (*facoltativo*)
- Algoritmo di firma del certificato
- Firma della Certification Authority (*Calcolata basandosi sul certificato*).

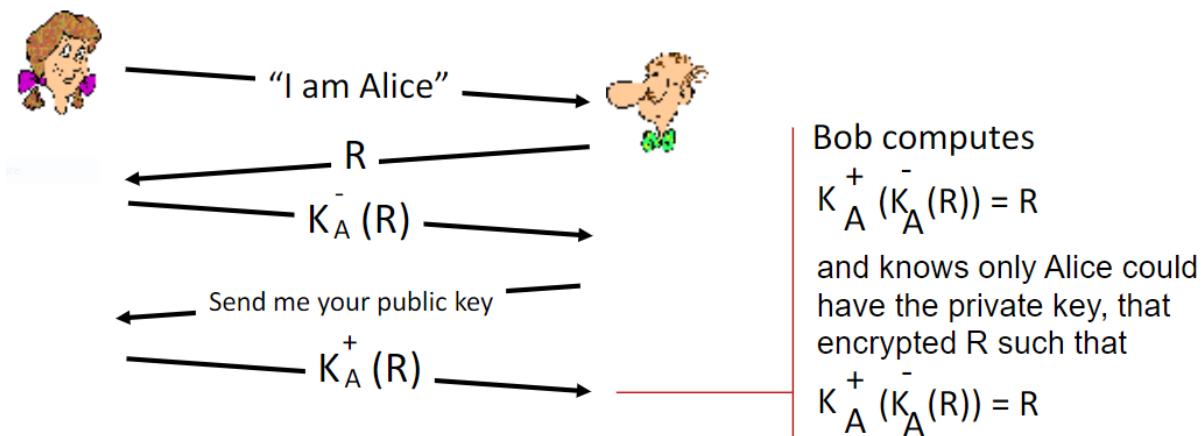
Il certificato viene incapsulato in un messaggio cifrato con la chiave privata della CA e destinato a Bob.

Dato che la CA è considerata “*trustable*”, ovvero: la sua chiave pubblica la conoscono tutti, è facile sgamare qualcuno che si finge una CA.



Protocollo di Autenticazione

Bob vuole provare l'identità dell'interlocutore, per verificare che non sia Trudi.



Ancora meglio se la chiave pubblica di Alice è fornita da una CA.

Implementazioni della Sicurezza

Spesso tutto ciò che riguarda la sicurezza viene implementato direttamente a livello applicazione, questo perché gli sviluppatori sono coloro che conoscono bene ciò di cui ha bisogno una certa applicazione, e quindi per garantire la massima flessibilità devono essere loro a provvedere ai meccanismi di sicurezza.

Questa cosa, come vedremo dopo, non è del tutto vera.

PGP - Secure E-Mail - Confidentialità

Alice vuole inviare una mail confidenziale a Bob.

Il messaggio può essere molto lungo, quindi preferiamo cifrarlo con un cifrario simmetrico. Mentre lo scambio della chiave simmetrica lo facciamo con un cifrario asimmetrico.

Supponiamo che Alice abbia preso la chiave pubblica di Bob da un certificato, quindi Alice è sicura che solo Bob può leggere K.

1. Alice invia $< C(m, K_{simm}), K_{Bob}^+(K_{simm}) >$.
2. Bob calcola $K_{simm} = K_{Bob}^-(K_{Bob}^+(K_{simm}))$.
3. Bob decifra $m = D(C(m, K_{simm}), K_{simm})$.

PGP - Secure E-Mail - Autenticazione e Integrità

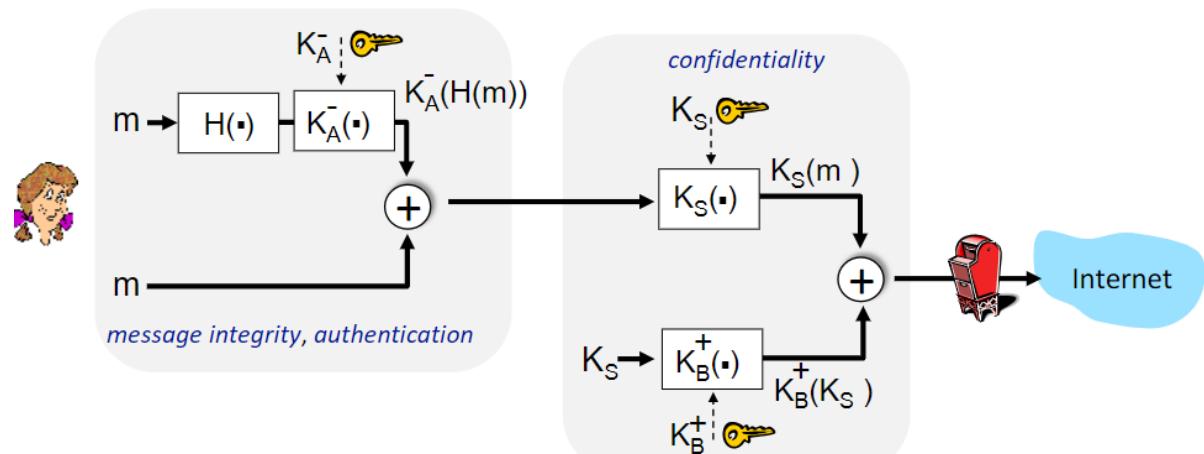
Alice vuole essere sicura che i messaggi arrivino integri.

Bob vuole essere sicuro che i messaggi arrivino effettivamente da Alice e che siano integri.

1. Alice invia $< m, K_{Alice}^-(H(m)) >$
2. Bob calcola $x = K_{Alice}^+(K_{Alice}^-(H(m))) = H(m)$, calcola il digest di m .
3. Se $x = H(m) \rightarrow$ Integrità e Autenticazione sono garantite.

L'autenticazione è garantita perché solo Alice può generare $K_{Alice}^-(H(m))$.

L'integrità è garantita perché insieme al messaggio viene inviato un digest calcolato sulla base di m .



PGP - Secure E-Mail - Versione Finale (Autenticazione + Integrità + Confidenzialità)

1. Alice produce $M = < m, K_{Alice}^-(H(m)) >$
2. Alice produce $C = < C(M, K_{simm}), K_{Bob}^+(K_{simm}) >$ e lo invia.
3. Bob calcola $\rightarrow K_{simm} = K_{Bob}^-(K_{Bob}^+(K_{simm}))$.
4. Bob decifra $\rightarrow M = D(C(M, K_{simm}), K_{simm})$.
5. Bob ora ha $M' = < m', K_{Alice}^-(H(m)) >$

6. Bob calcola $\rightarrow X = K_{Alice}^+ (K_{Alice}^- (H(m))) = H(m)$
7. Se $X = H(m')$ \rightarrow OK.

Discussione sulla sicurezza implementata a livello applicazione

I pro e contro di implementare i servizi di sicurezza nel livello applicazione:

Il vantaggio è che il programmatore sa cosa interessa all'applicazione e cosa non interessa, ad esempio un app può necessitare di integrità ma non di confidenzialità.

Quindi il programmatore ha la massima possibilità di personalizzazione.

Lo svantaggio è che il servizio di sicurezza deve essere implementato per ogni applicazione.

Se ho 4 applicazioni devo implementare 4 servizi di sicurezza.

Quindi aggiungiamo un altro livello alla pila protocollare, che si occupa di tutto ciò che riguarda la sicurezza.

TLS - Transport Layer Security

Livello intermedio tra il livello di applicazione e il livello di trasporto.

Prevede un trasporto sicuro su una connessione TCP.

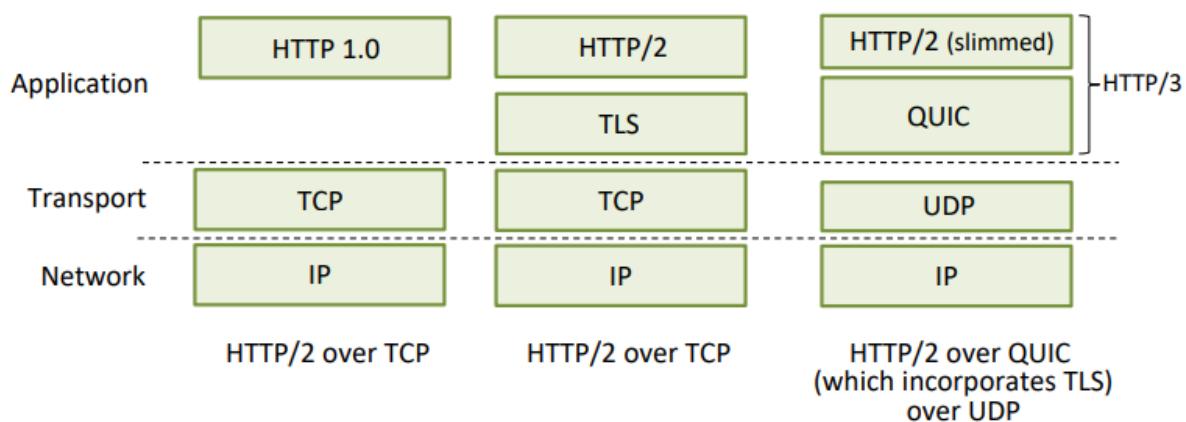
TLS implementa i sistemi di sicurezza a livello middleware e sono disponibili a tutte le applicazioni.

Le app sono libere anche di usare TCP senza sicurezza.

TLS ha sostituito SSL (obsoleto dal 2015).

TLS garantisce.

- **Confidenzialità** \rightarrow Cifratura Simmetrica del Messaggio.
- **Integrità** \rightarrow Tramite MAC fatto con Hashing.
- **Autenticazione** \rightarrow Crittografia a Chiave Pubblica.



TLS "Semplificato"

Una sessione TLS si compone da 4 fasi:

1. Handshaking

- Alice e Bob si scambiano:
 - Certificati \rightarrow per autenticarsi a vicenda, a volte solo uno dei due invia il certificato.

- Condividono un segreto , detto “*Master Secret*”.

2. Derivazione delle Chiavi

- Alice e bob usano una funzione concordata con la quale partendo dal Master Secret ottengono le chiavi per la cifratura simmetrica.

3. Trasferimento Dati

4. Chiusura Sessione TLS

- Diversa dalla chiusura della connessione TCP.

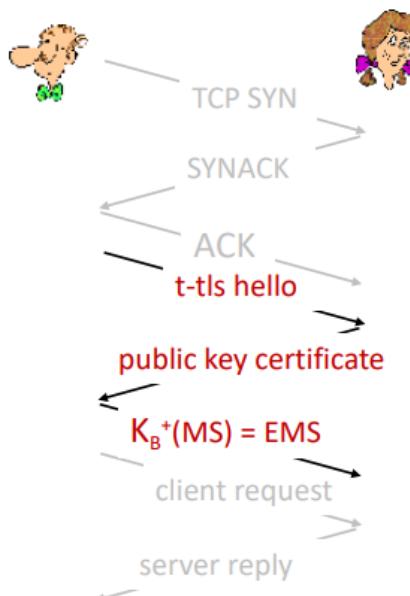
TLS - Handshake

Diverso dall' handshake del TCP.

Lo Handshake del TLS serve per far scambiare tra Bob e Alice un segreto comune che servirà successivamente per la generazione delle chiavi.

1. Bob stabilisce una connessione TCP con Alice.
2. Bob invia un “*TLS Hello*”.
3. Alice invia a Bob il suo certificato.
4. Bob verifica il certificato.
5. Bob prende la chiave pubblica di Alice dal certificato.
6. Bob crea un “*Master Secret*”, una sequenza randomica.
7. Bob invia ad Alice il “*Master Secret*” cifrato con la chiave pubblica di Alice.
8. Alice decifra il “*Master Secret*”.
9. Sia Bob che Alice hanno il *Master Secret*.

Prima che il client sia pronto a ricevere dati di solito passano almeno 3 RTT (includendo Handshake TLS e TCP).



TLS - Generazione delle Chiavi

Dopo l'handshake TLS Alice e Bob hanno il segreto comune, il “*master secret*”.

Partendo dal master secret (e optionalmente qualche dato random aggiuntivo) viene applicata una KDF (*Key Derivation Function*) la quale mi permette di generare 4 chiavi:

1. **Kc**
 - Per cifrare i dati da mandare dal Client verso il Server.
2. **Mc**

- Chiave del MAC per i dati da mandare dal Client verso il Server.

3. Ks

- Per cifrare i dati da mandare dal Server verso il Client.

4. Ms

- Chiave MAC per i dati da mandare dal Server verso il Client.

TLS - Integrità

Ogni messaggio tra client e server contiene un MAC.

$\text{MAC} = H(\langle \text{Dati}, \text{Lunghezza Dati}, \text{Numero Seq}, \text{Ms/Mc} \rangle)$

Calcolato con Ms se il messaggio è da server verso il client o Mc viceversa.

Record TLS

Il pacchetto a questo livello vengono chiamati Records.

Byte	+0	+1	+2	+3
0	Content Type		N/A	
1..4	Legacy version		Length	
5..n		Payload		
n..m		MAC		
m..p		Padding (block ciphers only)		

Il Tipo (*content type*) è un bit che esprime la tipologia di messaggio.

- 0 → Record contenente i Dati.
- 1 → Record di chiusura connessione TLS.

Il payload è il messaggio che viene dal livello applicazione.

Il numero di sequenza non viene esplicitamente inviato con il pacchetto, ma è inserito nel MAC e viene “memorizzato” in locale dal mittente e dal ricevitore.

Infine il record TLS viene cifrato con Ks se il flusso è da Server a Client o Kc viceversa.

Al record si aggiunge un numero di sequenza (perché il numero di sequenza del TCP è in chiaro e quindi facilmente manipolabile), il numero di sequenza TLS è incorporato nel MAC. Quindi se si cambia il numero di sequenza del TLS il controllo del MAC fallirà.

- I record sono cifrati con la chiave simmetrica → *Confidenzialità*
- I record sono affiancati da un MAC → *Integrità*
- I record sono affiancati da un Numero di Sequenza TLS → *Non si può riordinare la sequenza*.



Nota sul numero di sequenza TLS

Il livello TLS è posizionato tra il livello applicazione e trasporto, quindi il messaggio non è stato ancora segmentato.

Il numero di sequenza fa riferimento, durante una sessione, all'ordine dei messaggi "interi".

Il numero di sequenza del TLS è come quello del TCP, ma quest'ultimo è in chiaro nell'intestazione.

TLS - Chiusura Connessione

Dobbiamo evitare che trudi chiuda la connessione tra Alice e Bob inviando a entrambi un pacchetto di fine connessione TCP.

Il problema si evita imponendo una regola:

La connessione TCP può essere chiusa solo se la Connessione TLS è stata precedentemente chiusa.

TLS - Cipher Suite

La suite di cifratura rappresenta le specifiche con cui viene svolto TLS:

- Algoritmo per generare le chiavi partendo dal segreto comune.
- Algoritmo di cifratura a chiave pubblica.
- Algoritmo di cifratura a chiave simmetrica.
- Algoritmo di creazione del MAC.

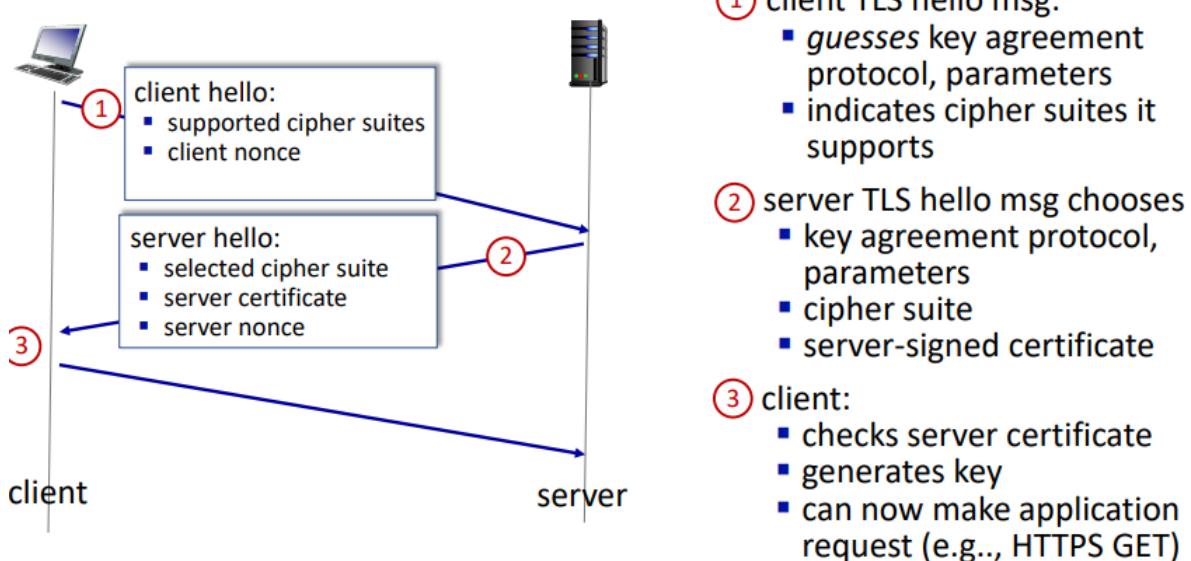
Il client dice al server tutti gli algoritmi che supporta, il server li sceglie e li comunica al client.

Gli algoritmi scelti dal server compongono la cipher suite.

TLS - Handshake "Vero"

Trudi non può replicare il singolo messaggio, ma può Trudi può replicare l'intera sessione.

1. Client Hello
 - a. Manda la Cipher Suite.
 - b. Manda il Client Nonce (Numero Random).
2. Il Server riceve
 - a. Sceglie gli algoritmi tra quelli che il client supporta.
3. Server Hello
 - a. Manda la suite scelta.
 - b. Manda il certificato del Server.
 - c. Manda il Server Nonce (Numero Random).



Sicurezza a Livello Network - IPsec

Posso implementare la sicurezza a livello IP, aggiungendo un nuovo protocollo IPsec.

IPsec fornisce:

- Confidenzialità.
- Autenticazione.
- Integrità.

IPsec ha 2 modalità di lavoro:

- **Transport Mode**
 - Solo il payload del datagram è cifrato e autenticato, l'intestazione IP rimane in chiaro.
 - Il MAC è calcolato solo sul payload e non sull'intestazione IP.
 - Computazionalmente più leggero del secondo.
 - Usato dagli endpoint, non dai gateway.
 - I Gateway vedono solo normale un pacchetto IP (dato che solo il payload è cifrato).
 - Gli endpoint devono implementare IPsec per decifrare il payload.
- **Tunnel Mode**
 - Tutto il datagram è cifrato e autenticato.
 - Il MAC è calcolato su tutto il datagram.
 - Ma se è tutto cifrato i router non possono leggere gli indirizzi e quindi diventa "*un routable*", quindi per risolvere il problema lo incapsuliamo con un nuovo header IP, chiamato "*new IP header*".
 - Questa modalità nasconde tutte le informazioni dal livello network in su.
 - Computazionalmente più oneroso del primo.
 - Usato esclusivamente dai gateway, non dagli endpoint.
 - Gli endpoint non si accorgono nemmeno della presenza di IPsec, dal loro punto di vista si sta usando il protocollo IP e basta.

- Solo il router della rete del mittente e il router della rete del destinatario è necessario che implementino IPsec, i quali rispettivamente cifrano il datagram all'uscita e decifrano all'arrivo.
- I router intermedi vedranno solo gli header di IPsec aggiunti dopo il tunneling fatto dal router della rete del mittente.



transport mode:

- *only* datagram *payload* is encrypted, authenticated

tunnel mode:

- entire datagram is encrypted, authenticated
- encrypted datagram encapsulated in new datagram with new IP header, tunneled to destination

VPN - Virtual Private Networks

Una applicazione concreta di IPsec usando la tunnel mode.

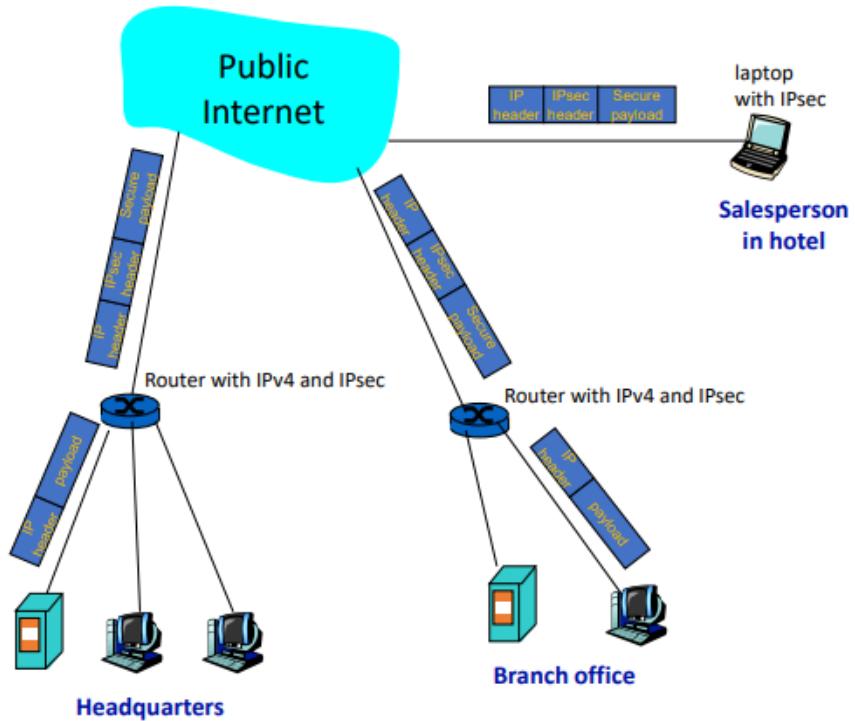
Le istituzioni spesso vorrebbero reti private per avere la massima sicurezza, ma è costoso!

Con una VPN , posso far girare il traffico interno lungo la rete pubblica senza rinunciare alla sicurezza.

Il traffico in uscita dal router del mittente viene cifrato e poi verrà decifrato sul router del destinatario.

Il mittente e il destinatario non si accorgono di nulla, dal loro punto di vista il traffico avviene in chiaro.

Tramite una VPN posso collegarmi alla rete istituzionale dove lavoro direttamente da casa e lungo il percorso nessuno potrà vedere il contenuto dei datagram.



Come è strutturato IPsec

A differenza di IP, IPsec è “*connection oriented*”.

IPsec è costituito da 2 protocolli:

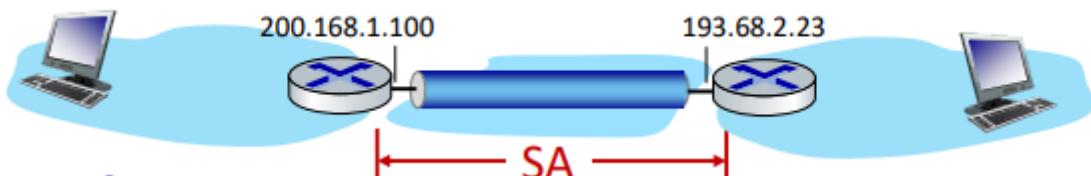
- **AH - Authentication Header**
 - Fornisce autenticazione e integrità ma non confidenzialità.
- **ESP - Encapsulation Security Protocol**
 - Fornisce autenticazione e integrità e confidenzialità.
 - Molto più usato rispetto ad AH.
 - Noi descriviamo ESP, dato che fornisce le stesse cose di AH e anche di più.

SAs - Security Associations.

Una SA è una specie di connessione tra due router.

Una SA ha lo scopo di gestire tutti i datagrammi che viaggiano tra i due router.

Prima di inviare i dati, viene stabilita una SA dal router del mittente al router del destinatario.



Le SA sono unidirezionali.

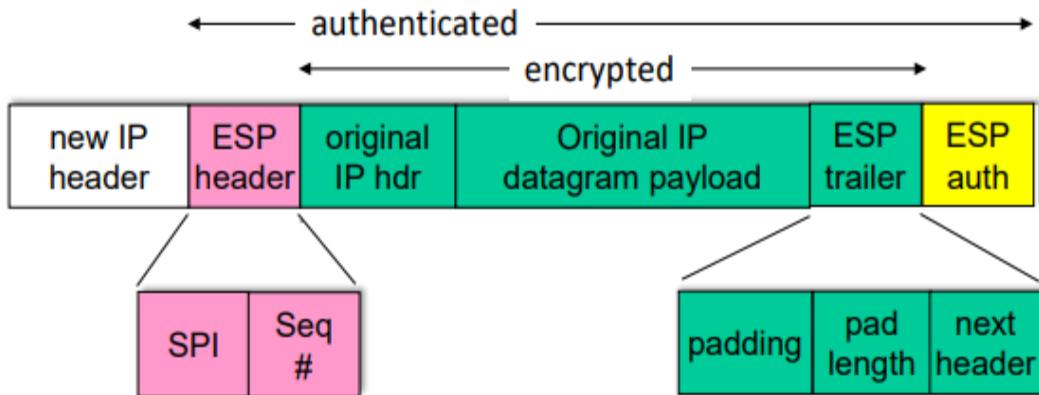
Quindi per avere un flusso bidirezionale devo stabilire due SA.

I Parametri di una SA

- **Security Parameter Index (SPI)**
 - Identificatore a 32 Bit.

- Un Router origine può avere molteplici SA attive contemporaneamente, quindi serve un modo per identificarle in modo univoco nella macchina.
- **Interfaccia di Origine**
 - Indirizzo IP del Router Mittente (200.168.1.100).
- **Interfaccia di Destinazione**
 - Indirizzo IP del Router Destinatario (193.68.2.23).
- **Algoritmo di Cifratura Usato.**
- **Chiave di Crittografia**
- **Algoritmo usato per generare e verificare il MAC.**
- **Chiave per l'autenticazione.**

Struttura del Datagram di IPsec - Tunnel Mode



- **New IP Header**
 - Header IP aggiunto dopo aver cifrato tutto il datagram originale.
- **SPI (Security Parameter Index)**
 - Intero a 32 bit.
 - Il router destinatario vede SPI, capisce a quale SA corrisponde e quindi decifra ed esegue i controlli di integrità e autenticazione secondo le specifiche della SA.
- **seq#**
 - Numero intero che rappresenta il numero di sequenza del datagram all'interno della SA.
 - Previene l'attacco di “Record and Replay” e verifica l'esistenza di duplicati.
 - Quando una SA viene creata, si parte da Seq# = 0.
 - Seq# è gestito dal mittente.
 - Ogni volta che un datagram viene inviato tramite la SA il mittente incrementa il contatore e piazza il valore nel campo Seq# del datagram IPsec.
 - In questo modo anche il destinatario conosce il valore del numero di sequenza e può aggiornarlo.
- **Padding**
 - Campo di riempimento.
 - Alcuni algoritmi di cifratura lavorano su blocchi di lunghezza fissa, a volte nel dividere a blocchi l'ultimo blocco ha dimensione minore rispetto a quella che dovrebbe avere.

- Serve a far crescere la dimensione dei dati fino a divenire multiplo della dimensione blocco che l'algoritmo in uso riesce a gestire.
- **Pad Length**
 - Lunghezza del campo padding, espressa in ottetti.
- **Next Header**
 - Identifica il protocollo dei dati trasferiti.
- **ESP auth**
 - Contiene i dati necessari per l'autenticazione.

Creazione del Datagram - Tunnel Mode

1. Il Mittente concatena in fondo un **Trailer ESP** al datagram originale.
2. Il Mittente cifra il risultato con l'algoritmo a chiave simmetrica specificato dalla SA.
3. Il Mittente concatena un **Header ESP** di fronte al **Crittogramma**.
4. Il Mittente in base a <**Header ESP** , **Crittogramma** > crea un MAC usando chiave e algoritmo specificati dalla SA.
5. Il Mittente mette il MAC nel campo **ESP Auth**.
 - a. <**Header ESP** , **Crittogramma** , **ESP Auth**>
6. Il Mittente crea un Nuovo Header IP, impostando i parametri per inviare il tutto all'endpoint del tunnel, ossia il router del destinatario.
 - a. Quest'ultimo effettuerà le operazioni inverse e inoltrerà il datagram originale all'host destinatario.

IPsec Security Databases

Ogni router dotato di IPsec dispone di 2 database da usare insieme alle SA.

- **SPD - Security Policy Database**
 - Specifica “COSA deve essere fatto” in una specifica SA.
 - Una “Security Policy” è una regola che stabilisce che tipo di traffico deve essere instradato in una determinata SA e quindi essere coperto da IPsec.
- **SAD - Security Association Database**
 - Specifica “COME deve essere fatto” in una specifica SA.
 - Gli endpoint mantengono lo stato delle SA in un database SAD.
 - Quando si invia un datagram IPsec, il router mittente accede al SAD per determinare come deve processare il datagram per inviarlo in una certa SA.
 - Quando si riceve un datagram, il router destinatario esamina il campo SPI nel datagram IPsec.
 - Usa SPI come indice per accedere a SAD dove troverà le istruzioni.

Domande Sicurezza

MAC (Message Authentication Code) - proprietà del MAC e della funzione hash utilizzata

Il MAC è un valore calcolato basandosi sui messaggi.

Spesso viene calcolato usando funzioni Hash con determinate proprietà:

- One Way → Facile da calcolare ma difficile da invertire.
- Una funzione Hash deve essere deterministica, quindi dato un valore x, H(x) deve restituire sempre lo stesso risultato.
- Deve essere difficile trovare due valori di x e y diversi tali che H(x) = H(y).

Il MAC viene calcolato dal messaggio m come $\rightarrow \text{MAC} = H(<m, \text{info}>)$ dove info dipende da come si vuole implementare.

MAC (Message Authentication Code) - controllo integrità

1. Il mittente invia $<m, \text{MAC}>$, con $\text{MAC} = H(m)$
2. Il destinatario riceve $<m', \text{MAC}'>$
3. Se $H(m') = \text{MAC}' \rightarrow$ Integrità Verificata

MAC (Message Authentication Code) - a cosa serve?

Lo scopo principale del MAC è quello di garantire l'integrità del messaggio.

MAC (Message Authentication Code) - come si implementa?

Il MAC si implementa tramite una funzione Hash che rispetta determinate proprietà.

MAC (Message Authentication Code) - cosa significa che un messaggio è corrotto?

Un messaggio è corrotto se il controllo sul MAC fallisce, ossia:

1. Il mittente invia $<m, \text{MAC}>$, con $\text{MAC} = H(m)$
2. Il destinatario riceve $<m', \text{MAC}'>$
3. Se $H(m') \neq \text{MAC}' \rightarrow$ Messaggio Corrotto.

MAC (Message Authentication Code) - quali altri servizi (oltre integrità) offre?

Nel caso in cui il MAC venga calcolato concatenando insieme al messaggio un segreto comune noto solo ai due interlocutori, a quel punto si ha anche un lieve servizio di autenticazione.

1. Il mittente invia $<m, \text{MAC}>$, con $\text{MAC} = H(<m, s>)$
2. Il destinatario riceve $<m', \text{MAC}'>$
3. Se $H(<m', s>) = \text{MAC}' \rightarrow$ Integrità e Identità del Mittente Verificata

L'identità del mittente è verificata perché (oltre al destinatario) solo lui conosce s .

MAC (Message Authentication Code) - cosa succede quando arriva al destinatario

Il destinatario esegue il controllo sul MAC.

1. Il destinatario riceve $<m', \text{MAC}'>$
2. Se $H(m') = \text{MAC}' \rightarrow$ Integrità Verificata

MAC (Message Authentication Code) - prevenire record and playback

Nel caso in cui il MAC è calcolato solo basandosi su informazioni "statiche" come il messaggio stesso e il segreto comune, allora un utente malevolo potrebbe intercettare uno dei messaggi e inviarlo tempo dopo.

Il messaggio malevolo passerà i controlli di sicurezza del MAC perché il messaggio e il MAC sono coerenti dato che il terzo utente non ha modificato il messaggio.

Per risolvere questo problema basta inserire anche un Nonce o Token di Sessione all'interno del calcolo del MAC.

In questo modo se il terzo utente provasse a inviare il messaggio allo scadere della sessione, il controllo del MAC fallirà.

Questa strategia non funziona se il messaggio replicato viene trasmesso durante la sessione ancora in corso.

Il terzo utente potrebbe anche replicare l'intera sessione.

■ **Firma Digitale - proprietà?**

Un algoritmo di produzione di firma digitale per essere valido deve rispettare 4 proprietà:

1. La firma deve essere Verificabile
 - Il destinatario deve poter provare che una firma appartiene o meno ad un certo utente.
2. La firma non è replicabile
 - La firma non deve poter essere falsificata.
3. Il messaggio firmato non può essere ripudiato dal mittente
 - Un mittente che ha inviato un messaggio firmato non può negare di averlo inviato.
4. Il messaggio non deve poter essere alterabile
 - E' possibile verificare che il messaggio è stato modificato da un intermediario

■ **Firma Digitale - implementazione?**

1. Alice genera $f = K_A^- H(m)$.
2. Alice invia $\langle m, f \rangle$
3. Bob riceve $\langle m', f' \rangle$
4. Bob verifica $H(m') = K_A^+(f')$

■ **Firma Digitale - perché il mac non è definibile una firma digitale?**

Il protocollo MAC soddisfa solo 1 requisito su 4.

Prendiamo in esempio il MAC calcolato su messaggio e segreto condiviso.

Se Alice inviasse a Bob $\langle m, MAC \rangle$.

Bob potrebbe inviare di nuovo la stessa cosa e risulterebbe corretta, questo perché il MAC è calcolato usando il messaggio e un segreto condiviso da entrambi, ergo il MAC non è una cosa che può essere prodotta solo da Alice o solo da Bob.

- Il MAC non è Verificabile
 - Non può provare che un messaggio è stato effettivamente inviato da Bob o da Alice.
- La firma non è replicabile
 - Bob può replicare il MAC prodotto da Alice perché conosce sia il messaggio che il segreto comune.
- Il messaggio firmato non può essere ripudiato dal mittente
 - Dato che il MAC non è una cosa che può essere prodotta solo da qualcuno allora automaticamente il MAC non prova che un certo messaggio è stato effettivamente inviato da qualcuno.

- MAC garantisce l'integrità del messaggio
 - E' possibile verificare che il messaggio è stato modificato da un intermediario

Metodi per inviare una chiave di sessione con riservatezza - con chiave pubblica

1. Alice genera K_{simm}
2. Alice invia a Bob $K_{Bob}^+(K_{simm})$
3. Bob riceve e decifra $K_{simm}^- = K_{Bob}^-(K_{Bob}^+(K_{simm}))$
4. Bob e Alice hanno entrambi la chiave di sessione.

Metodi per inviare una chiave di sessione con riservatezza - KDC

Supponiamo che sia Bob che Alice si siano precedentemente registrati al KDC.

1. Alice informa il KDC che vuole comunicare con Bob.
2. Il KDC invia ad Alice un messaggio cifrato con $K_{Alice-KDC}$ contenente:
 - R1 , ossia la chiave di sessione tra Bob e Alice.
 - $Ticket = K_{Bob-KDC}(R1)$
3. Alice decifra, ottiene < R1 , Ticket > e invia il Ticket a Bob.
4. Bob decifra R1 usando $K_{Bob-KDC}$.

PGP (Pretty Good Privacy) e Confidenzialità dei Messaggi

PGP è una tecnica che permette mantenere la massima confidenzialità durante uno scambio di messaggi.

PGP sfrutta sia i cifrari simmetrici che i cifrari asimmetrici.

Supponiamo che Alice voglia inviare un messaggio confidenziale a Bob.

Supponiamo che le chiavi pubbliche siano state prelevate tramite i certificati.

1. Alice genera una chiave simmetrica R1.
2. Alice genera $M = < m , K_A^-(H(m)) >$
3. Alice genera $C = < C(M, R1), K_{Bob}^+(R1) >$
4. Bob riceve.
5. Bob calcola $R1 = K_{Bob}^-(K_{Bob}^+(R1))$
6. Bob calcola $M' = D(C, R1)$
7. Bob verifica $H(m') = K_A^+(K_A^-(H(m)))$

TLS

Serve a rendere sicure le connessioni TCP.

TLS è un livello compreso tra il livello Applicazione e Trasporto, ha il compito di fornire alle applicazioni del livello superiore delle funzionalità con l'obiettivo di garantire Integrità, Autenticazione e Confidenzialità.

Le applicazioni possono sfruttare tutte, parte o nessuna funzionalità di TLS dipendentemente dai loro bisogni.

I messaggi in TLS vengono chiamati records.

TLS si può dividere in 2 parti:

Record TLS: ha il compito di encapsulare il messaggio in accordo alla cipher suite.

Il record TLS è composto da:

< Tipo , Lunghezza , Payload , MAC >

Il MAC è calcolato usando una funzione Hash applicata a:

- Dati.
- Lunghezza Dati.
- Tipo.
- Numero di Sequenza (Non riportato in chiaro sul messaggio).
 - Previene record and playback e riordino.
- Chiave di Autenticazione (dipende da chi tra i due interlocutori sta componendo il messaggio).

Infine tutto il record viene cifrato usando una chiave simmetrica.

Handshake TLS: Ha il compito di instaurare una sessione TLS tra i due host.

Viene eseguito subito dopo il three way handshake di TCP e ha lo scopo di:

- Far autenticare gli interlocutori mediante l'invio dei propri certificati
- Creazione della Cipher Suite: Il client dice al server quali algoritmi supporta e il server sceglie tra questi.
 - La cipher suite quindi indica gli algoritmi usati per la creazione dei record e per la verifica del MAC.
- Scambio del Master Secret, la base da cui verranno calcolate le chiavi di autenticazione del client e del server e le chiavi per la comunicazione del client e del server mediante gli algoritmi KDF (*Key Derivation Function*).

IPsec - Come si Realizza

IPsec nasce con lo scopo di rendere sicure le comunicazioni a partire dal livello network in su.

IPsec può essere realizzato in 2 modi:

- **Tunnel Mode**
 - Viene cifrato e autenticato tutto il datagram IP (payload e intestazione), per renderlo routable viene aggiunta una nuova intestazione IP.
 - Il datagram viene cifrato dal router del mittente e poi viene decifrato dal router del destinatario, gli host non si accorgono di nulla.
 - Entrambi i router devono implementare IPsec.
- **Transport Mode**
 - Viene cifrato e autenticato solo il payload del datagram IP, viene mantenuta la vecchia intestazione IP.
 - Il datagram viene cifrato dal mittente e poi viene decifrato dal destinatario, i router non si accorgono di nulla.

- Entrambi gli host devono implementare IPsec.

IPsec Tunnel Mode si basa sulle security association.

Le SA sono una connessione stabilita tra i due router (il router del mittente e il router del destinatario) una SA specifica per quali datagram bisogna usare IPsec e per essi come bisogna operare.

Queste informazioni sono contenute in 2 database:

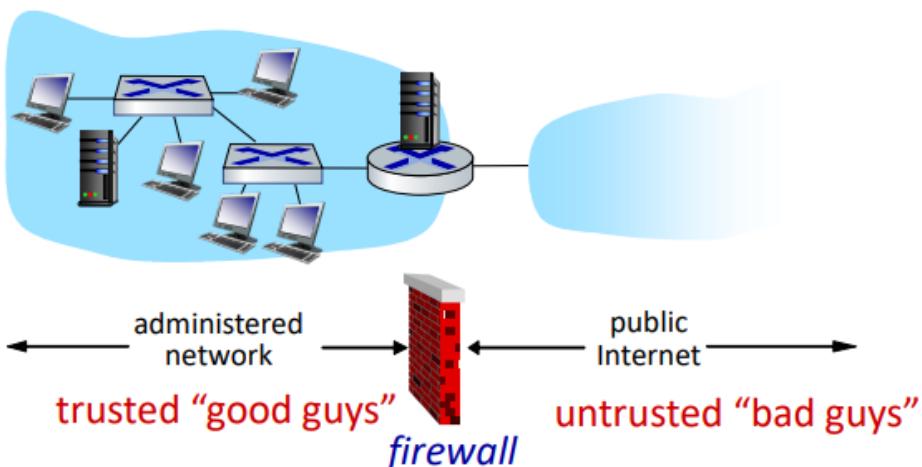
- SAD: il quale indica come deve essere processato il datagram (quale algoritmo usare etc.)
- SPD: il quale indica per ogni SA quali datagram devono passare per quella SA.

IPsec - cosa sono le Security association

Le SA consistono in una relazione tra due router che implementano IPsec.

Una SA esprime come devono essere gestiti i datagram di IPsec, ovvero come devono essere cifrati/decifrati, come è stato calcolato il MAC e come deve essere fatto il controllo su di esso.

Firewall



Il firewall è un sistema Hardware o Software che controlla le connessioni in ingresso e in uscita e applica delle regole.

Isola una rete privata da molte minacce dalla rete globale.

Un'organizzazione ha una rete interna, costituita da tanti computer e magari sottoreti interni, ovviamente abbiamo anche almeno un router di frontiera.

Io non vorrei che i dipendenti eseguissero accessi a siti non raccomandabili o comunque che non dovrebbero visitare durante le ore di lavoro (tipo i dipendenti comunali che stanno ore a guardare la Gazzetta dello Sport durante la prima ora di lavoro), ossia voglio un controllo sugli accessi eseguiti con la rete dell'organizzazione.

Ma non posso negare tutti gli accessi, la mia rete non deve essere completamente isolata. Come le mura di Lucca, ho bisogno di Porte, dove verrà eseguito un controllo degli accessi e delle uscite da parte delle Guardie.

- Città → Rete dell'Organizzazione.
- Persone → Pacchetti.

- Porta → *Router di Frontieria*.
- Guardie → *Firewall*.

Il Firewall impostato in modo opportuno può:

- Prevenire Attacchi DoS (*Denial of Service*).
- Evitare Accessi Non Autorizzati e consentire solo Accessi Autorizzati.

Attacco DoS di tipo “SYN Flooding”

Uno o più client hanno lo scopo di inviare milioni di richieste di connessioni TCP al server bersaglio con lo scopo di far collassare il sistema.

Ad ogni richiesta di connessione TCP il server alloca le risorse necessarie alla connessione, le strutture dati per mantenere le connessioni TCP vengono allocate in RAM, quindi se le richieste sono milioni la memoria si riempie e il server collassa.

Il Server inoltre, per ogni richiesta di connessione, manda un “SYN ACK” e aspetta la risposta del client malevolo il quale ovviamente non risponderà mai, facendo perdere tempo al server.

Tipologie di Firewall

Ci sono 2 tipologie di firewall:

- **Packet Filters**
 - I campi di ogni pacchetto in entrata e in uscita vengono ispezionati.
 - Ogni pacchetto viene analizzato esclusivamente in base ai campi dell'intestazione TCP/IP.
 - Il pacchetto passa (*allowed*) solo se i campi verificano certe condizioni altrimenti viene scartato (*dropped*).
 - Questa tipologia si divide in 2 sottocategorie:
 - **Stateless Packet Filters**
 - Non tiene conto delle connessioni TCP attive.
 - Meno preciso ma più veloce.
 - **Stateful Packet Filters**
 - Tiene conto delle connessioni TCP attive.
 - Più preciso ma più lento.
 - Tiene traccia delle connessioni TCP e degli scambi UDP in corso, quindi è in grado di discriminare le connessioni legittime da quelle sospette.
- **Application Gateways**
 - Operano fino a livello applicazione facendo una “*deep packet inspection*”.
 - Può vedere tutto il traffico che proviene da un host, quindi può anche ricostruire una pagina web oppure ricostruire una conversazione email.
 - Più efficaci dei Packet Filters, ma usano maggiori risorse computazionali.
 - Efficace contro malware, vulnerabilità note, comportamenti dannosi delle applicazioni, ecc.

Funzionamento generale di un Firewall

Il firewall consiste in una tabella di regole.

Ogni regola contiene caratteristiche di pacchetto (*criteria*) e azioni da intraprendere (*target*) dove scarta (*drop*) o accetta (*accept*).

Per ogni pacchetto in entrata o in uscita scorre le regole (in ordine crescente di indice).

Quando trova una regola che fa match con il pacchetto, quest'ultimo viene fatto passare/droppare e la scansione delle regole viene arrestata.

Le regole devono essere ordinate in modo opportuno , dalla più specifica alla meno specifica.

L'ultima regola della tabella è detta "Regola di Default".

Regola di Default

A seconda della regola di default (l'ultima regola della tabella) il firewall può essere:

- **Inclusivo**

- L'ultima regola blocca tutto.
- Le regole non di default definiscono ciò che può passare.
- Più comodi e sicuri da definire perché è molto più facile definire ciò che può passare invece che pensare ad ogni singolo caso in cui un pacchetto non può passare.

- **Esclusivo**

- L'ultima regola include tutto.
- Le regole non di default definiscono ciò che non può passare.
- E' difficile prevedere tutti i possibili casi dannosi.

Firewall - Stateless Packet Filter

Il firewall di questo tipo è implementato (tipicamente) sul router di frontiera.

Fa il suo lavoro semplicemente ispezionando i campi statici del singolo datagram, senza considerare eventuali relazioni con altri datagram passati precedentemente.

I campi dei pacchetti sia in entrata che in uscita vengono ispezionati.

Se i campi di un pacchetto verificano certe condizioni viene fatto passare altrimenti viene scartato o viceversa.

Esempi di Policy

Policy	Firewall Setting
no outside Web access	drop all outgoing packets to any IP address, port 80
no incoming TCP connections, except those for institution's public Web server only.	drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
prevent Web-radios from eating up the available bandwidth.	drop all incoming UDP packets - except DNS and router broadcasts.
prevent your network from being used for a smurf DoS attack	drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255)
prevent your network from being tracerouted	drop all outgoing ICMP TTL expired traffic

ACL - Access Control List

Il firewall ha bisogno di uno spazio di memoria in cui mettere le policy e in cui può consultarle.

Le policy sono inserite in una tabella situata in memoria chiamata **Access Control List**.

La struttura delle ACL è molto simile alla tabella di OpenFlow, infatti un Router di frontiera dotato di OpenFlow può tranquillamente sostituire un Firewall Stateless Packet Filter.

Al momento dell'arrivo di un pacchetto al router, il firewall accede alla ACL ed esegue una scansione dalla prima riga fino all'ultima, per ogni riga controllerà la presenza del match.

Al primo match si interrompe la scansione e si esegue l'azione specificata.

Action	Source Address	Destination Address	Protocol	Source Port	Destination 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 (ack bit settato)
Allow	222.22 / 16	Outside of 222.22 / 16	UDP	> 1023	53	---
Deny	all	all	all	all	all	all

In ogni ACL è bene definire regole che facciano passare solo pacchetti che hanno senso, ad esempio un pacchetto UDP destinato ad un servizio che notoriamente usa solo TCP non ha senso e quindi può essere malevolo.

Firewall - Stateful Packet Filter

Il firewall tiene traccia dello stato di ogni connessione TCP attiva al quel momento.

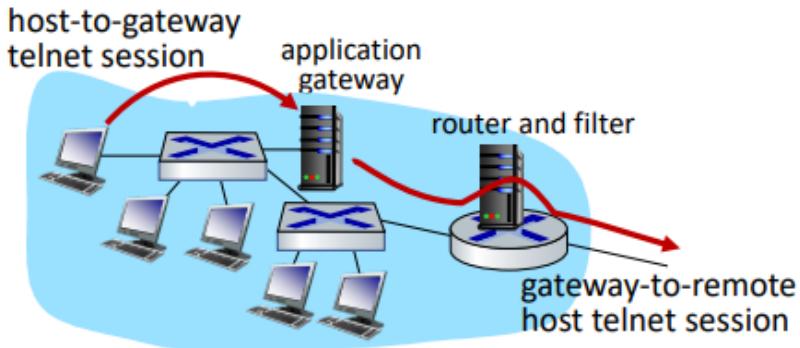
Può farlo perché vede tutto il traffico in entrata e in uscita e semplicemente ispezionando le intestazioni degli header dei protocolli di livello trasporto, quindi osservando tutti i pacchetti in transito prende nota dei pacchetti di *Connection Setup* (SYN) e *Teardown* (FIN). Quindi al contrario dello stateless, questo tipo di firewall considera anche eventuali pacchetti passati prima di quello in esame in questo momento.

Action	Source Address	Destination Address	Protocol	Source Port	Destination 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	---	X
Deny	all	all	all	all	all	all	

La X significa che per far passare quel pacchetto si necessita una connessione TCP attiva.

Firewall - Application Gateway



Un firewall application gateway applica una “deep packet inspection”, quindi il filtro dei pacchetti avviene anche sul contenuto del payload del livello applicazione, oltre che sui campi IP e TCP.

E' in grado di distinguere il traffico di un'applicazione indipendentemente dalla porta di comunicazione che questa utilizza.

Questo tipo di firewall sono dispositivi che spesso sono installati insieme a firewall packet filter, questo perché un firewall application gateway effettua controlli abbastanza onerosi, quindi è bene lasciare i controlli “banali” a firewall meno complessi e più veloci.

In genere questo tipo di Firewall ha lo scopo di proteggere un Server Applicativo.

E' usato quando nella nostra rete abbiamo un Server accessibile dall'interno o dall'esterno.

Fa da intermediario tra la Rete Internet (e la rete interna) e il Server Applicativo: ogni richiesta viene controllata in modo da prevenire il passaggio di contenuti malevoli.

Grazie all'application Gateway al Server Applicativo arriveranno solo richieste “sicure”.

Dopo che la richiesta ha passato i controlli, il firewall application gateway inoltra la richiesta al Server Applicativo.

Il Server Applicativo e tutti i Client sono ignari della presenza dell'Application Gateway.

Con questo tipo di firewall posso decidere se certi Utenti Interni o Esterni possono usare certe applicazioni.

Firewall - Application Gateway - Intermediario nelle Connessioni

Un'altra caratteristica che lo distingue da un packet filter firewall è la capacità di spezzare la connessione tra un host della rete interna e un host della rete esterna.

Ad esempio: voglio permettere agli utenti della rete interna di usare telnet verso l'esterno.

- Richiedo che ogni utente che usa il protocollo TELNET passi dal gateway.
- Per gli host autorizzati a usare telnet, il gateway imposta la connessione telnet con il destinatario e si pone come intermediario tra i due.
- Il Gateway blocca tutte le connessioni telnet che non provengono dal gateway.

Il software del client deve sapere l'indirizzo IP del' Application Gateway.

Firewall - Application Gateway - Intrusioni

Questo tipo di firewall è in grado di rilevare i tentativi di intrusione attraverso lo sfruttamento di un protocollo e di realizzare le funzionalità di logging e reporting.

Firewall - Application Gateway - Contro

- Un firewall application gateway è specifico per ogni applicazione.
 - Se ho 4 applicazioni, in genere dovrò mettere su 4 firewall di questo tipo.
- Costituisce un collo di bottiglia per le performance della rete.
 - I controlli eseguiti sono onerosi e devono essere fatti su ogni richiesta diretta a quel server.
- A causa dell'*IP spoofing* un router non può mai essere certo della vera provenienza dei messaggi.
- La sicurezza offerta non è comunque perfetta.

Firewall - Next Generation Firewalls (Extra - No Lezione)

Piattaforma che riunisce in un unico dispositivo diverse tecnologie per la sicurezza.

Fra queste ci sono le tecnologie di filtraggio dei firewall presentati in precedenza ovvero:

- Firewall Stateless Packet Filtering.
- Firewall Stateful Packet Filtering.
- Application Gateways.
- NAT.
- Supporto alle VPN.

Alcune delle altre caratteristiche tipiche di un next-generation firewall sono:

- Rilevamento e la prevenzione delle intrusioni
 - Sistemi IDS.
 - Sistemi IPS.
- La definizione di policy specifiche per ogni applicazione.
- Integrazione dell'identità dell'utente.
- Acquisizione di dati di supporto per la sicurezza da fonti esterne.
- La qualità di servizio.

L'obiettivo di questa tecnologia di firewall è la semplificazione di configurazione e gestione di un insieme eterogeneo di strumenti di sicurezza e allo stesso tempo il miglioramento del loro impatto sulle performance dell'intero sistema.

WAF - Web Application Firewall (Extra - No Lezione)

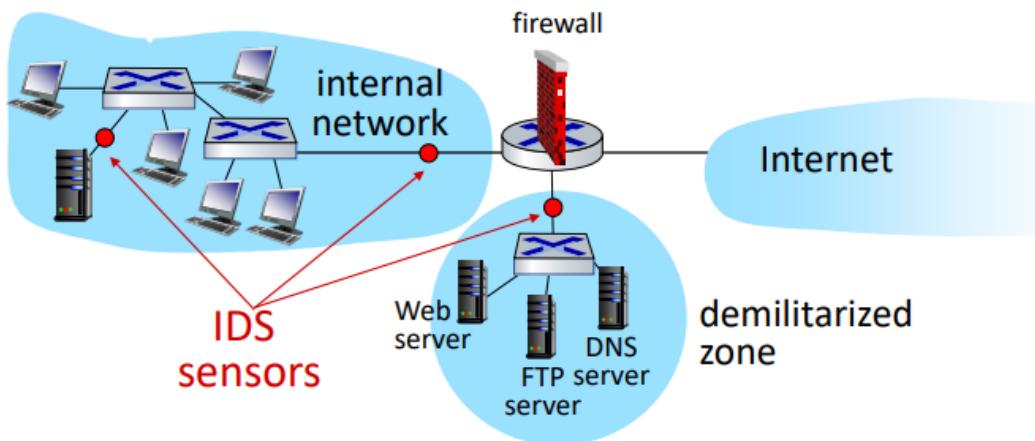
Specifico forma di firewall software che filtra, monitora e blocca traffico HTTP in entrata e uscita da un servizio web.

Ispezionando il traffico HTTP, una soluzione WAF può prevenire attacchi provenienti da minacce veicolate attraverso il web.

IDS - Intrusion Detection System

Dispositivi a supporto dei firewall, effettuano una *deep packet inspection*, quindi possono vedere tutto il pacchetto, anche il contenuto del pacchetto di livello applicazione.

Gli IDS devono essere numerosi e sparsi in modo capillare su tutta la rete interna, in modo da rendere più approfondita possibile l'ispezione, poiché un solo IDS non può stare dietro a tutti i pacchetti a causa del costo computazionale della *deep packet inspection* e non li vedrebbe nemmeno tutti.



Effettuano controlli su molteplici pacchetti, eseguendo anche correlazioni tra pacchetti diversi, ma non tra diverse sessioni.

Il loro scopo è la ricerca di stringhe sospette, questa cosa la fanno con un database di supporto con tutti i virus noti.

Sono in grado di verificare se in questo momento ci sono attacchi in corso.

Un IDS può rilevare vari tipi di attacchi eseguendo la correlazione tra pacchetti diversi:

- Port Scanning.
- Network Mapping.
- DoS Attack.

IPS - Intrusion Detection System (Extra - No Lezione)

In informatica un sistema di prevenzione delle intrusioni o intrusion prevention system (IPS) sono dei componenti software attivi sviluppati per incrementare la sicurezza informatica di un sistema informatico, individuando, registrando le informazioni relative e tentando di segnalare e bloccare le attività dannose.

Sono un'estensione degli IDS , perché entrambi controllano il traffico e analizzano le attività di sistema per identificare l'esecuzione di codice non previsto, ma a differenza di quest'ultimi, gli IPS sono posizionati inline e sono abilitati a prevenire e bloccare le intrusioni identificate.

Più specificamente, IPS può:

- Mandare un allarme.

- Eliminare pacchetti malevoli.
- Resetare le connessioni e/o bloccare il traffico da un indirizzo IP attaccante.
- Correggere gli errori CRC.
- Deframmentare pacchetti.
- Evitare problemi di sequenza TCP.
- Ripulire i livelli di trasporto e rete da opzioni indesiderate.

Collegamenti Wireless

Access Point

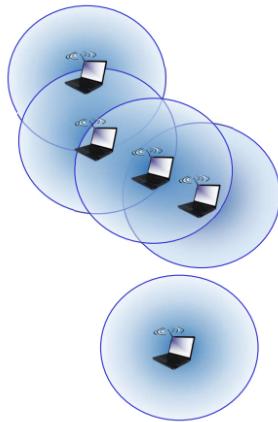
Chiamato anche “*Base Station*”.

Tipicamente connesso ad una rete cablata, permette l’accesso alla rete agli host wireless.

Essi ricevono pacchetti dagli host wireless e li inoltrano nella rete cablata, i quali si comporteranno come dei normali pacchetti.

Ogni access point ha una “*Area di Copertura*” (chiamata anche “*Hotspot*” o “*Cella*”), oltre al quale smette di funzionare.

Modalità di Reti Wireless “ad hoc”



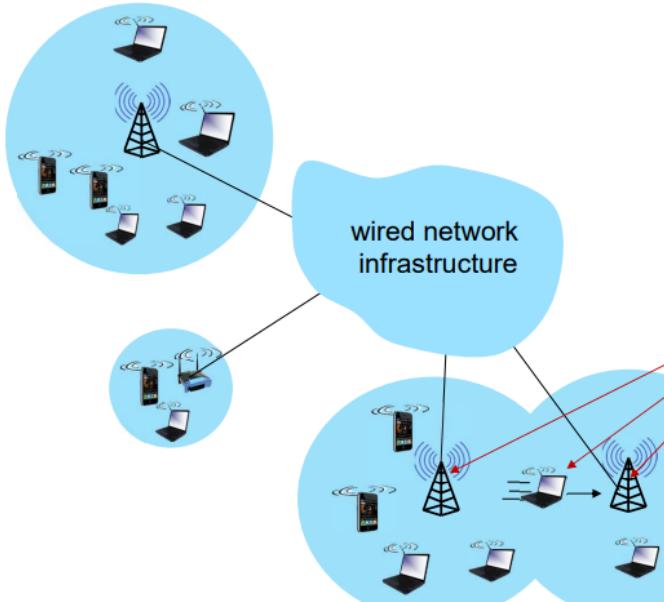
Non ci sono stazioni base.

Un nodo può trasmettere solo ai nodi che rientrano nel suo raggio di copertura.

La rete è organizzata dai nodi stessi e sono i nodi ad effettuare il routing dei pacchetti.

Il vantaggio principale di questa rete è la decentralizzazione assoluta.

Modalità di Reti wireless ad infrastruttura



Sono presenti gli access point, le quali fanno da ponte tra la rete wireless e quella cablata. Per connettersi alla rete gli host wireless comunicano con un Access Point il quale riceverà i segnali wireless, decodificherà i segnali e infine immetterà i pacchetti all'interno della rete cablata.

A quel punto circoleranno come i pacchetti classici visti fin'ora.

Tassonomia delle Architetture delle Reti Wireless

	Single hop	Multiple hops
Infrastructure-based	Host connects to base station which connects to larger Internet: <i>WiFi, cellular networks</i>	Host may have to relay through several wireless nodes to connect to larger Internet: <i>sensor networks</i>
Ad hoc	No base station, no connection to larger Internet: <i>Bluetooth</i>	No base station, no connection to larger Internet. May have to relay to reach other a given wireless node: <i>MANET, VANET</i>

Wireless Link

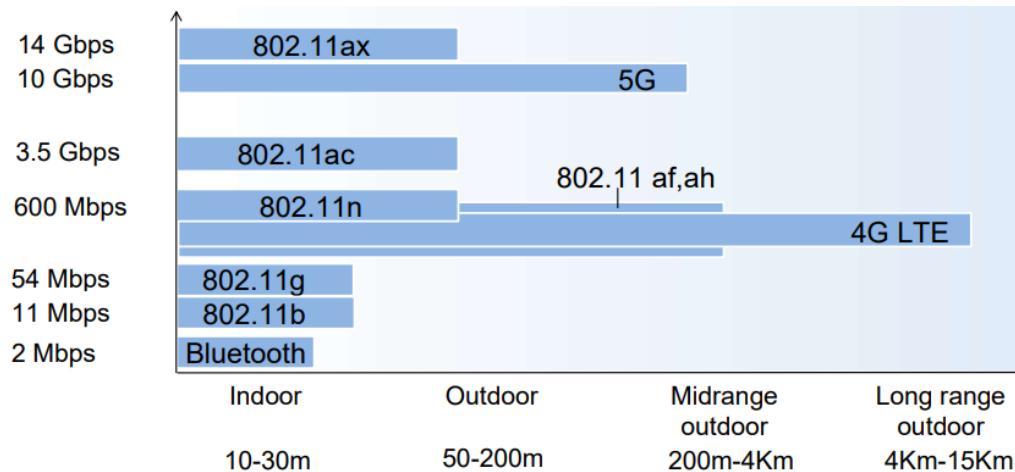
Ogni tipologia di collegamento wireless lavora ad una certa frequenza e un certo raggio di copertura.

Ci sono importanti differenze con un collegamento cablato:

- La potenza del segnale degrada in maniera esponenziale (x^2) con la distanza (*path loss*).
- Il segnale può rimbalzare contro gli oggetti con la possibilità di creare ritardi.
- Il segnale ha una certa frequenza, ci possono essere altri segnali con la stessa frequenza che potrebbero causare interferenze e far diminuire la qualità del servizio.

- Il Bit Error Rate di un link wireless in genere è sempre superiore rispetto ad un link cablato.

Tassonomia link wireless



SNR - Signal To Noise Ratio

SNR è anche detto "rapporto segnale-rumore".

Mette in relazione la potenza del segnale utile (quello che vogliamo trasmettere o ricevere) rispetto a quella del rumore (ossia la potenza delle interferenze).

$$SNR = \frac{Pot_{segnale}}{Pot_{rumore}}$$

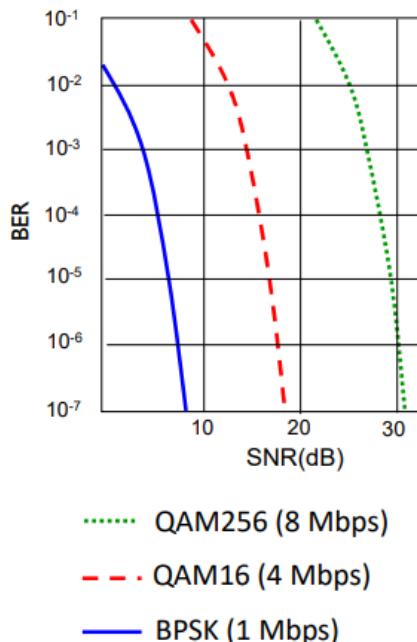
Più SNR è alto, migliore sarà la ricezione del segnale e quindi più basso sarà il Bit Error Rate (BER).

Supponendo che la potenza del rumore sia fissa e che l'oggetto sia immobile.

Fissato un valore di SNR: Il BER dipende dal tipo di codifica (*physical layer*) utilizzato.

Fissata una certa Codifica: Aumentando la Potenza del Segnale → SNR Aumenta → BER Diminuisce.

Il valore di SNR può variare anche in funzione della posizione del nodo, poiché più è grande la distanza tra ricevitore e trasmettitore, e più sarà grande la diminuzione della potenza del segnale utile che riceverà il ricevitore.



Il Problema del Nodo Esposto

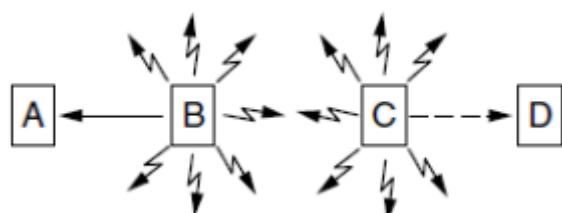
Supponiamo di avere 4 nodi: A, B, C e D.

- A vorrebbe trasmettere qualcosa a B.
- C vorrebbe trasmettere qualcosa a D.

Tutti i nodi possono vedere le trasmissioni di tutti.

C sente le trasmissioni di A, conclude che il canale è occupato e quindi non trasmette verso D.

Ciò introduce del ritardo alla trasmissione di C.



Il Problema del Nodo Nascosto

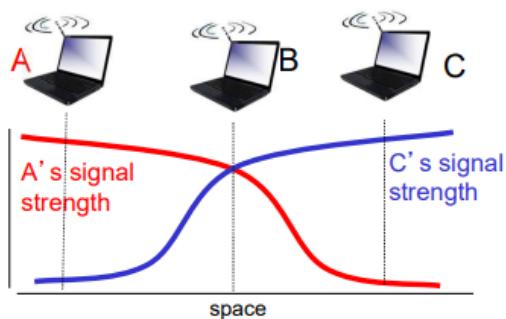
Supponiamo di avere 3 nodi: A, B e C.

- A e C vorrebbero trasmettere qualcosa a B.
- C è “nascosto” rispetto ad A.

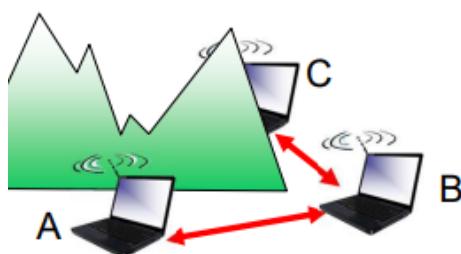
Per nascosto si intende che A non riesce a rilevare C e viceversa, ciò può accadere per vari motivi, ad esempio perché:

C è fuori dall'area di copertura di A, quindi il segnale emesso da A una volta arrivato a C ha una potenza talmente bassa da essere ignorato.

Come si vede dalla figura sottostante, la decadenza del segnale utile in funzione della distanza non è lineare ma esponenziale.



C'è un ostacolo (*fisico*) tra A e C, tale per cui il segnale emesso da A una volta arrivato a C ha una potenza talmente bassa da essere ignorato.



Quindi per C, dato che non vede la trasmissione del nodo A il link wireless risulta libero e quindi inizia a trasmettere, inevitabilmente si verifica una collisione al nodo B tra il segnale del nodo A e il segnale del nodo C.

Il nodo B è l'unico che si accorge della collisione.

Questa è una grandissima differenza con le reti Ethernet cablate.

La collisione viene avvertita solo dal destinatario, mentre nelle reti cablate è avvertita (anche se in tempi diversi) da tutti i nodi connessi al link di accesso comune.

Wireless LAN - Wi-Fi

Wi-Fi (*Wireless Fidelity*) è il nome commerciale dello standard chiamato: 802.11, definito nel 1999.

Esso definisce vari protocolli per la creazione di LAN Wireless.

802.11 può supportare sia la modalità infrastruttura che la modalità ad hoc.

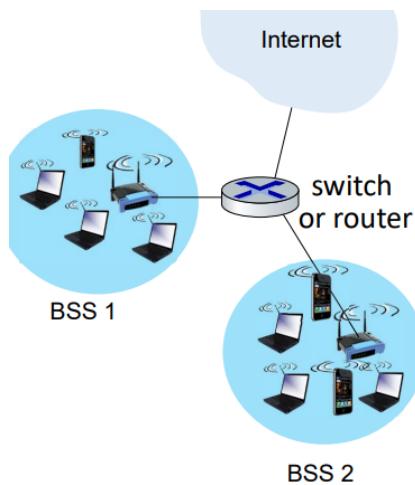
L'ultima versione è la 802.11ax (Wi-Fi 6.0) definita nel 2020.

Tutte le versioni di Wi-Fi usano CSMA/CA per regolare gli accessi multipli al canale.

BSS - Basic Service Set

Un Access Point e i dispositivi collegati a quell' Access Point definiscono un “*Basic Service Set*”.

In una versione ad hoc un BSS è composto solo dai dispositivi.



802.11 - Associazione tra Host e Access Point

Lo spettro delle frequenze è diviso in molteplici canali, ognuno con la propria frequenza. Sarà l'Admin a decidere la frequenza in cui lavorerà l'access point, le interferenze sono possibili.

L'access point manda ogni 100 ms un “*Beacon Frame*”, per segnalare la propria presenza agli host nelle vicinanze.

Il Beacon Frame contiene:

- Il SSID (*Service Set Identifier*).
 - Che identifica l'access point (in realtà identifica il BSS)
- Indirizzo MAC dell'access point.

Un host può ricevere i beacons di molteplici Access Point contemporaneamente, ma può associarsi ad uno solo di essi.

L'associazione può richiedere anche una autenticazione (tipicamente una password o un login).

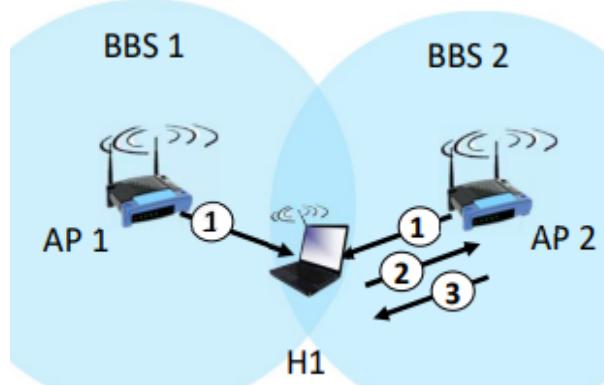
Dopo l'associazione, in genere, l'host appena connesso esegue una richiesta DHCP per ottenere una configurazione valida per navigare in rete.

802.11 - Scanning

Lo scanning è un'operazione fatta da un Host wireless con lo scopo di cercare un access point con cui collegarsi.

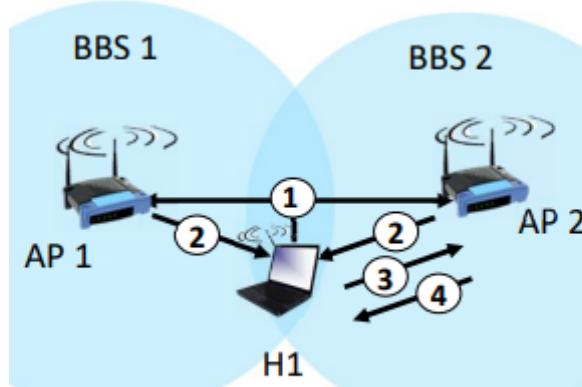
Scanning Passivo (Access Point fa la prima mossa)

1. L'host si sintonizza su una frequenza e attende i beacon frame dagli access point sintonizzati su tale frequenza.
 - a. Se non ottiene nulla cambia frequenza e riprova.
2. L'host riceve i beacon frame, sceglie l'access point, invia un frame di richiesta di associazione (specificando l'access point scelto) e attende il frame di risposta di associazione.
3. L'access point invia un frame di risposta di associazione.



Scanning Attivo (Host fa la prima mossa)

1. L'host invia a tutti i dispositivi wireless sintonizzati ad una certa frequenza un frame speciale chiamato "Probe Request Scanning".
 - o "Probe" → "Sonda"
 - o Serve ad avvertire gli Access Point che c'è un dispositivo che ha bisogno di connettersi.
2. Gli access point in ascolto di risposta inviano il Beacon Frame.
3. L'host riceve i beacon frame, sceglie l'access point, invia un frame di richiesta di associazione (specificando l'access point scelto) e attende il frame di risposta di associazione.
4. L'access point invia un frame di risposta di associazione.



802.11 - Regolazione degli Accessi Multipli

Il mezzo wireless è condiviso da molteplici nodi wireless.

Quindi c'è di nuovo il problema dell'accesso multiplo, già affrontato con le reti cablate.

Non posso usare CSMA/CD perché prevede che siano soddisfatte 2 condizioni:

- Il trasmettitore ascolti mentre trasmette per rilevare una collisione, ma se l'antenna viene usata per ricevere non posso usarla per trasmetterla.
 - o Anche mettendo 2 o più antenne non risolverà il problema per il punto successivo.
- Tutti i trasmettitori sul canale sono in grado di rilevare una collisione in corso, ma come abbiamo visto prima abbiamo il Problema del Nodo Nascosto
 - o La collisione quando si verifica, viene rilevata sempre nel Nodo Destinatario ma i trasmettitori potrebbero non accorgersi di nulla.

In una rete ethernet cablata tutti i nodi prima o poi ricevono lo stesso segnale, perché esso si propaga su tutto il mezzo condiviso.

Diversamente dalle reti cablate, data una trasmissione non è detto che tutti i nodi del BSS rilevano quella trasmissione.

NOTA - ACK

Se per caso hai le idee confuse sugli ACK leggi qua.

L'ACK a livello Data Link, se è presente (e molti protocolli a livello di collegamento non ce l'hanno affatto) ti dice solo che un determinato frame a livello di collegamento ha attraversato con successo (senza errori) UN SOLO HOP.

L'ACK del TCP, è di tipo cumulativo e ha il numero di sequenza dei byte PAYLOAD che ha raggiunto la destinazione, attraverso tutti i link-hop necessari e possibilmente (in caso di lunga distanza, comunemente) su molti TIPI di hop a livello di collegamento.

Lungo il percorso, il payload potrebbe essere stato scomposto e collocato in diversi numeri e dimensioni di frame a livello Data Link, non è scontato che tutti questi livelli data link posseggono un meccanismo di ACK.

Queste informazioni orientate al PAYLOAD consentono al TCP di ignorare qualsiasi cosa accada a livelli inferiori e funzioni come previsto.

CSMA/CA

CA → “Collision Avoidance”.

Il mio approccio è “*Evitare le Collisioni*”, perché in una rete ethernet cablata i nodi trasmettitori avevano modo di capire quando stava avvenendo una collisione, infatti era “*Collision Detection*”.

Invece in una rete wireless questa ipotesi non è più corretta a causa del problema del nodo nascosto.

Trasmettitore

1. Ascolta il canale.
2. Se il canale rimane libero (*idle*) per un tempo pari a **DIFS** (*Distributed Inter Frame Space*), allora il canale è libero.
 - DIFS è un parametro definito nello standard.
3. Si aspetta un tempo randomico (*Backoff Time*).
 - Di solito: Backoff Time > DIFS.
4. Se mentre si aspetta lo scadere del Backoff Time il canale non è più in *idle* (e quindi risulta occupato) si ritorna all'ascolto del canale (quindi al punto 1).
5. Quando il *Backoff Time* scade e il canale è rimasto sempre libero si trasmette il frame per intero.

Ricevitore

1. Riceve il frame.
2. Decodifica il frame ed esegue i controlli di correttezza.
 - Se il frame risulta corrotto → Si scarta e non si invia ACK.

3. Aspetterà un SIFS (Short Inter Frame Space)
 - o $SIFS < DIFS \rightarrow$ Dare priorità alle risposte dei ricevitori rispetto ad altre trasmissioni.
4. Invia ACK.

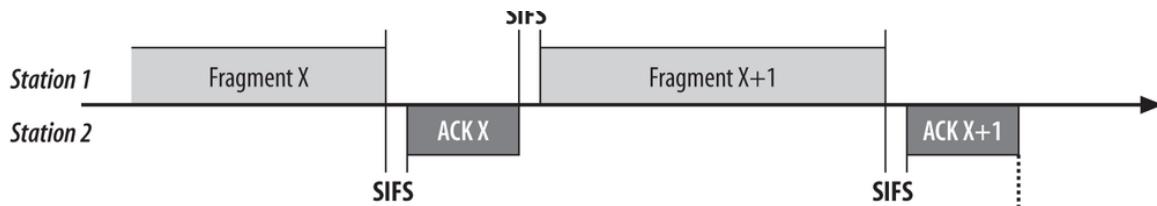
Trasmettitore

1. Se ricevo ACK \rightarrow **Tutto OK**.
2. **Se non ricevo ACK entro un determinato periodo di tempo**, allora potrebbe:
 - o Essersi verificata una collisione, il ricevitore ha ricevuto troppi segnali contemporaneamente e il segnale ricevuto è stato decodificato in modo scorretto.
 - o Il ricevitore ha ricevuto e decodificato correttamente, ma c'erano errori nel pacchetto.
 - o **In entrambi i casi assumiamo che si sia verificata una collisione (scelta conservativa)** quindi aspetto un certo intervallo di tempo (*Backoff Interval*).
3. Ripeto la trasmissione (*backoff*).

Se non fosse chiaro

In che modo il ricevitore avverte che c'è stata una collisione? Semplicemente evitando di inviare ACK.

Se il trasmettitore non riceve ACK entro un determinato quantitativo di tempo \rightarrow Collisione Rilevata.



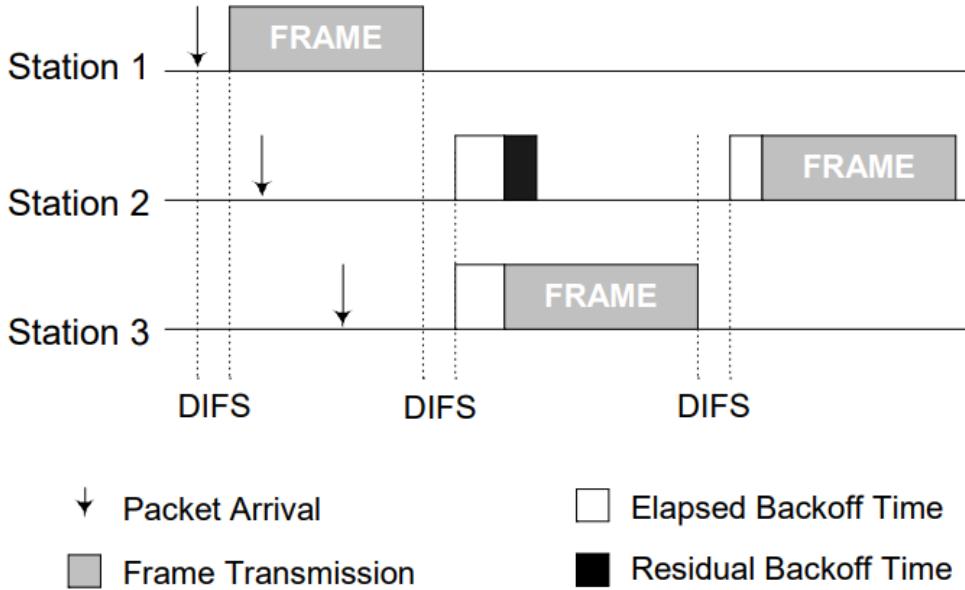
Backoff Time

Se il canale è occupato, ma poi per un periodo di tempo pari a DIFS non viene trasmesso nulla allora i nodi in ascolto desiderosi di trasmettere penseranno che sia libero, ma in questo caso la probabilità di collisioni è molto alta, perché a quel punto tutti i nodi trasmetterebbero contemporaneamente.

Il Backoff Time serve a desincronizzare i trasmettitori in attesa che il canale sia libero.

Quindi per evitare questa corsa alla trasmissione che porterà sicuramente ad una collisione si esegue una procedura chiamata "*Backoff Algorithm*" , usata per calcolare il *Backoff Time*.

Noi eseguiamo un Backoff "Preventivo", che a differenza di ethernet per le reti cablate (dove viene fatto a seguito di una collisione) viene fatto prima di trasmettere.



Backoff Algorithm

Algoritmo eseguito da ogni nodo trasmettitore per calcolare il Backoff Time.

Il tempo viene diviso in intervalli di lunghezza fissa detti “*Backoff Slots*”.

- **Backoff Time**

- Intervallo di tempo casuale compreso tra $[0, CW - 1]$
- Viene calcolato prima di trasmettere e il trasmettitore può trasmettere solo dopo lo scadere di questo intervallo.
- Mentre il timer è attivo si ascolta il canale: se il canale viene usato allora il timer viene fermato e si ritorna alla fase di ascolto del canale.

- **CW - Contention Window**

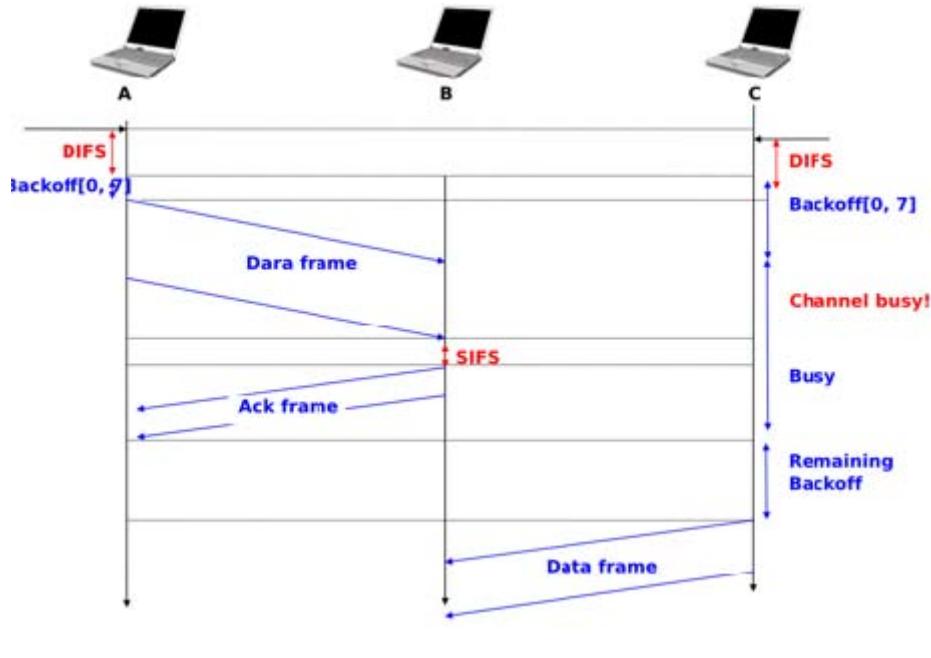
- Inizialmente $\rightarrow CW = CW_{min}$
- Ha un limite superiore $\rightarrow CW \leq CW_{max}$
- Se non riceve ACK $\rightarrow CW_{i+1} = 2 * CW_i$
 - CW_{max} e CW_{min} dipendono dalla codifica usata.

Con questo approccio la probabilità di collisione non si azzerà, le collisioni potrebbero ancora verificarsi:

Quando due stazioni generano lo stesso *Backoff Interval* e quindi trasmettono contemporaneamente allo scadere di esso.

A ogni sintomo di collisione faccio in modo che la probabilità che avvenga una collisione diminuisca sempre di più, poiché la probabilità che accada è inversamente proporzionale al valore di CW.

Ho sempre il Problema del Nodo Nascondo.



Virtual Carrier Sensing

Devo modificare il protocollo per attenuare il Problema del Nodo Nascosto.

Il mio obiettivo è preventivamente informare tutti i nodi di un BSS che sta avvenendo una trasmissione tra due nodi, in modo tale che gli altri nodi non trasmettono nulla nel mentre.

Supponiamo che il nodo A e il nodo C non riescano a vedersi.

Supponiamo che C ed A vogliano trasmettere qualcosa a B.

B ovviamente è nella portata sia di A che di C.

Nota: il nodo B di solito è l'access point, il quale è il ricevitore della maggior parte delle trasmissioni in una rete wireless.

1. A e C trasmettono entrambi un pacchetto RTS (*Request-To-Send*) in broadcast.
 - I pacchetti RTS vengono sentiti da tutti i nodi nella portata del mittente.
 - RTS(A) → “Il nodo A chiede il permesso a B di Trasmettere”.
 - RTS(C) → “Il nodo C chiede il permesso a B di Trasmettere”.
2. B riceve i pacchetti RTS.
 - RTS(A) e RTS(C) possono collidere. (*Vedi Dopo*)
3. B sceglie uno dei trasmettitori che ha inviato un pacchetto RTS e risponde (sempre in broadcast) con un pacchetto di tipo CTS (*Clear-To-Send*).
 - CTS(A) → “Il nodo A è autorizzato a Trasmettere a B”
4. Il nodo C vede CTS(A) e non trasmette perché in questo momento solo A è autorizzato a trasmettere a B.
 - C setta un timer con il valore di NAV, parametro contenuto in CTS(A) nel campo duration.
5. Il Nodo A trasmette, il nodo C nonostante veda il canale libero non trasmette perché il NAV non è ancora scaduto.
 - B riceve DATA(A).
6. Il Nodo B invia in broadcast (quindi il nodo C vede questo messaggio) l'ACK verso A.
 - ACK(A) → “Il nodo B ha ricevuto tutto da A ed è pronto a ricevere altre trasmissioni”

7. Il Nodo C invia RTS(C) e spera che stavolta sia il suo turno.

CTS e RTS sono 2 tipologie di frame con dimensione molto piccola (poiché non hanno dati) ma contengono dei campi particolari.

Virtual Carrier Sensing - Collisioni negli RTS

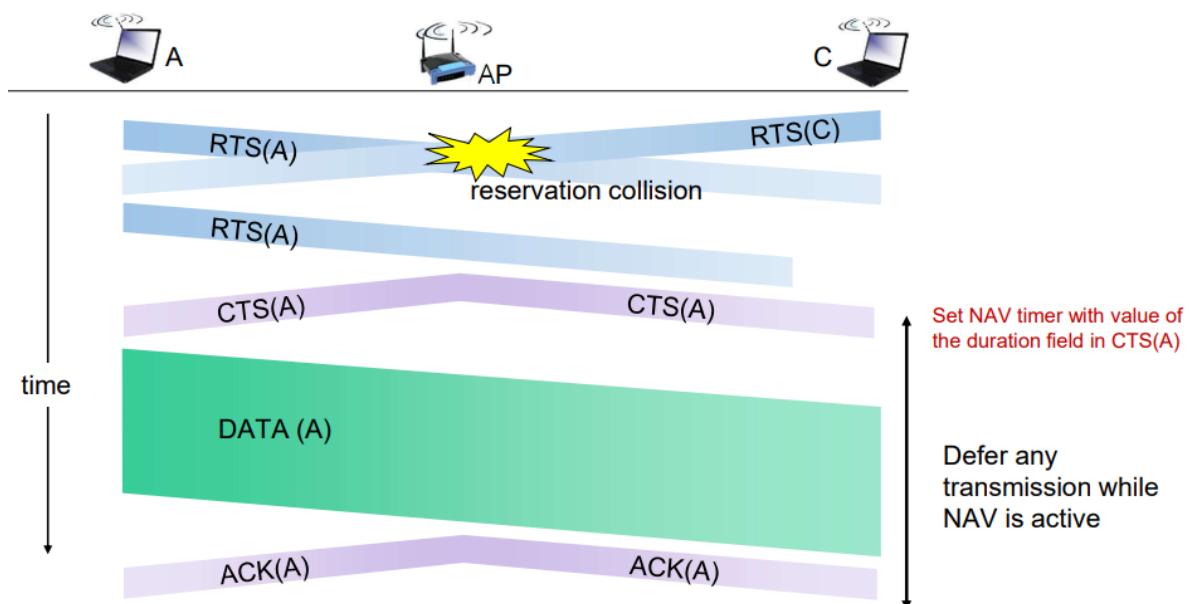
Cosa succede se sia A che C trasmettono RTS nello stesso istante?

RTS è trasmesso "senza protezione", e quindi RTS(A) e RTS(C) potrebbero collidere.

Ma comunque abbiamo un notevole guadagno:

La probabilità di collisione diminuisce, perché RTS è un molto più piccolo rispetto al frame che contiene i dati, quindi la probabilità di collisione viene ridotta molto perché la quantità di dati che viene trasmessa "senza protezione" è molto più piccola e quindi il tempo di trasmissione durerà molto meno.

Inoltre dato che RTS è molto piccolo, trasmetterlo di nuovo non costa molto.



Formato del Pacchetto del protocollo 802.11 Wi-Fi

2	2	6	6	6	2	6	0 - 2312	4
frame control	duration	address 1	address 2	address 3	seq control	address 4	payload	CRC

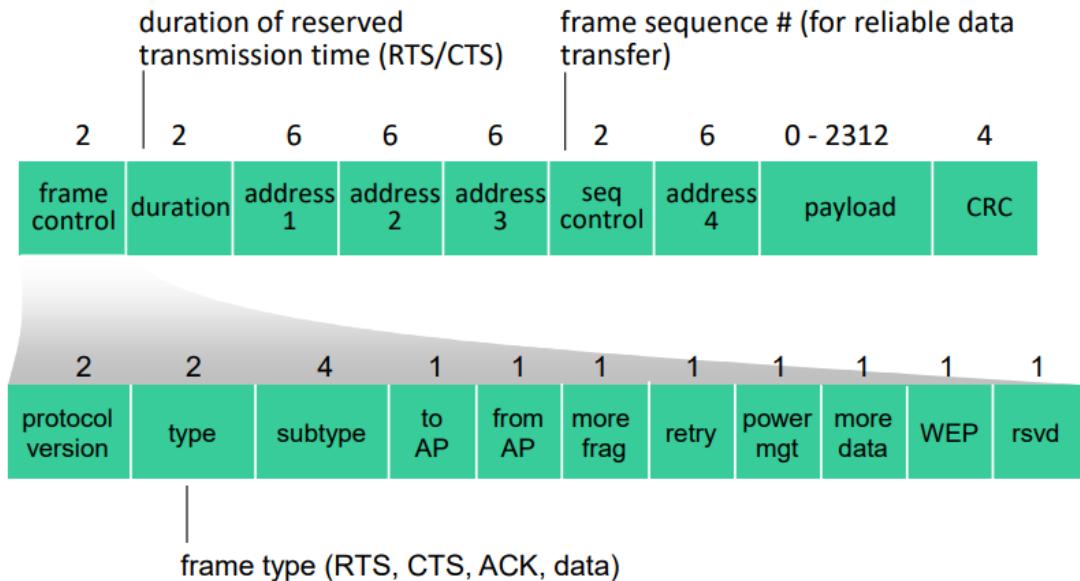
Address 1: MAC address of wireless host or AP to receive this frame

Address 2: MAC address of wireless host or AP transmitting this frame

Address 4: used only in ad hoc mode

Address 3: MAC address of router interface to which AP is attached

Sono presenti tre campi indirizzoMAC, il loro significato dipende dalla tipologia di frame:



Campo Duration

Campo contenuto nel CTS, quindi visibile anche da C.

Intervallo di tempo per cui il mezzo rimarrà occupato, questo intervallo di tempo è detto "NAV".

- Se il NAV è attivo il canale è occupato → *Si Aspetta*.
- Se il NAV è disattivo → *Si ascolta il canale*.

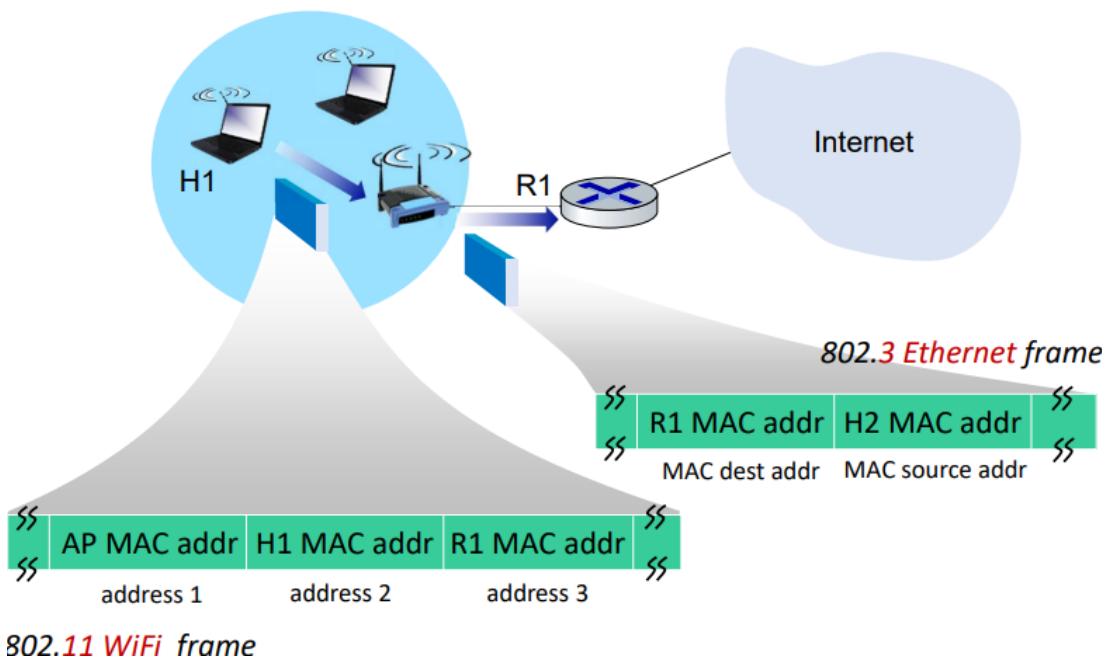
Indirizzamento del Frame nello standard 802.11

Come avviene il passaggio da rete wireless a cablata.

Il frame che va dall'host all'access point segue lo standard 802.11, invece il frame che va dal access point alla rete cablata ha un altro formato (non ha il campo che esprime l'indirizzo MAC dell'access point).

Quindi l'access point ha il compito di cambiare il formato del pacchetto da Wi-Fi (802.11) a Ethernet (802.3) e viceversa.

L'Access Point fa la stessa cosa che farebbe lo switch in una rete cablata.

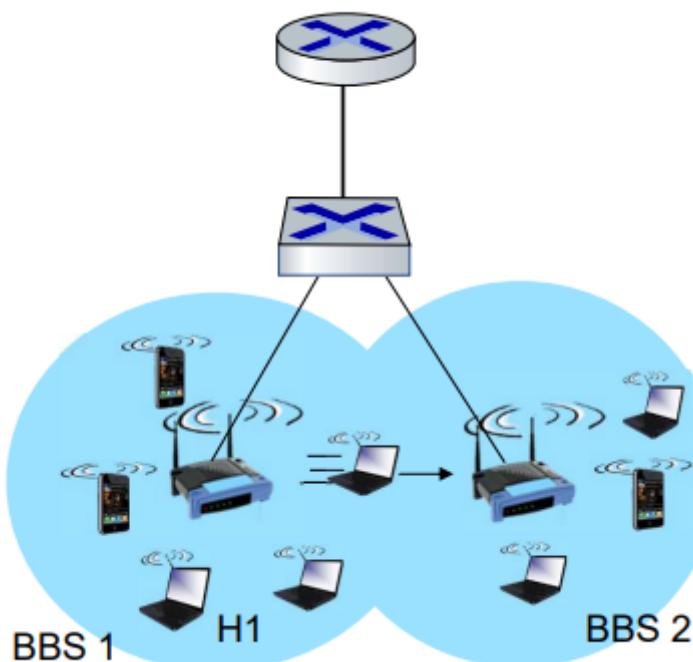


Nel frame come indirizzo MAC destinatario è specificato il MAC del router, questo perché il frame è effettivamente indirizzato al Router il quale poi usando l'indirizzo IP potrà inoltrare il pacchetto al destinatario originale.

Il Router riceverà il datagram, preleverà l'indirizzo IP del destinatario, lo tradurrà in un indirizzo MAC tramite il protocollo ARP e infine inoltrerà il pacchetto nella rete interna.

Gestione della Mobilità

Gli host connessi al Wi-Fi sono mobili e quindi può capitare che si passi dalla copertura di un access point ad un'altra di un altro access point.



Supponiamo che i 2 access point siano collegati tramite uno switch.

Ogni access point è collegato ad una specifica porta dello switch.

I frame inviati da un host wireless vengono ricevuti dall'access point e poi inoltrati allo switch.

Supponiamo che un host H1 si sposti da un BSS ad un altro, quindi effettui l'associazione con l'access point del BSS 2.

Ora H1 invia i suoi frame all'access point del BSS 2.

Lo switch si accorgerà dall'interfaccia di entrata relativa all'access point del BSS2 arrivano pacchetti da un host che non ha mai trasmesso da quella interfaccia, semplicemente aggiungerà una nuova riga alla tabella di switching.

Ogni riga della tabella di switching ha un TTL , allo scadere del quale la vecchia riga (quella relativa al BSS1) verrà eliminata.

L'unico effetto collaterale è che un eventuale pacchetto trasmesso da H1 durante la transizione dal BSS 1 al BSS 2 si potrebbe perdere, non è un problema, verrà trasmesso di nuovo poco dopo.

Capacità Avanzate - Rate Adaptation

La qualità della comunicazione può variare per la distanza dell'host ma anche per le condizioni dell'etere, se il luogo è soggetto ad elementi di disturbo la qualità del servizio diminuisce.

Quindi posso concludere che la qualità del servizio varia sia nello spazio che nel tempo.

A volte il *Bit Error Rate* diventa troppo elevato, talmente tanto che al momento del CRC si scartano troppi pacchetti e quindi l'impatto delle successive ritrasmissioni potrebbe diventare abbastanza pesante.

Le ritrasmissioni potrebbero fallire anche loro a causa dell'altissimo Bit Error Rate.

Occorre una codifica che a parità di rapporto segnale/rumore abbia un Bit Error Rate più basso.

Automaticamente, senza che l'utente lo sappia, l'interfaccia di rete passa da sola ad un'altra codifica che ha un Bitrate più basso e conseguentemente un Bit Error Rate più basso.

Ossia si rallenta la velocità di trasmissione per avere una qualità superiore in quest'ultima.

Capacità Avanzate - Power Management

Spesso i dispositivi wireless sono a batteria e quindi il consumo di energia elettrica è un problema primario.

In certi dispositivi la scheda di rete rappresenta una grossa parte del consumo elettrico. Quindi sarebbe cosa saggia ridurre la sua attività al minimo: Un nodo wireless non ha sempre pacchetti pronti da trasmettere/ricevere, quando non ha pacchetti pronti non ha bisogno di tenere l'interfaccia di rete accesa.

Ma spegnere e accendere è oneroso, quindi mando il circuito della scheda di rete in una fase di sleep dove l'energia elettrica viene erogata solo nella parte che si occupa di avviare il resto del circuito, in questo modo riavviare il circuito costa molto meno.

Per Migliorare il tutto:

- Nella fase di sleep eroga corrente solo al circuito che si occupa di riavviare l'intero circuito.

- In questa fase l'access point può lavorare al posto mio, poiché è sempre acceso ed è collegato alla rete elettrica fissa, quindi:
 - Prima di entrare nella fase di sleep si avverte l'access point che si entra nella fase di sleep.
 - L'access point invia periodicamente (ogni 100 ms) un Beacon Frame a tutti i nodi del BSS dove viene riportata la lista degli host mobili nel BSS che hanno "pacchetti pendenti" per loro.
 - I beacon frame vengono inviati con un certo periodo, quindi con un periodo impostato in modo opportuno l'host si sveglia poco prima dell'arrivo del beacon frame, in modo da poterlo ricevere.
 - L'host consulta il beacon frame
 - Se trova il suo nome nella lista riportata nel Beacon Frame chiede il frame pendente all'access point.
 - Se non ce ne sono torna in sleep.

Il risultato è che **l'interfaccia di rete lavora solo per il tempo strettamente necessario**. Un contro è il ritardo nella ricezione di un frame (almeno grande quanto il periodo del beacon).

In certe applicazioni posso disattivare questa opzione di power management e quindi annullare questo ritardo ma introducendo il problema del consumo.

Reti PAN (Personal Area Networks)

Reti con un raggio molto piccolo (pochi metri).

Una di queste è *Bluetooth*.

Bluetooth

Inizialmente pensata per introdurre periferiche esterne senza fili: tastiere, mouse, casse o cuffie.

Le connessioni di questo tipo hanno un basso Bit Rate e un basso consumo energetico.

Banda ISM → 2.4 Ghz

Frequenza dove si ha anche il Wi-Fi, quindi è una frequenza con numerosi elementi di disturbo.

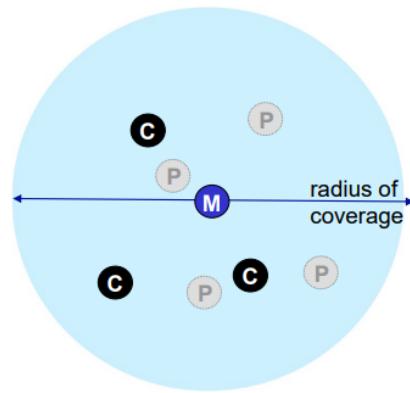
Dato che il segnale del Wi-Fi ha potenza maggiore, sovrasta il segnale del bluetooth.

La rete Bluetooth è chiamata *PicoNet*.

Ogni PicoNet ha un *Master Device* che comanda gli *Slave Devices*.

Gli slave devices sono l'equivalente dei client e hanno 2 possibili stati:

- Normale.
- Parked.
 - Stato in cui il client si mette per risparmiare batteria.



- M** master device
- C** client device
- P** parked device (inactive)

Il canale è condiviso, quindi per la regolazione del canale si utilizza la tecnica del Polling, ossia il master che coordina gli slave.

A turno il Master invia un pacchetto di Poll ad uno slave, chi ha il Poll può trasmettere un pacchetto oppure rispondere con `NULL` (se non ha nulla da trasmettere).

In bluetooth il canale non è bidirezionale, quindi un solo trasmettitore alla volta.

L'accesso segue un *Time Division Management* con slot da 625 Microsecondi.

Il Tempo è diviso in slot temporali, in ogni slot ho il tempo necessario a trasmettere un pacchetto.

Gli slot pari(o dispari) sono usati da Master, nei quali invia il pacchetto di Poll.
Gli slot dispari(o pari) viene data agli Slave.

Bluetooth - Frequency Hopping

Il trasmettitore usa 79 canali di frequenza.

Nel frequency hopping Master e Slaves cambiano canale ad ogni pacchetto in maniera coordinata.

La sincronizzazione viene fatta periodicamente dal master, dove tutti gli slave di sincronizzano al clock del master.

Supponiamo che nella frequenza 5 ho un disturbo: io mi becco il disturbo solo 1 volta su 79.

Da quale frequenza partiamo?

Se tutti usano una frequenza iniziale predeterminata in caso di 2 master nella stessa area di copertura essi comunicheranno sempre contemporaneamente e quindi avrei sempre un disturbo in tutti i 79 i canali e perciò nessuno dei due master riuscirebbero a comunicare.

Invece se io usassi come frequenza di partenza una frequenza pseudo-casuale riduco di molto la probabilità (passiamo da 1 a 1/79) che due master comunichino alla stessa frequenza.

Questo aiuta anche in caso di interferenze prodotte da un attaccante, se la frequenza è predeterminata esso è in grado di disturbare tutte le comunicazioni , invece se la frequenza è pseudo-random l'attaccante dovrà spendere tempo a cercarla e quindi c'è più possibilità di rilevarlo.

Reti Cellulari

Per reti wireless a vasta copertura.

Ne abbiamo di due tipi 4G e 5G.

Il "G" sta per "Generation"

1. Rete Cablata.
2. Prima rete Digitale.
3. Per trasmettere voce, video e dati.
4. Quella attuale tutti i dispositivi del core seguono il protocollo IP.
5. La prossima.

4G - Fourth Generation

Evoluzione di LTE (*Long-Term Evolution*) Standard.

Somiglianze con internet:

Ha una parte edge e una parte Core, ma in 4G entrambe le parti appartengono e vengono gestite da una sola organizzazione (wind, vodafone etc).

Si usano molti protocolli di internet.

Spesso le reti 4G sono connesse con Internet tramite i collegamenti classici.

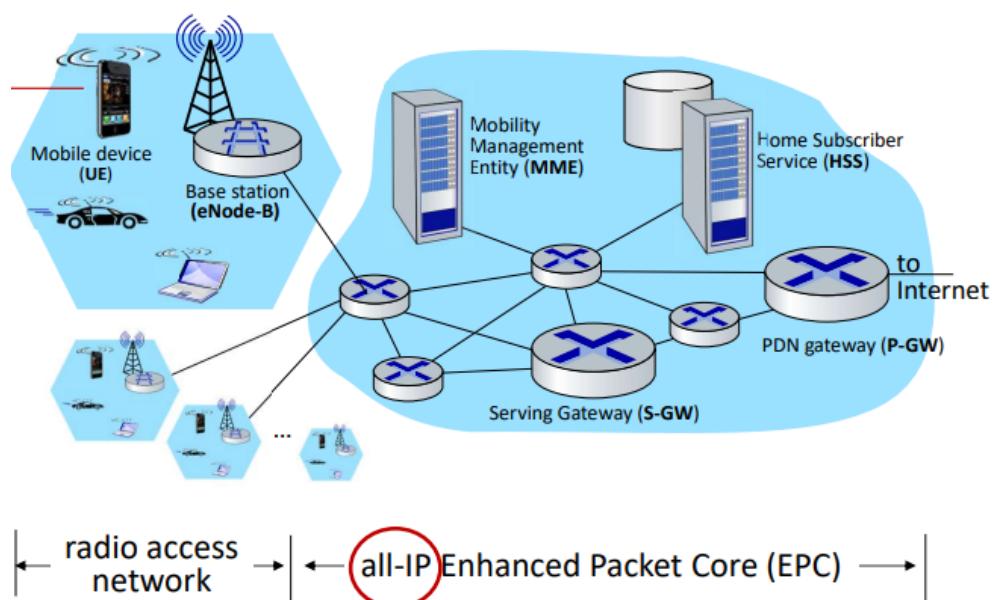
Differenze con internet:

- Link wireless diverso.
- L'identità degli utenti avviene tramite un identificatore all'interno della SIM card.
- Per accedere ad una rete 4G gli user devono pagare un'iscrizione con un provider.

Rete 4G/5G

Si divide in 2 parti:

- Radio Access Network
- all-IP Enhanced Packet Core (EPC).



- **Mobile Device** → UE
- **Base Station** → eNode-B

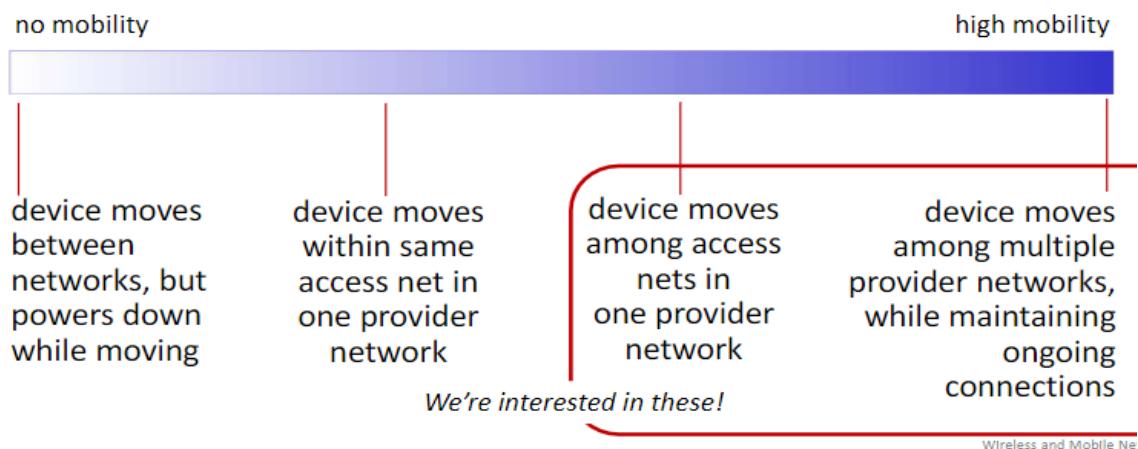
- Gestisce tutti i cellulari nella sua area di copertura.
- Ha un ruolo attivo nella mobilità dell'utente.
- Si coordina con le Base Station vicine per ottimizzare il passaggio da un'area di copertura all'altra.
- **Home Subscriber Service (HSS)**
 - Memorizza le informazioni relative ai dispositivi mobili iscritti al servizio.
- **Serving Gateway (S-GW)**
 - Spesso svolgono anche la funzione di MME.
- **PDN Gateway (P-GW)**
 - Funge da ponte tra Internet (quella classica) e la rete 4G.
- **Mobile Management Entity (MME)**
 - Si occupa di fare l'autenticazione dei dispositivi e si coordina con il HSS.
 - **Ha il compito di tracciare la posizione del dispositivo.**
 - Ha il compito di effettuare il trasferimento del dispositivo da una cella all'altra in caso di mobilità.

Mobile IP

Mobile IP è un protocollo progettato per consentire agli utenti con dispositivi mobili di spostarsi da una rete all'altra mantenendo un indirizzo IP permanente.

Quindi che permette di mantenere intatte le connessioni TCP.

La mobilità può essere espressa secondo varie sfumature:



Noi vorremmo tecniche per gestire la maggior parte di questi casi.

Per gestire si intende fare in modo che la degradazione del servizio non sia troppo grande.
A noi interessano i 2 casi a destra.

Estremo Sinistro

Detta anche “No Mobility” o “Nomadicità”.

La persona si trova per un pò in una locazione, poi spegne il dispositivo (lo disconnette) e dopo un pò di tempo si connette in una locazione completamente diversa.

Ossia il movimento è fatto mentre non si è connessi alla rete.

Le connessioni TCP e il servizio viene interrotto durante il movimento perché la persona cambia del tutto rete di accesso e quindi anche indirizzo IP.

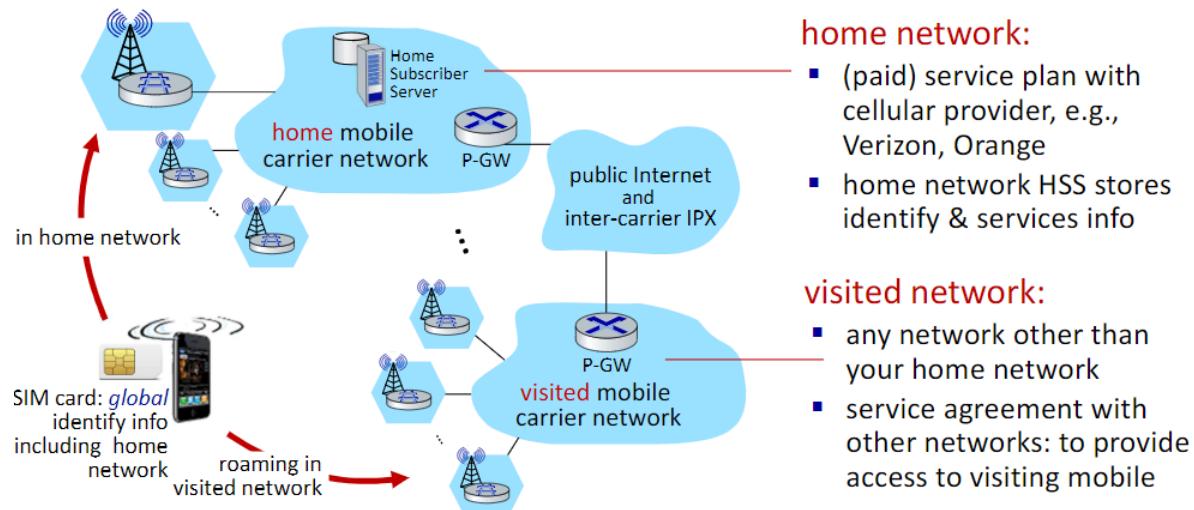
In questo caso purtroppo non c'è molto che possiamo fare...

Estremo Destro

Detta anche *"High Mobility"* dove il movimento è fatto tenendo il dispositivo acceso (e ovviamente connesso) e sempre nella stessa rete di accesso (quindi cambia Access Point ma l'indirizzo IP non cambia perché la rete di accesso rimane la stessa), le connessioni TCP rimangono inalterate perché l'indirizzo IP dell'host non cambia.

Approcci per la Gestione della Mobilità

Il nostro obiettivo è quello di preservare le connessioni TCP (quindi evitare le interruzioni del servizio) nel caso in cui un Host cambi rete di accesso.



Approccio Router

Non praticabile concettualmente e nel pratico, perché non è scalabile su miliardi di dispositivi.

Approccio end-systems - Routing Indiretto

La comunicazione tra il mittente e il destinatario è indiretta.

Uato da Mobile IP.

Chi vuole comunicare con un host mobile si riferisce solo alla rete base dell'host mobile, ogni comunicazione che parte dal mittente a quell'host passa dalla rete base del destinatario e poi verrà inoltrata all'host destinatario mobile dal router della Home Network.

Approccio end-systems - Routing Diretto

La comunicazione tra il mittente e il destinatario è diretta.

Inizialmente chi vuole comunicare con un host mobile si riferisce alla rete Base dove ottiene l'indirizzo che attualmente possiede l'host mobile.

Dopo averlo ottenuto comunicherà direttamente con l'host senza passare di nuovo dalla Home Network.

IMSI

Lo IMSI è codificato fisicamente nella SIM.

Non può cambiare ed è unico.

E' l'equivalente dell'indirizzo MAC per le schede di rete.

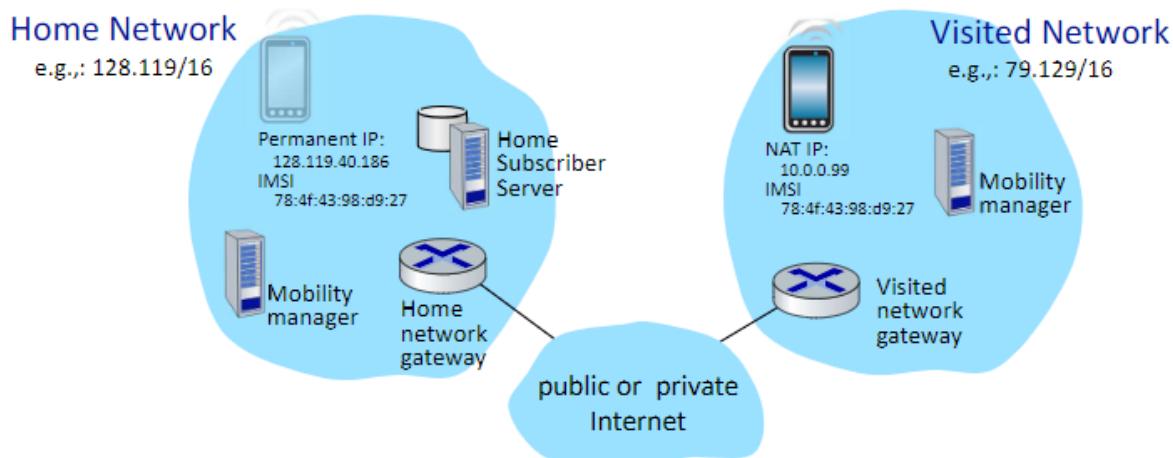
La SIM card identifica l'host mobile e indica anche la Home Network al quale appartiene.

Home Network e Visited Network

La home network è detta anche “Rete Base”.

L'utente è registrato in una Home Network (precisamente, l'utente è registrato in un HSS della sua Home Network).

Ogni Host all'interno della Home Network è identificato da un indirizzo IP e l'Identificativo nella Rete Cellulare detto IMSI.



Supponiamo che il dispositivo e relativo utente possa fare roaming e quindi possa trovarsi in una rete posseduta da un provider al quale il dispositivo non è registrato (quest'ultima è detta “*Visited Network*”).

Quando il dispositivo accede alla rete visitata può essergli assegnato un indirizzo IP pubblico o privato (l'iMSI non cambia perché è scritto nella SIM).

Sia la Home Network che la Visited Network hanno un Mobility Manager.

Supponiamo poi che ci sia un Host esterno ad entrambe le reti che vuole comunicare con il dispositivo.

Il Servizio non dovrebbe interrompersi in caso di mobilità tra la Visited Network e Home Network.

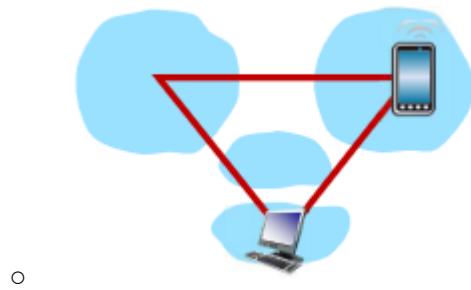
1. Quando il dispositivo si sposta nella Rete Visitata deve registrarsi in essa, quindi informa il Mobility Manager della Visited Network e si registra.
2. Il Mobility Manager informa la Home Network (più precisamente, contatta l'HSS nella home network) che l'host mobile si trova nella rete visitata.
 - a. Quindi la Home Network alla fine della fiera conosce sempre la posizione dell'host.
3. Il Correspondent (l'host esterno) contatta la Home Network inviando un datagram, destinandolo all'indirizzo IP che avrebbe avuto il dispositivo destinatario nella sua Home Network.
4. Il dispositivo non è nella Home Network, quindi quando il HSS della Home Network riceve il Datagram lo instrada verso la Visited Network.
5. A quel punto il Visited Network Gateway lo instrada al dispositivo destinatario.

Nota che il correspondent non dovrebbe sapere dove si trova ora il dispositivo (per la privacy).

Il Problema della Triangolazione

Il dispositivo mobile dovrà anche rispondere al datagram.

- Se il routing è indiretto
 - Il datagram di risposta deve passare dalla Home Network e poi essere inviato al correspondant, è inefficiente ma fornisce trasparenza e privacy al dispositivo mobile, perché il correspondent vede che la risposta viene dalla Home Network e quindi non conosce la vera posizione del destinatario.
- Se il routing è diretto.
 - Il datagram di risposta andrà dalla visited network direttamente al correspondent, è più efficiente ma ha il problema della triangolazione, perché il correspondent scopre il nuovo indirizzo IP e in quale rete si trova l'host mobile.
 - La Mobilità diventa "Non Trasparente".



Successiva Mobilità

Supponiamo che l'host mobile dalla Visited Network si sposti in un'altra Visited Network.

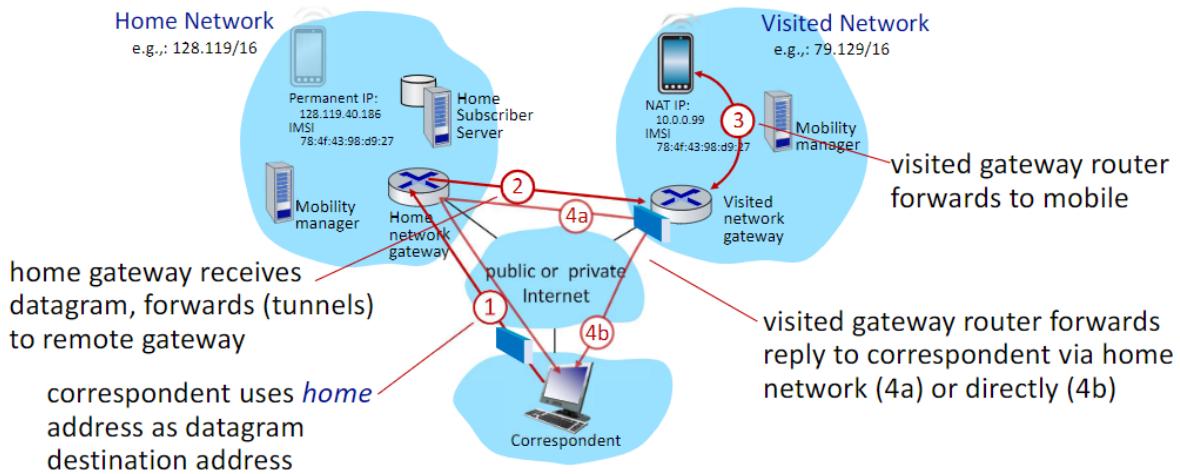
Si ripete il procedimento fatto prima con la Home Network.

Che succede se nel frattempo ho un flusso di dati in corso?

Caso Indiretto

Continua a funzionare bene, perché la nuova visited network contatterà lo HSS della home network e l'home gateway si configura in modo da inviare i datagram nella nuova rete mantenendo la trasparenza.

L'unico impatto è la perdita di pacchetti mentre si sta facendo la registrazione nella seconda. In quel periodo l'host si trova nella "Terra di Nessuno", in questo frangente perderà pacchetti, ma non è un grande problema, al massimo si ritrasmetterà qualche datagram (si rimane nella logica Best Effort del protocollo IP).



Caso Diretto

La home network informerà il correspondent della nuova rete in cui si trova l'host.

Il correspondent dovrà inviare i datagram nella nuova rete, quindi ad un nuovo indirizzo IP. Un'eventuale connessione TCP attiva viene eliminata (poiché ricordiamo che l'indirizzo IP di destinazione è uno dei 4 parametri fondamentali di una connessione TCP), la continuità del servizio non è garantita.

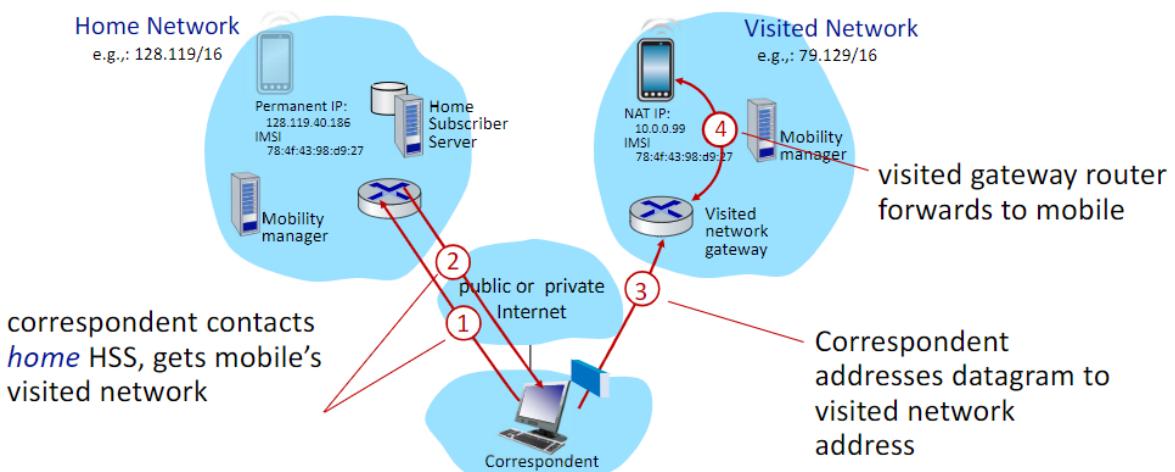
La soluzione per mantenere la connessione TCP è abbastanza contorta.

Quando un host si sposta nella Rete Visitata 2, quest'ultima informa la Rete Visitata 1 che deve inoltrare i pacchetti nella Rete Visitata 2.

La Home Network in tutto questo non viene informata, lei continuerà a pensare che l'host si trova nella Rete Visitata 1.

Il correspondent continuerà a mandare i pacchetti alla Rete Visitata 1, la quale inoltrerà i pacchetti alla Rete Visitata 2 la quale inoltrerà il pacchetto al destinatario.

Quindi alla fine si crea questa catena di inoltre tra le reti visitate, ma dato che il correspondent non si accorge di questa cosa (l'indirizzo IP del destinatario rimane quello della rete visitata 1), la continuità del servizio è garantita.



Impatto della Mobilità sui Protocolli di Livello Superiore

Durante la transizione tra una rete all'altra si perdono dei pacchetti , ma ciò è in linea con la filosofia Best Effort di IP.

Il Protocollo di Connessione TCP assume che ad ogni perdita è dovuta a congestione , in presenza di collegamenti wireless il pacchetto si potrebbe perdere anche perché il segnale ha subito disturbi oppure per la mobilità (tra la mobilità tra un access point all'altro).

Per il TCP ogni perdita è dovuta a congestione, attiva il protocollo della congestione e quindi il throughput si abbassa molto.

Domande Wireless e Mobile Network

■ Problema del Nodo Nascosto e Nodo Esposto - Da cosa deriva il nome

Supponiamo di avere i nodi A, B e C.

A e C vogliono trasmettere verso B.

A e C non si sentono a vicenda.

Per il problema del nodo nascosto, il nome deriva dal fatto che il Nodo C è “nascosto” rispetto ad A e viceversa.

Per “Nascosto” si intende che uno non riesce a rilevare la trasmissione dell'altro.

Supponiamo di avere i nodi A, B, C e D.

A vuole trasmettere verso B.

C vuole trasmettere verso D.

Tutti si sentono a vicenda

Per il problema del nodo esposto, il nome deriva dal fatto che il Nodo D è “esposto” rispetto ad B e viceversa.

Per “esposto” si intende che le trasmissioni che C vorrebbe fare verso D verrebbero sicuramente disturbate da tutte le trasmissioni verso B.

■ Problema del Nodo Nascosto - Ci sono due modi di raffigurarla, dirli entrambi.

Supponiamo di avere i nodi A, B e C.

A e C vogliono trasmettere verso B.

A e C non si sentono a vicenda.

Il problema del nodo nascosto può essere rappresentato in 2 modi:

La distanza tra A e C è talmente grande che il segnale di uno arriva all'altro con una potenza talmente bassa da essere trascurabile.

Tra A e C c'è un ostacolo fisico, il quale causa un depotenziamento del segnale talmente grande che il segnale di uno arriva all'altro con una potenza talmente bassa da essere trascurabile.

■ Problema del Nodo Nascosto- chi riceve i messaggi se inviati da uno specifico nodo?

I messaggi inviati tramite un canale di tipo wireless sono ricevuti da tutti, perché il segnale è composto da onde elettromagnetiche che si diffondono verso tutte le direzioni.

La ricezione di queste onde avviene tramite un'antenna che funge da spira.

Ergo, ogni nodo in grado di sentire la trasmissione la riceverà.

Problema del Nodo Nascosto- come si risolve il problema

Il problema del nodo nascosto è risolto usando il Virtual Carrier Sensing.

Ossia si fa in modo che le trasmissioni rivolte verso uno specifico Nodo siano regolate.

Se i Nodi A e C vogliono trasmettere qualcosa verso B, allora ognuno di essi deve inviare un RTS, il quale avverte il nodo B che un nodo vorrebbe trasmettergli qualcosa.

Il nodo B sceglie il nodo trasmettitore e invia un CTS(A), con il quale informa tutti i nodi (compreso C) che solo il nodo A è autorizzato a trasmettere qualcosa.

CTS(A) contiene anche un campo che compone il NAV, il quale farà da valore iniziale del timer per C.

Se C ha ancora il timer NAV attivo non può trasmettere anche se il canale risulta libero.

Gli RTS possono collidere.

Protocollo CSMA/CA - Specificare i Passaggi

CSMA/CA è un protocollo con l'obiettivo di ridurre la possibilità che avvenga una collisione.

Questo perché nei link di accesso wireless non tutti i trasmettitori potrebbero accorgersi della collisione, a causa del problema del nodo nascosto.

Ogni nodo ascolta il canale, se risulta libero per un tempo pari a DIFS allora risulta libero. Prima di trasmettere aspetta un tempo pari a Backoff Timer, tempo randomico calcolato prima di trasmettere.

Se durante il Backoff Timer il canale risulta sempre libero, allora allo scadere di quest'ultimo, si trasmette.

Il calcolo del Backoff Timer avviene tramite un algoritmo "adattivo", ossia maggiore è il numero di collisioni e maggiore sarà il valore massimo del Backoff Timer e quindi minore sarà la probabilità che due trasmettitori trasmettono allo stesso istante.

Il ricevitore dopo aver decodificato e dopo aver eseguito i controlli di integrità, ascolta il canale per un tempo pari a SIFS < DIFS.

Se dopo SIFS il canale è libero si Trasmette il segnale di ACK.

Protocollo CSMA/CA - calcolo del tempo di backoff

Il calcolo del Backoff timer avviene tramite un backoff algorithm.

L'algoritmo usato è di tipo "adattivo", ossia si adatta al numero di collisioni che avvengono sul canale.

Backoff Timer è un valore randomico compreso tra 0 e CW-1.

Ogni volta che avviene una collisione il valore di CW viene raddoppiato.

Il Backoff timer serve a sincronizzare i trasmettitori in attesa che il canale sia libero, se non ci fosse il Backoff Timer, tutti i trasmettitori trasmetterebbero dopo DIFS.

Un nodo trasmettitore conclude che si è verificata una collisione quando non riceve ACK di risposta.

Protocollo CSMA/CA - Specifica nel dettaglio perché DIFS e SIFS hanno durata differente

CSMA/CA impone che $DIFS > SIFS$ perché noi vogliamo dare priorità alle risposte a discapito delle nuove trasmissioni.

Questo perché se la risposta arrivasse troppo tardi il Nodo trasmettitore concluderebbe che si sia verificata una collisione.

Quindi aumenterebbe il range di valori del backoff timer portandolo ad essere più lento del necessario.

Mobile IP - Cosa vuol dire che cambia il punto di accesso?

Un Host Mobile (come ad esempio un telefono) ha un range di mobilità molto ampio.

Il Fornitore del Servizio a cui è iscritto non può avere una copertura globale e quindi spesso un host mobile finisce nella rete di accesso di una seconda network (detta Visited Network).

Nonostante ciò, l'obiettivo di Mobile IP è quello di garantire la continuità del servizio e quindi anche il mantenimento delle connessioni TCP in corso.

Mobile IP - Come possiamo permettere le continuità del servizio in caso di mobilità?

Dipende dalla modalità di routing selezionata.

Ma comunque in entrambi i casi un Host Mobile iscritto ad un certo servizio ha delle informazioni contenute nell'HSS (Home Subscriber Service) e nel MME.

Quest'ultimo ha lo scopo di tracciare la posizione dell'host in ogni momento, quindi la Home Network sa se un certo host in questo momento è Spento, Dentro la Home Network o in un'altra Network.

Quando un host entra nel raggio di una nuova network si iscrive ad essa e il Mobility Manager della Visited Network informa il Mobility Manager della Home Network.

Il dispositivo (che chiameremo correspondent) che desidera comunicare con noi invia la richiesta alla Home Network.

Se il routing è diretto.

Al correspondent gli viene restituito l'indirizzo IP corrente (quello relativo alla visited network) dove in questo momento risiede l'host.

Il correspondent infine invia la richiesta direttamente all'indirizzo IP, e quindi direttamente all'host desiderato.

Se il routing è indiretto.

La Home Network inoltrerà la richiesta al router della visited network, il quale lo inoltrerà all'host effettivo.

La risposta passerà di nuovo dalla Home Network.

Quindi la home network funge da intermediario tra il correspondent e il destinatario.

La posizione del destinatario risulta essere la Home Network agli occhi del correspondent.

Mobile IP - Come viene gestito lo spostamento nelle due modalità di routing indiretto e diretto?

Se il routing è diretto.

Se il routing è indiretto.

Quando l'host si sposta verso un'altra Network avviene il solito procedimento.

Si iscrive alla nuova network e il Mobility Manager della Visited Network informa il Mobility Manager della Home Network.

Semplicemente ora le richieste del Correspondant verranno inoltrate verso il router della nuova visited network, le richieste del correspondant non devono essere modificate e la connessione TCP precedentemente instaurata viene mantenuta.

Se il routing è indiretto.

Lasciato così fallirebbe, perché spostandosi in una nuova visited network, l'host cambia indirizzo IP e dato che la connessione TCP tra il correspondant e l'host era diretta essa viene spezzata.

Mobile IP - Quale approccio utilizza il mobile ip?

Mobile IP usa un approccio Indiretto.

Parte del Prof. Pistolesi

Comando “ip”

Visualizza e manipola le impostazioni di rete.

```
ip [options] object { command | help } object := {link | address | route | ... }
```

lo (loopback) → Interfaccia di rete che permette di inviare messaggi a noi stessi, utile per testare le applicazioni.

eth0 () → Una delle interfacce di rete “vere” della macchina che possiamo usare per inviare messaggi ad altre macchine.

```
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
```

Comando “ip addr show”

ip addr show → Mostra tutte le interfacce.

ip addr show up → Mostra tutte le interfacce accese.

- LOWER_UP:
 - Significa che il “Cavo è Collegato”.
- *mtu (Maximum Transmission Unit)*
 - Esprime la dimensione massima di un datagram IP.
- *qdisc (Queuing Discipline)*
 - Stabilisce il prossimo pacchetto da inoltrare.

- *state*
 - UP
 - La Scheda è Abilitata.
 - DOWN
 - La Scheda è Disabilitata.
- *qlen (Transmission Queue Length)*
 - Lunghezza della coda di trasmissione.
- *link/ether + <Ind. MAC>*
 - Il protocollo data link usato dalla scheda è Ethernet
 - L'indirizzo MAC specificato è l'indirizzo fisico della scheda.
- *brd + <Ind. MAC>*
 - Esso è l'indirizzo MAC di broadcast.
- *inet + <Ind. IP>*
 - Si sta usando il protocollo internet a livello network, poi è specificato l'indirizzo IP della scheda e maschera in notazione compatta.
- *brd + <Ind. IP>*
 - Indirizzo IP di broadcast.

```
ip link set eth0 {up | down}
```

Prendo l'oggetto *link* e gli dico di settare eth0 a *up* o *down*, ossia **serve a abilitare o disabilitare un interfaccia di rete.**

```
ip addr add/delete 192.168.1.42 / 24 broadcast 192.168.1.255 dev
eth0
```

Aggiungo l'indirizzo IP all'interfaccia eth0.

Un'interfaccia di rete può avere molteplici indirizzi IP.

Pacchetto ifupdown

Ma ogni volta che riavvio la macchina, ciò che faccio con il comando ip va perduta.

Come faccio a **rendere una configurazione persistente?**

La configurazione manuale va perduta, ma esiste il file di configurazione.

Permette la configurazione permanente, il file di configurazione si trova in **etc/network/interfaces**

Se lo apro:

auto lo → *All'avvio accendi l'interfaccia di Loopback.*

iface lo inet loopback

iface eth0 inet static → *All'avvio la configurazione di eth0 sono quelli nel file di configurazione.*

```
address 192.168.1.2
netmask 255.255.255.0
broadcast 192.168.1.255
```

- *ifup eth0 :*
 - *Abilita eth0, con la configurazione specificata nel file.*
- *ifdown eth0 :*
 - *Disabilita eth0.*

- *ifup -a* :
 - Abilita tutte le interfacce tutte le interfacce specificate nella sezione auto nel file di configurazione.

Invio dei Pacchetti

La definizione di pacchetto varia a seconda del livello di riferimento.

```

dest_subnet := my_netmask &
dest_addr;

if(dest_subnet == my_subnet) then
    deliver to dest_addr;
else
    forward to default_gateway;
end if
  
```

Nota che ogni router sa quale è il prossimo Hop (salto) per creare il cammino minimo, non ha la visione dall'alto, non ha la visione della congestione, sa solamente cosa c'è intorno a lui.

Se il destinatario di un pacchetto è nella stessa sottorete, il router non viene coinvolto e il pacchetto è consegnato direttamente al destinatario attraverso il canale.

Se il destinatario di un pacchetto è in una sottorete diversa, il mittente invia il pacchetto all'indirizzo IP specificato nel Default Gateway, che corrisponde all'indirizzo IP del router.

Comando Route

Manipola le tabelle di routing IP del kernel.

Il suo utilizzo principale è impostare percorsi statici verso host o reti specifici tramite un'interfaccia dopo che è stata configurata con il programma ifconfig.

Quando vengono utilizzate le opzioni add o del, route modifica le tabelle di routing. Senza queste opzioni, route visualizza il contenuto corrente delle tabelle di routing.

- Mi mostra l'indirizzo del default gateway.
 - *ip route show*
- Mostrare la tabella di routing
 - *netstat -rn*
- Invio diretto nella rete locale:
 - *ip route add/delete 192.168.1.0/24 dev eth0*
- Scoprire la rotta usata per un certo indirizzo IP.
 - *ip route get 70.143.3.67*
- Aggiungere il Default gateway
 - *ip route add/delete default via 192.168.1.1*
- Posso aggiungere anche delle ruote in modo manuale, senza aggiornare il file di configurazione.
 - *ip route add/delete default via 192.168.1.1 / 24 dev eth0*

Note

- Notazione Compatta: \20 => "20 bit identificano la rete".

Risoluzione Statica dei Nomi

Il file “`etc/hosts`” contiene alcune risoluzioni dei nomi.

Ossia una lista di record del tipo < indirizzo IP , Nome >.

```
127.0.0.1      localhost
127.0.1.1      studenti

151.101.37.140 www.reddit.com
131.114.73.85  www.unipi.it
```

man hosts

DNS

Gli indirizzi dei server DNS sono messi in `/etc/resolv.conf`.

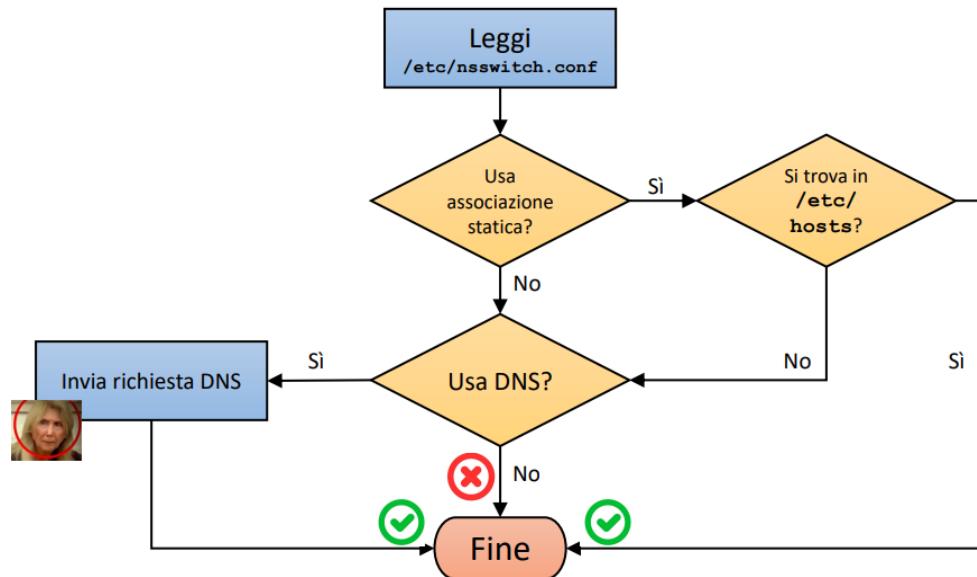
Esso contiene la lista degli indirizzi IP dei server DNS che l'host può contattare (e quindi effettuare una query DNS) nel caso in cui il nome simbolico non appaia nella lista in `etc/hosts`.

La lista dei DNS del file “`resolv.conf`” è del tipo:

`nameserver 8.8.8.8`

`nameserver 8.8.4.4`

Per effettuare una richiesta DNS manualmente: `nslookup www.google.it`



Name Service Switch

Meccanismo che i sistemi UNIX usano per ricavare nomi di cose da diverse fonti, nel nostro caso l'host.

il file `etc/nsswitch.conf` specifica le fonti da usare e l'ordine in cui usarle.

ESERCIZI

1. Visualizzare la configurazione delle interfacce di rete
 - a. `ip addr show` → *ti mostra tutte le tue interfacce di rete e i loro dati.*
2. Visualizzare solo la configurazione di eth0 `ip addr show eth0`
 - a. Che indirizzo IP e Mask sono impostate? → *Non sono ancora state impostate.*
 - b. Che indirizzo MAC ha la scheda di rete? → *08:00:27:47:fe:bc*
 - c. Cos'e' l'MTU e quanto vale? → *MTU Esprime la dimensione massima di un pacchetto IP , e vale 1500 bit.*
3. Impostare come indirizzo IP dell'interfaccia eth0 l'indirizzo 10.0.2.15, maschera 255.255.255.0
 - a. `sudo ip addr add 10.0.2.15/24 broadcast 10.0.2.255 dev eth0`
4. Abilitare l'interfaccia di rete
 - a. `sudo ip link set eth0 up`
5. Visualizzare la tabella di routing → `ip route show`
 - a. Qual e' il default gateway? → *Non è stato ancora impostato.*
6. Impostare come default gateway l'host 10.0.2.2
 - a. `sudo ip route add default via 10.0.2.2`
7. Visualizzare la rotta usata per raggiungere 192.168.0.27
 - a. `ip route get 192.168.0.27` → *192.168.0.27 via 10.0.2.2 dev eth0 src 10.0.2.15*
8. Visualizzare l'indirizzo del server DNS in uso
 - a. `cat /etc/resolv.conf` → *nameserver 10.0.2.3*
9. Cambiare l'indirizzo del server DNS in 8.8.8.8
 - a. `sudo nano /etc/resolv.conf` → *scrivi manualmente l'indirizzo DNS voluto*
10. Scoprire l'indirizzo IP di Imgify.com → `nslookup imgify.com` → *18.205.222.128*
11. Rimuovere l'indirizzo IP di eth0 tramite ip
 - a. `sudo ip addr delete 10.0.2.15 / 24 broadcast 10.0.2.255 dev eth0`
12. Disattivare eth0 → `sudo ip link set eth0 down`
13. Fate una copia di backup del file interfaces (es. interfaces.bak)
 - a. `cd /etc/network`
 - b. `touch interfaces.bak` → *Se non esiste crea il file*
 - c. `cp interfaces interfaces.bak`
14. Impostare l'indirizzo 10.0.2.15, la netmask 255.255.255.0, e attivare la scheda di rete usando ifup e il file interfaces (Cancellare le impostazioni di eth0 presenti)
 - a. `nano interfaces`
 - b. `iface eth0 inet static ; address 10.0.2.15 ; netmask 255.255.255.0 ; broadcast 10.0.2.255.`
 - c. `iup eth0` → *Se lo fai prima del punto b ti dirà che eth0 non esiste.*
15. Aggiungete al file la sezione per impostare la scheda all'avvio
 - a. `nano interfaces`
 - b. `auto lo eth0`

DHCP

Installazione del Server DHCP

Installazione → `apt-get install isc-dhcp-server`

Installo sulla macchina un processo Server DHCP, al termine della quale andrò ad editare un file di configurazione

File di Configurazione del Server DHCP

File di Configurazione → `etc/default/isc-dhcp-server`

Manuale → man dhcpcd.conf

INTERFACES="eth0"

```
option domain-name-servers 192.168.0.2 , 8.8.8.8
option router 192.168.0.1
default-lease-time 3600
subnet 192.168.0.0 netmask 255.255.255.0 {
    range 192.168.1.10 192.168.1.100;
}
```

1. 192.168.0.2 → Un server DNS da assegnare ai miei Client.
2. 8.8.8.8 → Un server DNS da assegnare ai miei Client.
3. router 192.168.0.1 → Il gateway da assegnare ai miei Client.
4. default-lease-time 3600 → Durata configurazione in secondi.
5. L'ultima riga esprime il range di indirizzi assegnabili dinamicamente tramite DHCP.

Dopo le modifiche

systemctl restart isc-dhcp-server.service → rendo permanenti le modifiche.

Impostare la configurazione DHCP sull'host

Nel file etc/network/interfaces → **iface eth0 inet dhcp**.

Metto "dhcp" al posto di "static", dico al sistema operativo che quella interfaccia di rete deve essere configurata mediante DHCP.

Subito dopo la scheda di rete effettuerà una richiesta DHCP.

Test di Connessione - Internet Control Message Protocol - Protocollo ICMP

ICMP è un protocollo di servizio a supporto del protocollo IP che ha la funzione di rilevare malfunzionamenti e scambiare messaggi di controllo o di errore.

• Comando Ping

- Il comando ping serve a contattare un destinatario nella mia rete oppure fuori dalla mia rete, questo comando può avere successo o può fallire.
 - Il comando dopo essere stato lanciato entra in un loop dove la mia macchina manda di continuo messaggi di richiesta ICMP.
 - Di default ne viene inviato 1 al secondo.
- Il Mittente impone un timeout (*time-to-live* o *TTL*) espresso in millisecondi oltre al quale si presuppone che il pacchetto sia stato perso → *Packet Loss*.
 - Mando → *ICMP Echo Request*.
 - Ricevo → *ICMP Echo Reply*.
- Man mano che le risposte arrivano, la mia macchina stila delle statistiche:
 - *Round Trip Time*:
 - Il tempo passato tra l'invio della richiesta e l'arrivo della risposta.
 - Di solito è nell'ordine di millisecondi.
 - Tra 2 coppie (richiesta-risposta) può variare perché lo stato della rete varia continuamente nel tempo inoltre la route tra le due macchine potrebbe anche cambiare.
 - Come fa il Mittente a capire se è stato perso il messaggio di richiesta oppure quello di risposta?

- Se il destinatario è nella rete e non lo riesco a contattare:
 - La configurazione della mia macchina è errata o assente.
- Se il destinatario NON è nella rete e non lo riesco a contattare:
 - La configurazione della mia macchina è errata o assente.
 - La configurazione del gateway è errata o assente.
 - La configurazione del server DNS è errata o assente.
 - Verificabile se il ping funziona con l'indirizzo IP ma non con il nome simbolico.
 - La configurazione del mio firewall è errata o assente.
 - Il firewall della rete che cerco di contattare non mi permette di contattarlo.
 - Il mio ISP ha problemi tecnici e i loro router sono fuori servizio.
- Possibili Errori del comando Ping:
 - *Network Unreachable*.
 - *100% Packet Loss*.
 - Il mittente non ha ricevuto nessuna risposta.
 - *Unknown Host*.
 - Non è stato possibile risolvere il nome dell'host, è un indizio per un errore che riguarda il mio DNS.

Comando traceroute

Estensione del comando ping, perché fa la stessa cosa di ping (mi dice se il destinatario ha risposto) però mi dice anche la route che fa il pacchetto dal mittente verso il destinatario.

Il percorso (route) è rappresentato dagli indirizzi IP dei router traversati.

I pacchetti utilizzano il protocollo UDP:

1. Vengono inviati verso il destinatario 3 pacchetti.
 - E' rarissimo che quei 3 pacchetti percorrano 3 strade diverse, in quel caso si hanno delle statistiche false.
2. Il Mittente iniziale impone TTL = 1 in ognuno dei 3 pacchetti e manda la terna.
3. R1 riceve la terna e decrementa il TTL (TTL → 0) in tutti e 3 i pacchetti.
4. R1 invia al mittente iniziale un messaggio di errore ICMP di tipo *Time Exceeded* con l'indirizzo del router che ha portato TTL a 0, quindi l'indirizzo IP di R1.
5. Il Mittente iniziale impone TTL = 2 in ognuno dei 3 pacchetti e manda la terna.
6. R1 decrementa il TTL e manda i pacchetti al prossimo router (R2).
7. R2 decrementa il TTL (TTL → 0), invia al mittente iniziale un messaggio di errore ICMP di tipo *Time Exceeded* con l'indirizzo del router che ha portato TTL a 0, quindi l'indirizzo IP di R2.
8. Così via fino alla destinazione finale indicata nel comando traceroute.
 - La destinazione finale manderà al mittente iniziale un messaggio particolare ossia *ICMP Destination Unreachable*.
9. Il messaggio *ICMP Destination Unreachable* seguirà tutto il percorso all'incontrario fino al mittente iniziale.
10. Il mittente iniziale riceverà il pacchetto, vedrà "Destination Unreachable" e capirà che il pacchetto è arrivato alla fine.

		IP del primo hop	Round-trip time prima terna (TTL=1)		
> traceroute www.google.com					
		traceroute to www.google.com (216.58.205.196), 30 hops max, 60 byte packets			
1	131.114.175.73 (131.114.175.73)		0.348 ms	0.351 ms	0.378 ms
2	jser-jing.unipi.it (131.114.191.129)		0.363 ms	0.371 ms	0.398 ms
3	ru-unipi-rx1-pi1.pi1.garr.net (193.206.136.13)		1.302 ms	1.312 ms	1.321 ms
4	rx1-pi1-rx2-mi2.mi2.garr.net (90.147.80.210)		6.881 ms	6.899 ms	6.898 ms
5	72.14.214.105 (72.14.214.105)		7.023 ms	6.974 ms	6.986 ms
6	* * *				
7	216.239.42.27 (216.239.42.27)		7.020 ms	6.931 ms	6.997 ms
8	mil04s29-in-f4.1e100.net (216.58.205.196)		6.949 ms	7.090 ms	7.087 ms

Hop identifiers

Scaduto il timeout (default 5 sec): non è stato ricevuto il messaggio di risposta per alcun pacchetto della terna. Possibili cause:

- device non configurato per rispondere a traffico ICMP/UDP;
- pacchetti scartati per motivi di rete (es: traffico bloccato da un firewall).

Perché traceroute stampa gli asterischi?

Perché è scaduto un timeout.

Questo può accadere per vari motivi, magari perché quell'host non è configurato per rispondere.

Come faccio a capire se il motivo è la configurazione? Riprovo a mandarlo dopo un pò di tempo, sperando che l'algoritmo i route selezioni di nuovo lo stesso circuito.

Ma il comando traceroute va sempre avanti, come mai?

L'host 6 (nonostante non risponda) esegue il compito di inoltro verso l'host 7.

Non c'è alcuna certezza che i pacchetti seguano la stessa strada all'andata e al ritorno.

Il calcolo del Round-Trip-Time risulta quindi inaffidabile.

Server DHCP in Virtual Box

Voglio creare un clone della mia VM che al suo interno ha un processo server DHCP.

Di default virtualbox ha una gestione dinamica degli indirizzi IP , come se avesse un server DHCP al suo interno.

Programmazione Distribuita in C

"Una applicazione si dice distribuita se comprende più programmi in esecuzione su macchine diverse connesse in una rete".

Due processi possono essere:

- Indipendenti
- Cooperanti.

Su una o più macchine (sistemi distribuiti).

Metodi per Comunicare

- Memoria Condivisa.

- Metodo visto a calcolatori elettronici.
- **RPC - Chiamata a procedura remota.**
 - Come la CALL ad una stored procedure, la macchina con cui chiamo la call può essere tranquillamente diversa dalla macchina che ospita il database.
- **Scambio di Messaggi.**
 - Quello che vedremo nel corso di Reti Informatiche.

Sincronizzazione

Un altro problema che si ha quando due processi devono interagire è la **sincronizzazione**, i dati devono seguire un certo ordine di invio.

Per garantire la sincronizzazione useremo i semafori.

Un semaforo è una variabile che risiede nello spazio condiviso con cui i processi interagiscono tramite *wait()* e *signal()*, primitive di sistema fornite dal sistema operativo che i processi chiamano quando vogliono fare qualcosa sul semaforo.

L'instaurazione e lo scambio di messaggi li faremo tramite delle system call.

Paradigma Client-Server

Il server è in attesa di richieste da parte dei client, per accontentarle (se può). Si basa sulla comunicazione basata su scambio di messaggio.

La macchina client e la macchina server hanno entrambi la pila protocollare, l'unica differenza tra le due è il loro ruolo, ossia il tipo di programma che ospitano.

Socket

L'estremità del canale di comunicazione fra due processi in esecuzione su due macchine diverse connesse in rete.

Il socket è uno strumento fornito dal sistema operativo.

Il socket è un punto di accesso, da cui le applicazioni lato client possono accedere alle funzionalità di rete.

I socket sono molto utili perché separano le applicazioni da tutti i livelli sottostanti.

Tramite delle primitive fornite dal sistema operativo io posso:

1. Creare un socket.
2. Assegnargli un indirizzo.
 - a. Per fare in modo che un'applicazione possa identificarlo in rete in modo univoco.
3. Connetterlo con un altro socket..
4. Accettare una connessione.
5. Inviare/Ricevere dati attraverso il socket.

Processo Server

Programma in esecuzione sulla macchina server.

Ha lo scopo di offrire uno o più servizi ai client che si connettono.

Fa uso di system call per utilizzare i socket.

Primitiva `socket()`

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int domain , int type , int protocol);
```

Deve essere chiamata nel momento in cui devo creare un end-point, ossia un punto di connessione tra un processo e l'altro.

Di default un socket è Bloccante.

- Domain:
 - Famiglia di protocolli da usare.
 - AF_LOCAL → Protocolli per la Comunicazione Locale.
 - AF_INET → Protocolli IPv4, TCP/UDP (*Noi usiamo questo*).
- Type:
 - Tipologia di Socket.
 - SOCK_STREAM → Socket TCP.
 - SOCK_DGRAM → Socket UDP.
- Protocol → (*Noi lo mettiamo sempre a 0*).

La primitiva `socket()` restituisce un intero che rappresenta un “descrittore di socket” (il quale in realtà è un descrittore di file, poiché in UNIX “tutto è un file”) se tutto è andato bene. Restituisce -1 se c’è stato un qualsiasi errore.

Dopo la chiamata della `socket()`, non posso ancora usarlo, perché non ho ancora associato un indirizzo IP, né una porta e nel caso di un socket TCP, non ho neanche stabilito una connessione TCP con chi dovrebbe comunicare.

Strutture `sockaddr_in` e `in_addr`

```
struct sockaddr_in{
    sa_family_t sin_family; → Famiglia di Protocolli usati dal Socket.
    in_port_t sin_port; → Numero di Porta espresso in Network Order.
    struct in_addr sin_addr;
}

struct in_addr{
    uint32_t s_addr; → Indirizzo IP in formato Numerico su 32 Bit.
}
```

Tipo “Timbrato”

Tipi che hanno il nome seguito da “_t”.

Garantito in termini di dimensione, ossia di numero di byte necessari per memorizzarlo è lo stesso su tutte le architetture del mondo.

E' necessario perché questi dati sono scambiati tra le macchine su tutta la rete e non devono emergere ambiguità.

Inoltre tra architetture eterogenee, possiamo trovare differenze sul numero di bit tra stesso tipo (ad esempio int o float), ma anche possibili differenze nella rappresentazione dei bit (little endian o big endian).

I tipi timbrati servono proprio a uniformare la tipologia di dati che possiamo incontrare nella rete.

Endianness - “Ordine dei Byte”

Nella rete i dati transitano sempre in formato Big Endian.

Mentre nelle architetture dipende dall'architettura stessa.

In big endian, il primo byte (quello più a sinistra) è il più significativo, quindi se un tipo occupa 4 byte, nel byte di indirizzo minore ho il byte più significativo.

In little endian, il primo byte (quello più a sinistra) è il meno significativo.

I singoli bit all'interno dei byte hanno ancora l'ordinamento classico, con il MSB a sinistra.

In tipi di dato con meno di 2 byte (tipo caratteri di stringhe), non si può parlare di endianness.

Funzioni di Conversione - “Funzioni Ninja”

```
uint32_t htonl(uint32_t hostlong); "Host to Network Long"  
uint16_t htons(uint16_t hostshort); "Host to Network Short"  
uint32_t ntohl(uint32_t netlong); "Network to Host Long"  
uint16_t ntohs(uint16_t netshort); "Network to Host Short"
```

Servono per convertire un dato da Host Order a Network Order e viceversa.

Formato degli Indirizzi

Gli indirizzi IP per essere usati devono prima essere convertiti in formato numerico.

```
inet inet_pton (int af , const char* src , void* dst)  
pton → Presentation(x.x.x.x) to Numeric(10101010...010011001) .
```

af → Famiglia (**AF_INET**)

src → Indirizzo espresso in not dec punt. (x.x.x.x)

dst → Puntatore ad un istanza di struttura **in_addr**.

```
const char* inet_ntop(int af , const void* src , char* dst ,  
socklen_t size)  
ntop → Numeric(10101010...010011001) to Presentation(x.x.x.x) .
```

af → Famiglia (**AF_INET**).

src → puntatore ad una istanza di struttura **in_addr**.

`dst` → Puntatore ad buffer di caratteri di lunghezza `size`.

`size` → Deve valere almeno `INET_ADDRSTRLEN` (*16 Byte*).

Sockets e Sistema Operativo

I sockets sono gestiti dal sistema operativo.

Il processo per creare e usare i socket userà le system call fornite dal sistema operativo, non andrà mai a *"metterci le mani"* direttamente.



Ching Cheng Anji Chin Chan Chii e Noooo Chi en sha Ne ro Shan Zu

I Socket e il Paradigma Client-Server

Il processo Server necessita le seguenti primitive per fare il suo lavoro.

- `socket()`
- `bind()`
- `listen()`
- `accept()`

Il processo Client necessita le seguenti primitive per fare il suo lavoro.

- `socket()`
- `connect()`

Dopo aver definito i socket e aver instaurato la connessione, è possibile fare lo scambio di messaggi attraverso:

- `send()`
- `recv()`

Per ora si parla di socket bloccanti e che usano il protocollo TCP.

Tipologie di Processi Server

Un server iterativo gestisce una richiesta alla volta, i server che gestiscono più richieste contemporaneamente sono detti *“concorrenti”*.

Per questi ultimi ci sono vari modi per realizzarli.

Primitiva Bind()

Primitiva di “Aggancio”.

La primitiva permette di associare ad un socket vuoto un indirizzo IP e una porta.

Associa al Socket la coppia < Indirizzo IP , Porta >.

Restituisce -1 se fallisce o 0 se ha successo.

Il Client di solito non eseguire la `bind()`, perché solitamente il Client non ha bisogno di pubblicizzare il suo indirizzo IP, perché nel Paradigma Client-Server è il client a fare la prima mossa..

Se il programmatore lo desidera può fare comunque la `bind()`.

Il Sistema operativo eseguirà la bind sui socket del client autonomamente attraverso le “*porte effimere*” (ossia che durano poco).

Per il server è obbligatorio usare la `Bind()`.

```
int bind(int sockfd , const struct sockaddr* addr , socklen_t  
addr);
```

`sockfd` → Identificatore del Socket, restituito dalla primitiva `socket()`.

`addr` → Puntatore ad una struttura `sockaddr`.

Noi usiamo `sockaddr_in` quindi per questo motivo abbiamo bisogno di un un `cast` del puntatore.

`addrlen` → Dimensione di `addr`.

```
ret = bind( sd , (struct sockaddr*)&myaddr , sizeof(my_addr));
```

Ora il socket può essere messo in ascolto di richieste di connessione da parte dei client.

In `htons` ho la porta del processo server, la inserisce il programmatore.

Primitiva Listen()

Non è Bloccante.

La `listen` mette il socket in uno stato di ascolto in modo da poter ricevere le richieste di connessione da parte dei client.

Alla chiamata di `listen`, il socket viene messo in ascolto e viene creata una coda di attesa (con un limite massimo di posti, specificato nella `listen` stessa).

Il socket può ricevere le richieste, ma non le può servire fino a che non viene chiamata la `accept()`, la quale viene immediatamente dopo nel codice.

Se la coda si riempie, (se il tasso di arrivo è maggiore del tasso di servizio) le richieste oltre il limite vengono scartate (*tail drop*).

La dimensione non deve essere né troppo piccola, né troppo grande (sennò le richieste in fondo devono aspettare un sacco di tempo).

```
int listen(int sockfd , int backlog);  
sockfd → Descrittore del Socket.  
backlog → Dimensione della coda di Ascolto.
```

Primitiva Accept

Primitiva bloccante.

Se il server chiama la `accept` con la coda vuota, il processo server va in attesa e si sbloccherà all'arrivo di una richiesta.

```
int accept(int sockfd , struct sockaddr* addr , socklen_t* addrlen)
```

La chiamata della `accept` comporta l'accettazione di una richiesta di connessione TCP.

Dunque alla creazione di un Socket di Comunicazione.

Quando il socket di comunicazione viene stabilito, il server riparte da dopo la `accept`.

Restituisce il Socket ID del socket di Comunicazione.

Socket di Comunicazione - “Socket Connesso”

Questo socket verrà usato per l'effettivo scambio di messaggi tra il client e il server.

Supponiamo di avere 3 richieste in attesa nella coda delle richieste.

Il socket di ascolto rimane in ascolto e rimane riservato all'ascolto di richieste di connessione in arrivo.

Nel momento della `accept` viene creato un socket di comunicazione, il quale serve solo ad inviare e ricevere gli effettivi dati che si scambiano i processi.

Quando la richiesta è accettata, il processo Client comunica con il processo server tramite il socket di comunicazione.

Il socket di comunicazione dopo che la richiesta è stata soddisfatta può essere distrutto.

Creazione Socket Lato Server

```
int ret, sd, new_sd, len;  
struct sockaddr_in my_addr, client_addr; // Due strutture  
  
sd = socket(AF_INET, SOCK_STREAM, 0);  
// Creazione del Socket.  
  
memset(&my_addr, 0, sizeof(my_addr));  
// Pulizia della zona di memoria  
  
my_addr.sin_family = AF_INET;  
// Specifico la famiglia.  
  
my_addr.sin_port = htons(4242);  
// La porta del server in Network Order.  
  
inet_nton(AF_INET, "192.168.4.5", &my_addr.sin_addr);
```

```

// Inserisco nel campo sin_addr l'indirizzo IP in formato
Numerico.

ret = bind(sd, (struct sockaddr*)&my_addr, sizeof(my_addr));
// Aggancio il socket ai parametri scritti in my_addr

ret = listen(sd, 10);
// Metto il socket in ascolto con una coda di 10 posizioni.

len = sizeof(client_addr);
// Lunghezza della struttura che contiene l'indirizzo del Client.

new_sd = accept(sd, (struct sockaddr*)&client_addr, &len);
// Accetto la richiesta di connessione del client.

```

Primitiva Connect()

Primitiva Bloccante.

Se la coda del processo Server non è vuota, allora il processo Client si sospende alla connect.

```
int connect(int sockfd , const struct sockaddr* addr , socklen_t
addrulen)
```

sockfd → Descrittore del socket locale.

addr → Puntatore alla struttura contenente l'indirizzo del socket remoto.

addrulen → Dimensione di *addr*.

La `connect` restituisce 0 se ha successo o -1 se fallisce.

Creazione Socket Lato Client

```

int ret, sd;
struct sockaddr_in server_addr;
// Struttura dati per contenere le info del Server.

sd = socket(AF_INET, SOCK_STREAM, 0);
// Creazione del Socket.

memset(&server_addr, 0, sizeof(server_addr));
// Pulizia della zona di memoria.

server_addr.sin_family = AF_INET;
// Famiglia dell'indirizzo del Server.

server_addr.sin_port = htons(4242);
// Porta del Server.

inet_nton(AF_INET, "192.168.4.5", &server_addr.sin_addr);

```

```

// Inserisco nel campo sin_addr l'indirizzo IP del Server in
formato Numerico.

ret = connect(sd, (struct sockaddr*)&server_addr,
sizeof(server_addr));
// Invio una richiesta di connessione TCP al server.

if(ret < 0){
    // Socket Non Connesso.
}else{
    // Socket Connesso.
}

```

Scambio di Dati

La più grande incognita nella comunicazione di rete che nessuno degli *end-point* (processi comunicanti) può dare per scontato.

Il Formato del Messaggio, server e client devono essere d'accordo sul formato del messaggio prima di avviare la comunicazione.

Per fare ciò usiamo un protocollo che definisce il formato dei messaggi, ossia la dimensione e come sono organizzati.

Primitiva send()

Primitiva che invia un messaggio attraverso un socket TCP connesso (il socket di comunicazione, creato a seguito della accept() dal server, in questo socket fluiranno effettivamente i dati).

La primitiva è bloccante, il processo rimane bloccato finché non ha scritto il messaggio (o parte di esso) del buffer di invio del kernel.

Restituisce il numero di Byte scritti o -1 se fallisce.

```
int send(int sockfd , const void* buf , size_t len , int flags)
```

sockfd è il socket di comunicazione.

buf è un puntatore al buffer con il messaggio da inviare.

ossia un'area di memoria in cui il programmatore salva il messaggio da inviare nel suo formato definitivo.

len è la lunghezza in byte del messaggio nel suo formato definitivo.

flags esprime eventuali opzioni, noi per ora li mettiamo tutti a 0.

Supponiamo che il messaggio sia 1024 byte.

Se mi restituisce 1024 allora la send ha copiato con successo i 1024 byte contenuti nell'area di memoria nel buffer di invio del kernel.

Perché il kernel (dopo che si ha creato un socket) crea un buffer di invio e un buffer di ricezione per il socket di comunicazione.

Se restituisce un numero < 1024 , allora sono stati copiati solo parte dei byte nel buffer.
Questo succede quando il buffer è pieno, questo può succedere quando.

- Il buffer è condiviso e c'è un altro processo che sta inviando qualcosa mediante lo stesso socket.
- Se il buffer è privato , perché il rateo di invio è troppo basso e il buffer non riesce a svuotarsi.

In questo caso, il programmatore scorre il puntatore del numero di byte effettivamente inviati e richiama la send , quindi spezza in più parti l'invio del messaggio.

Quindi ad una sola chiamata di send non si presuppone una sola chiamata di recv.

Infatti vengono messe in un loop dove ci si resta finché tutti i byte non sono stati spediti/ricevuti.

Primitiva recv()

Riceve un messaggio dal socket di comunicazione.

Primitiva bloccante, e rimane bloccata fino a che non legge almeno un byte dal buffer di ricezione del kernel.

Restituisce 0 solo se l'altra parte chiude la connessione TCP.

Restituisce il numero di byte ricevuti, ossia il numero di byte che sono stati copiati con successo dal buffer di ricezione del kernel nell'area puntata da buf.

Se metto Flags a MSG_WAITTALL, ho la certezza che al termine della recv, il processo che si sblocca ha ricevuto tutti i byte del messaggio.

close()

Chiude il socket.

errno

Mi aiuta a capire la causa del -1, ossia mi dice il codice dell'errore

se la bind restituisce -1:

- Se errno = EADRINUSE → Address already in use, ossia sto cercando di agganciare un indirizzo IP già occupato.
 - Spesso accade quando definisco 2 socket con lo stesso indirizzo ,devo dare tempo al sistema operativo di evacuare quell'indirizzo IP.
- Se errno = EINVAL → Invalid
 - Quando l'indirizzo non è valido.

perror()

perror("Error: "); stampa la motivazione dell'errore.

Protocollo Text

Un protocollo text è “orientato al testo”, ossia prende il testo, lo converte in una stringa e lo invia.

La stringa in questione è in un formato concordato tra mittente e destinatario.

Il destinatario riceve la stringa e la salva in memoria fino alla marca di fine stringa.

Quando invio una stringa devo per forza inviare anche il carattere di fine stringa?

Dipende da come si sono messi d'accordo destinatario e mittente, se i messaggi hanno lunghezza variabile serve, se i messaggi hanno lunghezza fissa invece non è necessario.

Protocollo Binary

Un protocollo binary invia i dati binari (grezzi, così come sono rappresentati nella macchina e quindi senza convertirli in stringa).

Utile quando voglio inviare delle strutture dati.

Quale usare? Dipende dalla natura dell'applicazione.

Se devo comunicare numeri grossi conviene il binary.

Anche l'ingombro dei dati è importante, e va minimizzato.

Un dato quanto occupa con un protocollo text?

Dipende dal valore scritto in quell'intero.

Se volessi inviare il numero intero 10:

- Binary → 4 Byte, ossia la grandezza di un numero Intero.
- Text → 2 Byte, perché uso la codifica ASCII, quindi 2 caratteri.

binary	byte 0	byte 1	byte 2	byte 3
1234	00000000	00000000	00000100	11010010

Text (ASCII)	byte 0	byte 1	byte 2	byte 3
"1234"	011 0001	011 0010	011 0011	011 0100

Quando voglio inviare una struttura dati io non posso inviarla tutta intera, ma bensì inviare singolarmente ogni attributo e lato receiver ricostruire la struttura dati pezzo per pezzo.

Protocollo Binary - Send

```
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);
```

Protocollo Binary - Recv

```
struct temp t;
...
ret = recv(new_sd, (void *)&t.a, sizeof(uint32_t), 0);

if (ret < sizeof(uint32_t)) {
    // Gestione errore
}

t.a = ntohs(t.a); // Convertire in host order il campo 'a'

ret = recv(new_sd, (void *)&t.b, sizeof(uint8_t), 0);

if (ret < sizeof(uint8_t)) {
    // Gestione errore
}
```

Serializzazione e Deserializzazione

I protocolli di tipo Binary hanno il problema della endianness.

Modo per rendere una struttura dati seriale, ovvero riuscire a farla transitare come flusso di bit e permettere al ricevente di ricostruirla e riallocarla.

I caratteri non hanno bisogno di `htonl`, perché essendo un unico Byte la endianness non ha senso per loro.

Problemi della Fork

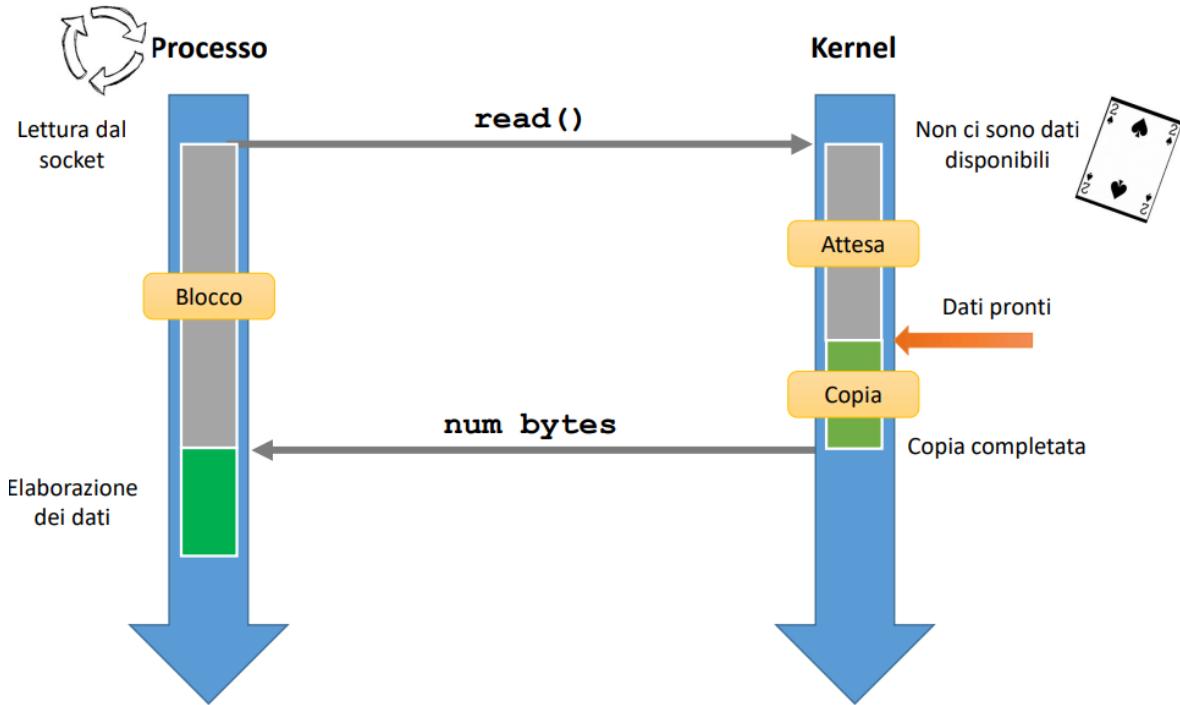
Molto lenta, perché deve duplicare tutte le risorse di un processo.

Si è molto deboli ad attacchi DoS, eseguire tante fork una dopo l'altra fa schiantare il processo Server.

Ci sono altri metodi oltre alla fork per fare un server concorrente?

Socket Bloccanti

Se eseguo la `recv` e non ci sono dati pronti nel buffer del kernel → Il processo si sospende.



Socket Non Bloccanti

Voglio aumentare le prestazioni della nostra applicazione di rete.

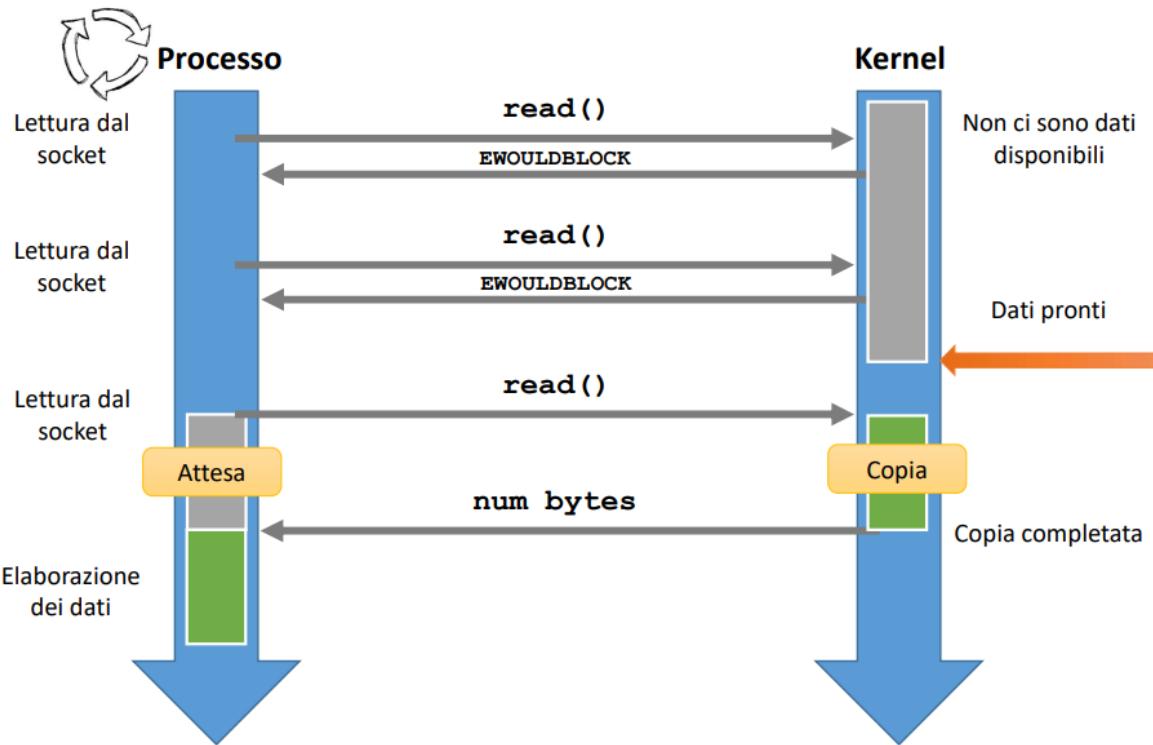
Un socket bloccante (*blocking socket*) è tale perché: `accept`, `connect`, `send` e `recv` sono tutte primitive bloccanti e potrebbero sospendere il processo chiamante.

Posso rendere il socket disponibile anche durante il tempo di attesa?

Inventiamo dei socket che non si bloccano su queste primitive.

Quando il processo esegue una di queste primitive non rischia di essere sospeso.

Il processo in attesa può fare altro.



In un socket non bloccante, l'unico tempo di attesa che devo per forza subire è quello dovuto alla copia dei dati dal buffer del kernel al buffer indicato dall'utente.

```
int sd = socket(AF_INET, SOCK_STREAM | SOCK_NONBLOCK, 0);
```

connect → Restituisce -1 e imposta errno a EINPROGRESS se non può connettersi.

accept → Se non ci sono richieste restituisce -1 e imposta errno = EWOULDBLOCK.

send → Restituisce -1 se non può inviare tutto il messaggio e imposta errno = EWOULDBLOCK.

recv → Restituisce -1 se non ci sono messaggi disponibili e imposta errno = EWOULDBLOCK.

EWOULDBLOCK → “Deh E Uou! Blocche” ~ Sohet Pisani

I/O Multiplexing - Primitiva Select

Consentono a un processo di monitorare più descrittori di file, in attesa che uno o più descrittori di file diventino "pronti" per qualche classe di operazioni di I/O (ad esempio, input possibile).

Un socket diventa “pronto” quando nel caso Bloccante si sbloccherebbe.

Un Socket è pronto in lettura :

- Se ho almeno un byte da leggere.
- Quando viene chiuso, quindi la read restituirebbe 0.
- In caso di errore, la read restituirebbe -1.
- E' in ascolto e ci sono connessioni effettuate.

Un Socket è pronto in Scrittura

- C'è spazio nel buffer per scrivere.
- C'è un errore, la write restituirebbe -1.

Insieme di Descrittori di Socket

E' un intero che va da 0 a FD_SETSIZE, di solito pari a 1024.

Un set è un insieme di descrittori, ossia una variabile di tipo fd_set.

Ci inserisco i Socket che voglio "monitorare in lettura" e quelli da "monitorare in scrittura".

Per agire sul descrittore devo usare delle MACRO.

void FD_SET(int fd , fd_set* set) → Inserisco fd in fd_set.

int FD_ISSET(int fd , fd_set* set) → Verifica se fd è in fd_set.

void FD_CLR(int fd , fd_set* set) → Rimuove fd da fd_set.

void FD_ZERO(fd_set* set) → Svuota fd_set.

La select prende fd_set, ci accede in lettura e nell'insieme rimangono solo quelli pronti in lettura, quindi la select lascia in fd_set solo quelli pronti mentre gli altri li leva.

Lo standard input è un socket pure lui, se l'utente digita qualcosa allora il socket dello standard input diventerà pronto in lettura poiché ci sarà qualcosa da leggere nel socket.

Come seleziono un socket tra quelli pronti?

```
int select(
    • int nfds
        ○ Numero del descrittore più alto tra quelli da controllare.
    • fd_set* readfds
        ○ Lista di descrittori da controllare per la lettura.
    • fd_set* writefds
        ○ Lista di descrittori da controllare per la scrittura.
    • fd_set* exceptfds
        ○ Lista di descrittori da controllare per le eccezioni.
        ○ Non ci interessa.
    • struct timeval* timeout
        ○ Intervallo di timeout.
        ○ Se scade il timeout il processo chiamante interrompe l'attesa dovuta alla sospensione della select.
```

```

)
struct timeval{
    long tv_sec; // Secondi
    long tv_usec; // Microsecondi
}

```

La select è bloccante, finchè almeno uno dei descrittori non è pronto non può fare niente.
 La select restituisce un intero m ossia quanti descrittori pronti ha lasciato nel set , oppure -1.

1. Metto i descrittori nello *fd_set*.
2. *select()*.
3. Prendo i descrittori pronti.
4. Scorro i Descrittori rimasti nel set.
5. Normali operazioni con i socket.

Codice I/O Multiplexing

```

int fd_max = 0;
fd_set read_fs;
fd_set master;

FD_ZERO(&master); // Azzero il set master.
FD_ZERO(&read_fs); // Azzero il set copia.

FD_SET(request_socket, &master); // Inserisco nel set il socket di ascolto.
FD_SET(STANDARD_INPUT, &master); // Inserisco nel set lo standard Input.

if (request_socket > fd_max)
{
    fd_max = request_socket;
// fd_max contiene il valore massimo dei descrittori di socket di master.
}

while (1){
    read_fs = master;
// Salvo la lista di descrittori master in un'altra lista read_fs
// Poiché dopo la select perdo i descrittori non pronti e dovrei
// Inserirli di nuovo.

    select(fd_max + 1, &read_fs, NULL, NULL, NULL);
// Eseguo la select sul set dei socket pronti in lettura

```

```

        for (i = 0; i <= fd_max; i++){ // scorro tutti i socket in
master

            if (FD_ISSET(i, &read_fs)){// Il Socket i è pronto in
lettura?

                if (i == STANDARD_INPUT){ // Se il socket pronto è
stdin.

                    // Cose da fare con il stdin.

                    // Se questo socket risulta pronto in lettura
                    // significa che l'utente ha inserito
                    // qualcosa da tastiera
                }

                if (i == request_socket)// request_socket è pronto
in lettura?
                {
                    // E' il request_socket, ossia il socket di
ascolto.

                    len = sizeof(client_addr);

                    new_sd = accept( // Accetto la richiesta di
conessione
                        request_socket,
                        (struct sockaddr *)&client_addr,
                        (socklen_t *)&len
                    );

                    FD_SET(new_sd, &master);
                    // inserisco il socket di comunicazione
                    // nel Set di Socket da Monitorare in lettura

                    if (new_sd > fd_max){
                        fd_max = new_sd;
                    }
                    /*
                    GESTIONE RICHIESTA
                    */
                }
                // Socket pronto in lettura non è quello che mi
aspettavo
                FD_CLR(i, &master); // Lo rimuovo dal set
                close(i); // e lo chiudo.
            }
        }
    }
}

```

Socket UDP

Il protocollo UDP è *connectionless*, quindi un socket UDP non è mai connesso.

Inoltre UDP non prevede nessuna operazione di recupero dei pacchetti persi e nessuna operazione di riordino prima del demultiplexing.

Se volete per forza usare i Socket UDP, ma volete comunque queste garanzie, dovete implementare quelle cose lato applicazione.

Però notando il codice si vede che il client comunque effettua la `connect`, questa primitiva se fatta su un socket UDP non ha lo stesso effetto che in un socket TCP.

In un socket UDP la `connect` specifica solo l'indirizzo remoto a cui inviare i pacchetti, ma non viene stabilita una connessione, è solo un modo per non specificare ogni volta l'indirizzo di destinazione quando si chiamano le primitive di invio e ricezione.

Il Server e il Client devono fare solo la `bind`.

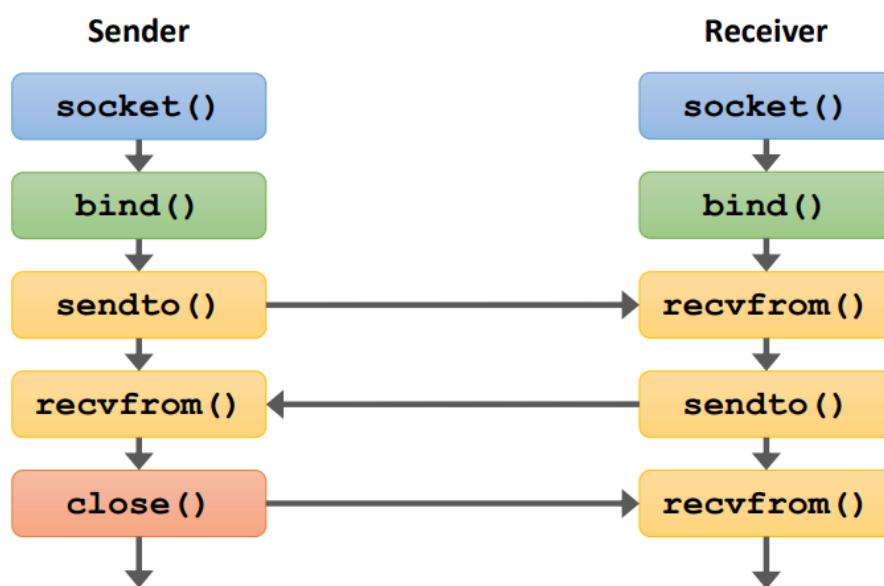
La `listen` e la `accept` servono solo per ascoltare e accettare le richieste di connessione TCP.

Dopo la `bind` un socket UDP è pronto all'uso.

Primitive per lo Scambio di Messaggi con Socket UDP

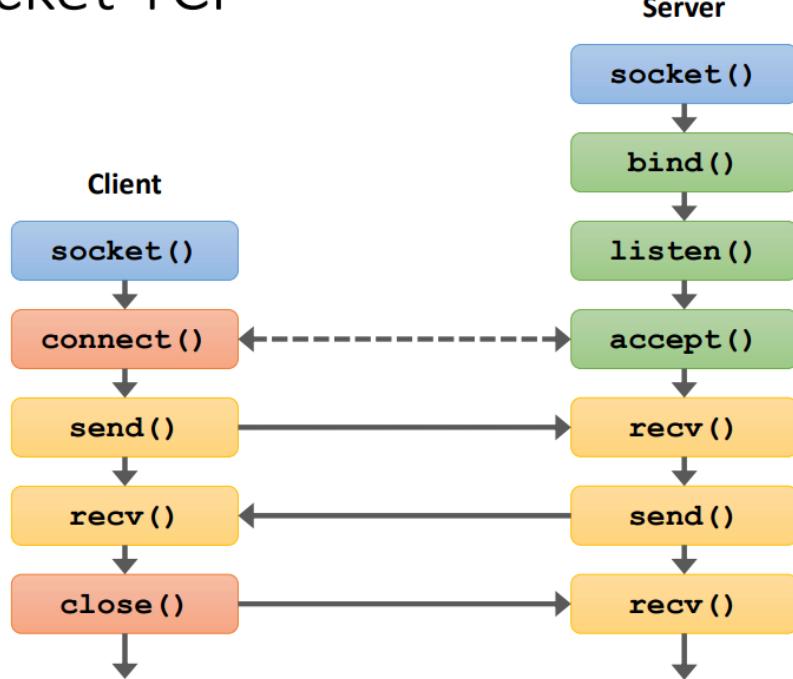
`sendto` → Invia un messaggio attraverso un socket UDP.

`recvfrom` → Riceve un messaggio attraverso un socket UDP.



Per fare un confronto, viene mostrata anche l'ordine delle primitive in un socket TCP.

Socket TCP



Firewall Application Layer

Operano a livello applicazione, facendo una *deep packet inspection*.

Molto sicuri ma allo stesso tempo computazionalmente onerosi.

Firewall Network Layer (Packet Filter)

Operano a livello TCP/IP, analizzando gli header IP e l'header TCP/UDP.

Si dividono in due categorie:

- **Stateless**
 - Ogni pacchetto è analizzato in base a campi statici dell'intestazione.
- **Stateful**
 - Oltre alla funzionalità stateless esso è in grado di tenere traccia delle connessioni TCP e scambi UDP attualmente in corso.
 - Quindi è in grado di fare una correlazione tra un pacchetto e l'altro.

Regole del Firewall

Una regola è composta da 2 parti:

- **Criteria** → Caratteristiche del pacchetto.
- **Target** → Azione da intraprendere per il pacchetto che soddisfa i *Criteria*.

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

Quando un pacchetto arriva dal firewall, esso scorre tutte le regole (in ordine crescente di indice) e quando ne trova una in cui i criteri corrispondono, applica quella sola regola, poi passa al prossimo pacchetto.

Detto ciò, l'ordine delle regole è fondamentale, è cosa buona e giusta mettere per prime le regole più specifiche e mettere a seguire quelle più generiche.

L'ultima regola è detta "regola di default", che viene applicata a tutti i pacchetti.

Netfilter

Netfilter è il componente del Kernel Linux che offre le funzionalità di:

- Stateless (o Stateful) Packet Filtering.
- NA[P]T
- Packet Mangling
 - Manipolazione generica dei pacchetti.

iptables

Programma (a linea di comando) per configurare le tabelle delle regole dei firewall.

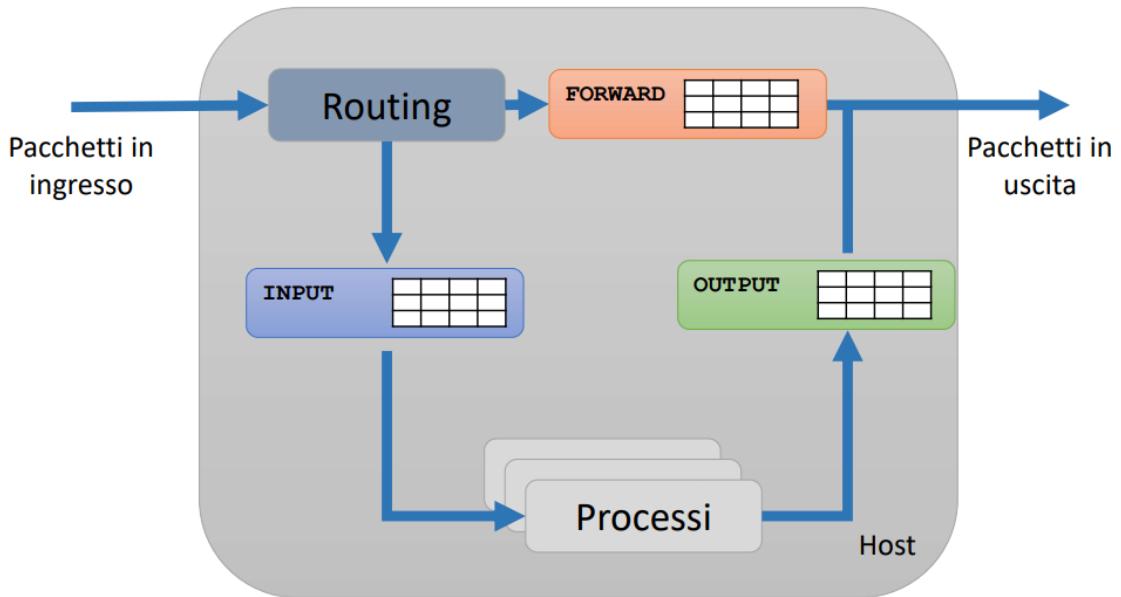
Lavora su più tabelle.

- **Ogni tabella è dedicata a una specifica funzionalità.**
 - Avremo una tabella per il Nat e una per il Firewall.
- **Ogni tabella contiene diverse chain.**
 - Le chain servono a suddividere una tabella in sezioni, questa cosa è molto utile, perché come vedremo dopo le chain si differenziano per il momento in cui vengono considerate.
- **Ogni chain contiene una lista di regole da applicare a una categoria di pacchetti.**

Tabella Filter (Firewall)

Ha 3 chain, ogni chain è considerata in base alla provenienza/destinazione del pacchetto.

- **Input**
 - Le regole di questa chain vengono considerate per i pacchetti in ingresso destinati ai processi locali della macchina.
- **Output**
 - Le regole di questa chain vengono considerate per i pacchetti in uscita dai processi locali della macchina.
- **Forward**
 - Le regole di questa chain vengono considerate per i pacchetti in transito, ovvero da inoltrare ad altri host.



iptables - Comandi

Visualizzare le regole → `iptables [-t table] -L [chain]`

- Se la tabella non è specificata → Viene selezionata filter.
- Se la chain non è specificata → Vengono elencate tutte le chain della tabella `table`.

Aggiungere una regola in fondo alla chain:

`iptables [-t table] -A chain rule-specification`

Aggiungere una regola in una posizione specifica della chain:

`iptables [-t table] -I chain [num] rule-specification`

- Se `num` è omesso → Mette la regola nella 1° posizione della chain.

Per rimuovere una regola da una chain:

`iptables [-t table] -D chain rule-specification`

Per rimuovere tutte le regole da una (o più) chain:

`iptables [-t table] -F chain`

Per cambiare la regola di default (policy) DROP/ACCEPT:

`iptables [-t table] -p target`

iptables - rule-specification

E' una stringa che serve a specificare la regola da inserire.

Opzione	Descrizione
<code>-p <protocollo></code>	protocollo (TCP, UDP, ICMP, ...)
<code>-s <address></code>	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

Le regole non vengono salvate e allo spegnimento si perdono le regole.

Per salvare le regole in un file.

```
iptables-save > file
```

Per caricare le regole da un file.

```
iptables-restore < file
```

NAT e PAT/NAPT

Può essere eseguito dal Router o dal Firewall.

Nasce come metodo per risparmiare indirizzi IP di versione 4.

Ora il NAT ha anche altre funzioni, soprattutto legate alla sicurezza.

Ha il compito di convertire tutti gli indirizzi IP privati dei client appartenenti ad una rete privata in un' unico indirizzo IP pubblico.

All'interno della rete privata si useranno indirizzi IP “privati”, validi solo all'interno della rete privata, all'esterno gli indirizzi IP privati (di sicuro) non sono unici nel mondo e quindi non validi.

L'unico indirizzo IP valido è quello del router di frontiera della rete privata, l'unico indirizzo IP effettivamente assegnato dallo ISP.

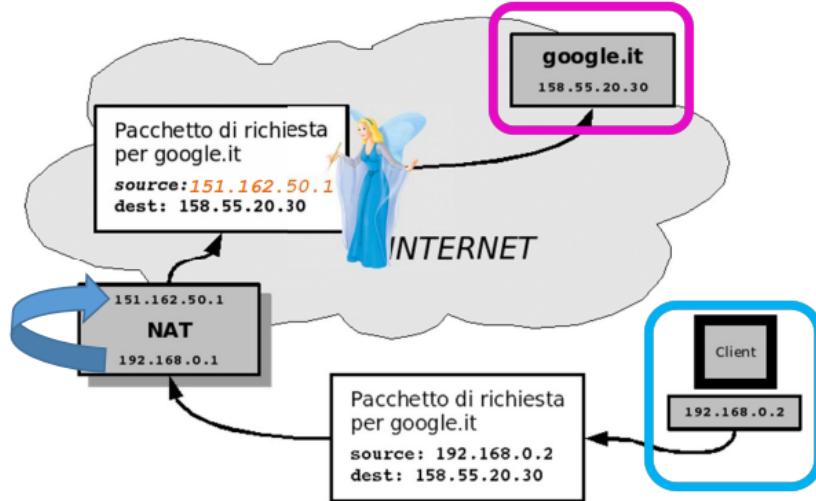
Gli host privati della rete privata useranno quell'unico indirizzo IP per navigare verso l'esterno.

Il risultato è che dall'esterno gli indirizzi IP privati degli Host sono “nascosti”, vedranno solo l'indirizzo IP pubblico della rete privata.

Uscita - Source NAT

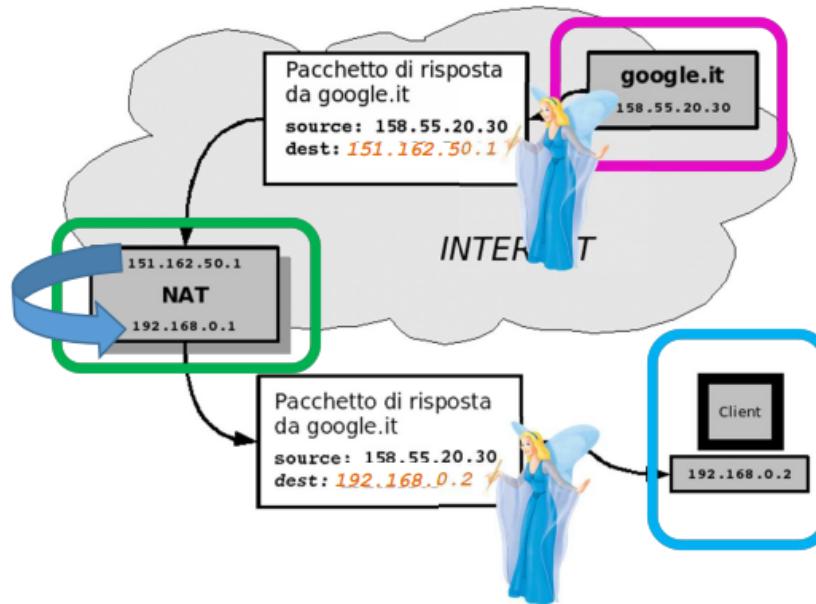
Le connessioni effettuate da un Host interno della rete privata sono alterate per mostrare all'esterno un indirizzo IP diverso da quello originale.

Chi riceve le connessioni le vede provenire da un IP diverso da quello utilizzato da chi le genera.



Ingresso - Destination NAT

Le connessioni effettuate da Host esterni sono alterate in modo da essere rediritte verso indirizzi IP diversi da quelli originariamente specificati nel datagram originale.



Port Address Translation (PAT)

Si tratta di una estensione di NAT, il quale ha a che fare solo con gli indirizzi IP. Con il NAT appena definito questa cosa funziona solo se gli host della rete privata effettuano richieste verso l'esterno uno alla volta. Come posso fare in modo che NAT funzioni per più Host privati contemporaneamente?

PAT Mappa più host della rete privata in un solo indirizzo IP.

Con PAT un pacchetto proveniente dall'esterno avente una porta di destinazione (detta *porta esterna*) è tradotto in un pacchetto avente una porta differente (detta *porta interna*).

Lo ISP assegna un indirizzo IP (unico nel mondo) al router di frontiera della rete privata.

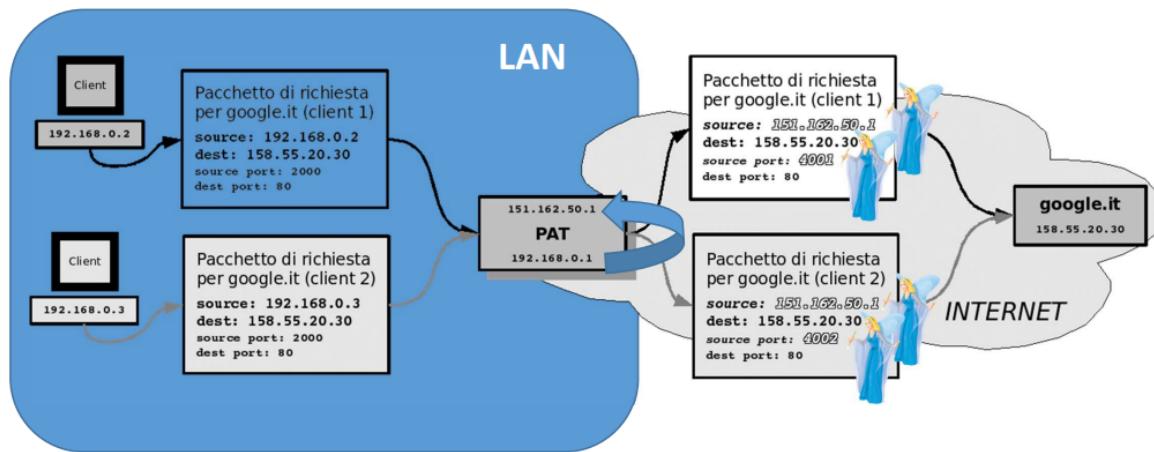
Il Router PAT assegna a questo Host Interno un numero di porta che viene accoppiato all'indirizzo IP privato dell'Host Interno.

Il Router PAT, per ogni Host interno della rete privata, terrà la coppia:
<Indirizzo IP Privato, Porta Esterna>

La coppia in questione serve solo al router per identificare in modo univoco un Host nella rete privata.

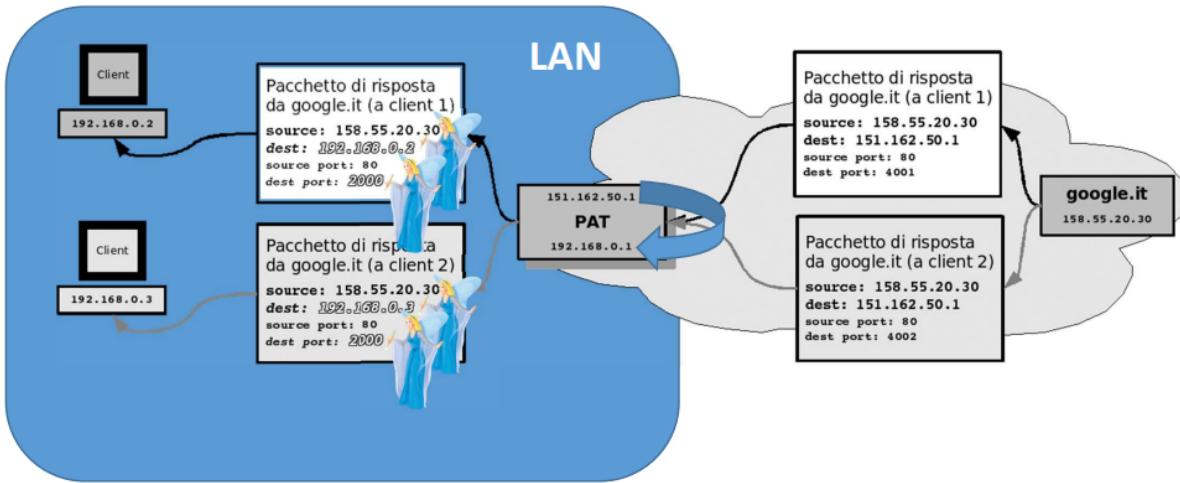
Pacchetto di Richiesta NATP

1. L'Host interno intende inviare una richiesta ad un Server esterno alla rete.
2. Quindi invia il pacchetto verso il default gateway, ossia il Router PAT.
3. Il Router PAT sostituisce : < Indirizzo IP Privato , Porta Sorgente >
Con la coppia: < Indirizzo IP Pubblico , Porta Esterna >
4. Infine il Router invia il pacchetto verso l'esterno



Pacchetto di Risposta NATP

1. La risposta del Server sarà diretta verso l'indirizzo IP Esterno e verso la Porta Esterna.
2. Il Router PAT sostituisce : < Indirizzo IP Pubblico , Porta Esterna >
Con la coppia: < Indirizzo IP Privato , Porta Sorgente >
3. Infine il Router immette il pacchetto nella rete interna.



iptables - Tabella NAT

Visualizzare le tabelle NAT → `iptables -t nat -L`

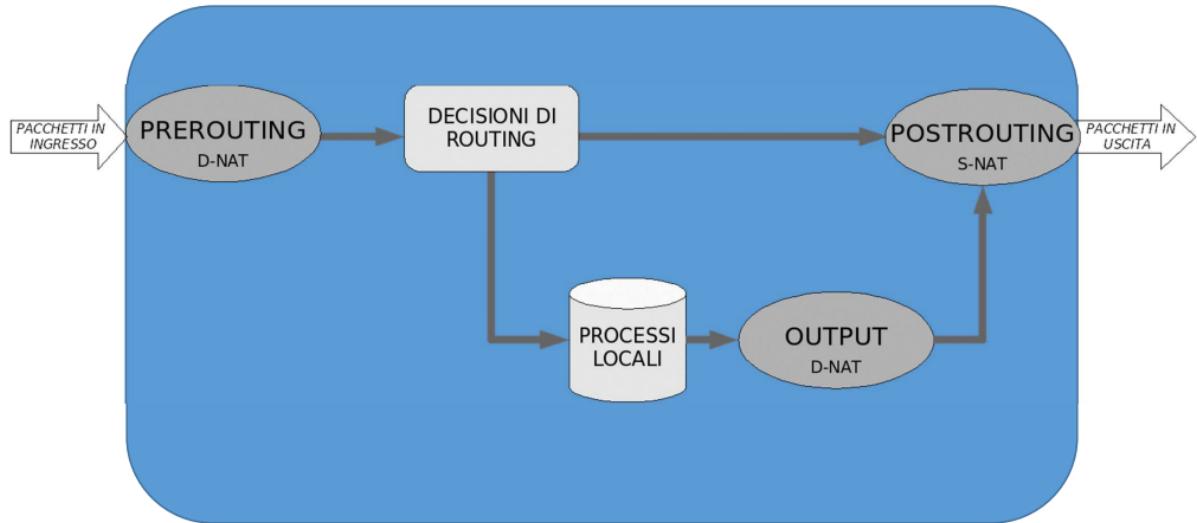
Ripulire le tabelle NAT → `iptables -t nat -F`

`iptables` gestisce il NA[P]T nella tabella NAT.

La tabella di NAT è usata per modificare gli Indirizzi IP e i numeri di porta sorgenti e destinazione.

La tabella NAT ha 3 catene:

- **PREROUTING:**
 - Fa Destination NAT (D-NAT), cioè altera indirizzo/porta di destinazione dei pacchetti.
 - Responsabile dei pacchetti appena arrivati all'interfaccia di rete, la router decision (ossia dove deve essere spedito il pacchetto) non è ancora stata presa.
- **OUTPUT:**
 - Fa Destination NAT (D-NAT) dei pacchetti destinati all'esterno e uscenti dai processi locali della macchina.
 - Essi poi passeranno anche dalla catena di POSTROUTING.
- **POSTROUTING:**
 - Fa Source NAT (S-NAT), cioè altera indirizzo/porta sorgente dei pacchetti.
 - Responsabile dei pacchetti uscenti da una interfaccia di rete, applicata dopo la decisione di routing.



S-NAT (pacchetti in uscita)

Il Source NAT si ha quando si ha necessità di cambiare l'indirizzo IP sorgente di un pacchetto.

SNAT deve essere eseguito dopo l'algoritmo di routing ma ovviamente prima di essere trasmesso.

Ciò significa inoltre che si potrà utilizzare l'opzione '-o' (interfaccia uscente).

Il Mascheramento degli indirizzi IP Privati è una applicazione di S-NAT dove tutti i pacchetti provenienti dai client interni alla rete privata vengono mascherati in un solo IP address pubblico.

Per tutti i pacchetti in uscita (-A POSTROUTING)

Provenienti dalla sorgente 192.168.0.2 (-s 192.168.0.0/24)

Imposta IP a 151.162.50.1 (--to-source 151.162.50.1)

con porte nel range [4001 ; 4100] (--to-source ... :4001-4100)

```

iptables
-t nat
-A POSTROUTING
-s 192.168.0.0/24
-j SNAT
--to-source 151.162.50.1:4001-4100

```

IMPORTANTE → Contro del NAT - Limite delle Porte Esterne

“4001-4100” è il range di porte riservate per identificare gli host connessi (ossia il range che le porte esterne devono avere).

Vuol dire che il numero di richieste diverse che gli host interni alla rete possono fare ha un limite.

Questo è uno dei limiti imposti dall'utilizzo del NAT.

Mascherare tutti i pacchetti in uscita. (-A POSTROUTING)

```
Dall'interfaccia di rete eth1. (-o eth1)
Con lo stesso indirizzo IP assegnato all'interfaccia eth1. (-j
MASQUERADE)
iptables
-t nat
-A POSTROUTING
-o eth1
-j MASQUERADE
```

D-NAT (pacchetti in ingresso)

Il Destination NAT o DNAT viene particolarmente utilizzato quando si hanno più server in DMZ con indirizzamento privato del tipo 192.168.0.x ed un solo server firewall dotato di IP pubblico è quindi visibile da Internet.

Ricordiamo sempre che il DNAT avviene nella chain di PREROUTING ossia prima dell'esecuzione dell'algoritmo di routing.

Il DNAT sarà utilissimo in quanto ci permette di proteggere i server pubblici in DMZ filtrando il traffico a loro destinato dal firewall che farà NAT ad esempio solo per i servizi HTTP e FTP.

Creare una DMZ con D-NAT

Reindirizzare (-A PREROUTING)

l'indirizzo di rete pubblica 80.23.45.178 (-d 80.23.45.178)

Con la Porta 80 (-dport 80)

**verso il server web 192.168.0.254. (-to-destination
192.168.0.254:80)**

iptables

-t nat

-A PREROUTING (*decisione prima del Routing*)

-p tcp (HTTP usa solo TCP, se non ci fosse questa regola, anche le richieste UDP con porta 80 verrebbero redirette)

-d 80.23.45.178 (IP Esterno della Nostra Rete)

-dport 80 (Richiesta HTTP, quindi vuole comunicare con il nostro Server WEB)

-j DNAT

-to-destination 192.168.0.254:80 (reindirizzo verso l'indirizzo IP privato del mio Server WEB)

Protocollo HTTP

HTTP è un protocollo di livello applicazione per lo scambio di ipermedia.

HTTP si Basa sul Paradigma Client Server

Il client (browser) chiede un documento e il server HTTP risponde inviandolo

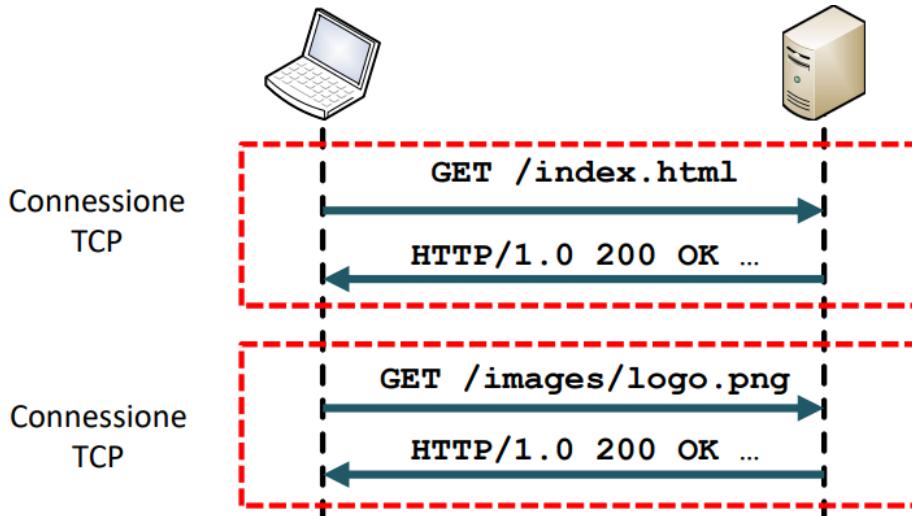
HTTP è Stateless

Il server non tiene traccia delle precedenti connessioni.

Ogni scambio richiesta/risposta è indipendente dai precedenti

HTTP ha comunicazioni non persistenti

Viene instaurata una connessione TCP per ogni richiesta del client.



HTTP Apache

Per avviarlo → # apache2ctl <comando>

Comandi principali: start, stop, restart, status, configtest.

Apache può accettare e servire più richieste contemporaneamente.

La configurazione si trova in /etc/apache2/apache2.conf

La configurazione avviene tramite direttive, eventualmente raggruppate in direttive contenitore.

```
# Esempio di commento
Direttiva1 valore
Direttiva2 valore

# Inizio contenitore
<Contenitore valore>
    Direttiva3 valore
    Direttiva4 valore
</Contenitore>
# Fine contenitore
```

La configurazione di Apache in Debian non è tutta contenuta in un unico file.

Tramite la direttiva "include" il file principale include altri file più piccoli, ognuno con le sue direttive.

Ad esempio il file /etc/apache2/ports.conf contiene le direttive per specificare le porte da usare.

Questi file vengono chiamati "Parti di Configurazione".

```
# File apache2.conf  
...  
Include ports.conf  
...
```

```
# File ports.conf  
Listen 80
```

Le parti di configurazione vengono messe nella directory /etc/apache2/conf-available.

I file di configurazione devono essere abilitati:

```
# a2enconf <nome_file>
```

Questo comando crea un soft link nella directory /etc/apache2/conf-enabled.

Il file di configurazione principale è impostato per includere tutti i file di configurazione contenuti conf-enabled.

Bisogna riavviare il server per ricaricare la configurazione.

Un file si disabilita con: # a2disconf <nome_file>

ServerRoot

La direttiva ServerRoot specifica la directory principale dei file di configurazione di Apache.

```
#ServerRoot /etc/apache2
```

I path relativi specificati nelle altre direttive sono risolti partendo da questa directory.

Su Debian, la direttiva ServerRoot è configurata automaticamente all'avvio del servizio dal comando apache2ctl.

KeepAlive

La direttiva KeepAlive specifica se offrire o meno le connessioni persistenti di HTTP 1.1.

```
KeepAlive on
```

KeepAliveTimeout

La direttiva KeepAliveTimeout specifica quanti secondi attendere la successiva richiesta dal client, su una stessa connessione, prima di chiuderla.

Specifica quanto tempo il server può restare fermo ad aspettare il client.

Valori troppo elevati potrebbero bloccare inutilmente un processo che sta servendo un client lento o disconnesso.

```
KeepAliveTimeout 5
```

Listen

La direttiva Listen specifica le porte su cui Apache si mette in ascolto di connessioni.

È obbligatoria, se la direttiva non c'è il server non parte

È possibile specificare più porte

```
Listen 80
```

```
Listen 8080
```

ErrorLog

La direttiva ErrorLog specifica il file di log degli errori.

```
ErrorLog /var/log/apache2/error.log
```

Formato e livello di dettaglio dei messaggi d'errore possono essere rispettivamente definiti con ErrorLogFormat e LogLevel.

Moduli

Parti di configurazione che forniscono funzionalità complesse.

Si trovano nella directory: /etc/apache2/mods-available

Essi devono essere abilitati, i moduli abilitati hanno un soft link in mods-enabled.

Dopo aver abilitato un modulo → Riavvio.

Direttive per Siti Web

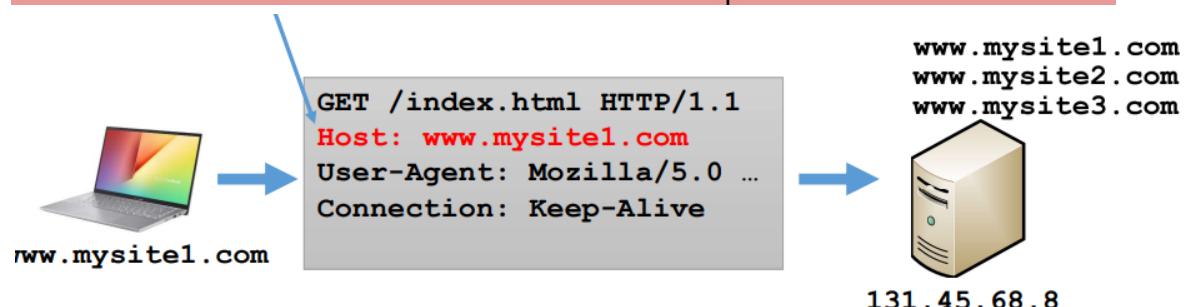
Nel caso più semplice, chiamiamolo 1-1-1-1, un server Web è in esecuzione su una macchina con un indirizzo IP, e ospita un solo sito Web.

Nel caso più semplice, chiamiamolo 1-1-1-1, un server Web è in esecuzione su una macchina con un indirizzo IP, e ospita un solo sito Web.

Virtual Host

Con il (name based) Virtual Hosting, si possono configurare più siti sullo stesso server Web, sulla stessa macchina, con lo stesso indirizzo IP.

Il server discrimina le richieste dei client in base al campo 'Host' della richiesta HTTP.



Come accade per parti di configurazione e moduli, i siti (detti virtual host) sono in una directory dedicata: /etc/apache2/sites-available

Si abilitano e disabilitano con:

```
# a2ensite <nome_file>
# a2dissite <nome_file>
```

Un sito abilitato ha un soft link in sites-enabled.

Bisogna riavviare il server per ricaricare la configurazione.

Apache ha un default Virtual Host Abilitato in:

/etc/apache2/sites-available/000-default.conf

```
<VirtualHost *:80>
    ServerName www.example.com
    ...
    DocumentRoot /var/www/html
</VirtualHost>
```

Direttiva Virtual Host

Permette di definire un Virtual Host, è una direttiva contenitore.

Necessita di un Indirizzo IP e di una Porta.

```
<VirtualHost ip:porta>
    ...
</VirtualHost>
```

ServerName

Direttiva che serve a specificare il nome simbolico del sito.

ServerName www.example.com

È indispensabile per discriminare le richieste fatte dai client ai vari virtual host ‘hostati’ dal server, i quali hanno lo stesso Indirizzo IP (quello del Server) e lo stesso numero di porta (80).

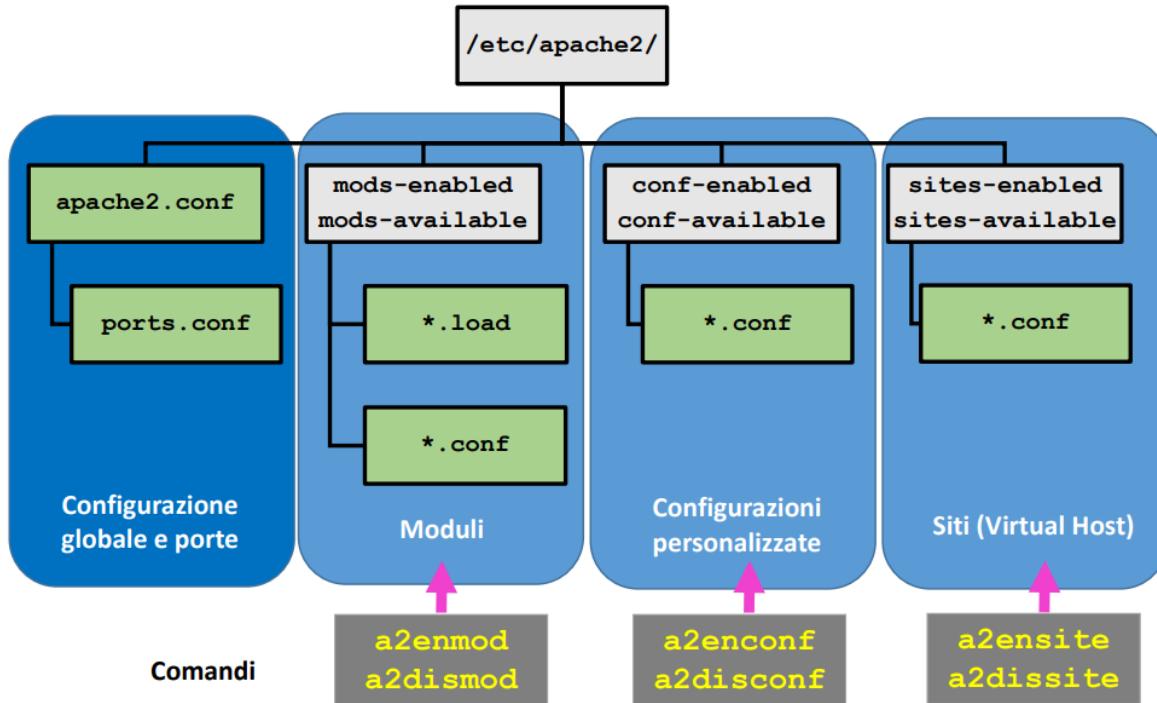
DocumentRoot

Specifica la directory dei file del sito, quelli accessibili ai client.

Diversamente da ServerRoot il cui contenuto non è accessibile ai client.

DocumentRoot /var/www/html

Sommario



Moduli Multi Processo

Come fa apache ad accettare e servire più richieste contemporaneamente?

Lo fa tramite i moduli multi-processing (MPM).

- gestione dei socket.
- binding delle porte
- processazione delle richieste usando processi figli e thread.

In UNIX abbiamo 3 MPM:

- prefork
- worker
- event

Modulo Prefork

Implementa un server multiprocesso, senza thread.

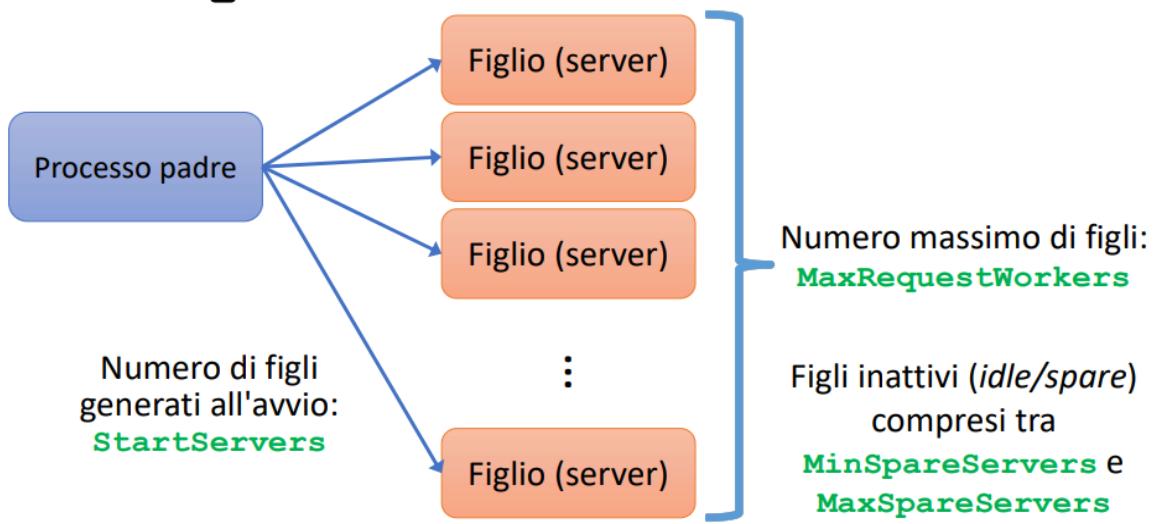
All'avvio, un processo padre lancia un certo numero di processi figli (*preforking*).

I figli (worker o server) restano in ascolto, accettano connessioni e le servono.

Dopo aver servito una connessione, il worker torna disponibile.

Il padre gestisce smista le richieste ai figli, cercando di mantenerne sempre alcuni disponibili.

Il preforking all'avvio e il riuso dei vari worker evitano l'overhead della `fork()` a ogni connessione.



Ogni figlio viene riciclato per `MaxConnectionsPerChild` connessioni poi viene terminato, per evitare *memory leak* accidentali

Vantaggi di Prefork

- Non usa i thread, quindi è compatibile con le librerie che non prevedono i threads.
- Se un processo crasha, solo una connessione viene interrotta, gli altri processi non ne risentiranno.

Svantaggi di Prefork

- Occupazione di Memoria, un processo è più pesante di un thread, a causa della duplicazione della memoria.
- Complessità della Configurazione, non è facile trovare il numero giusto di processi figli da creare nel preforking.
 - Troppo Alto → Overhead Spaziale, ho tanti processi inattivi e inutili.
 - Troppo Basso → Overhead Temporale, a causa delle successive fork.

Modulo Worker

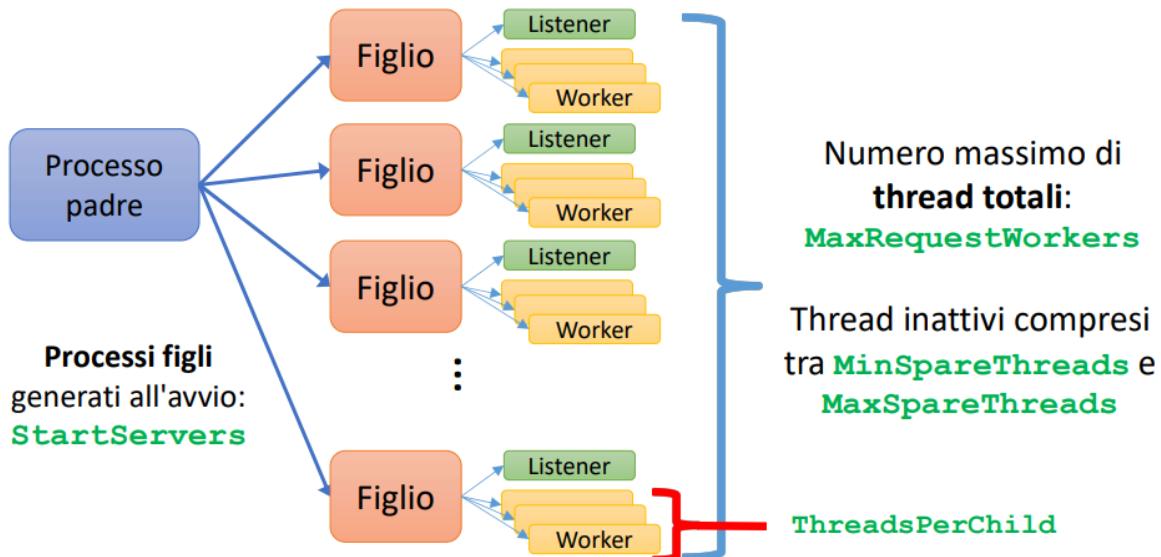
Server multiprocesso e multithread.

Il processo padre genera un certo numero di processi figli.

Ogni processo figlio genera:

- Un Thread Listener che accetta e smista le connessioni.
- Un certo numero di Thread Worker che servono le richieste.

Overhead ridotto grazie al preforking e risparmio di memoria grazie ai thread.



Modulo Event

Versione migliorata di worker che punta ad eliminare i tempi morti.

Oltre ad accettare connessioni, il listener gestisce le connessioni temporaneamente inattive

1. Un worker è connesso a un client che tarda a inviare una richiesta.
 - a. Invece di attendere, restituisce il controllo del socket al listener e passa a servire un altro client.
 - b. Quando il primo client invierà la richiesta, il listener la assegnerà a un altro worker libero.
 - c. Il worker bloccato viene messo a lavorare su un altro client in attesa che il primo faccia qualcosa.
2. Un worker sta servendo un client con una connessione lenta e il buffer di invio del socket si riempie.
 - a. Invece di attendere, restituisce il controllo del socket al listener che lo assegnerà ad un altro worker non appena sarà di nuovo scrivibile.

Aumenta il numero di connessioni servibili contemporaneamente a parità di numero di thread, eliminando i tempi morti.

Dal punto di vista delle direttive, vale quanto detto per worker.

Algoritmi di Routing

L'algoritmo di routing ha il compito di calcolare il percorso tra due end point lungo una rete. Ogni router ha una forwarding table, la quale per ogni range di Indirizzi IP di destinazione gli dice dove deve inoltrare il datagram.

Immaginiamo la rete come un grafo, dove ogni nodo è un router e ogni arco è un link di accesso.

Ogni arco ha un costo, per percorrere quell'arco devo “pagare” un costo.

Che nel nostro caso il costo è il ritardo accumulato.

Il mio obiettivo è quello di trovare il percorso di costo minimo tra due nodi.
L'algoritmo di routing ha lo scopo di trovare quel percorso.

Gli algoritmi di routing si dividono in varie categorie:

Globali o Decentralizzati

Gli algoritmi globali hanno una topologia completa della rete → Link State Algorithm

Gli algoritmi decentralizzati conoscono solo i router a loro vicini. → Distance Vector Algorithm

Statici o Dinamici

Le route cambiano nel tempo, ma molto lentamente.

Le route cambiano velocemente.

Algoritmi Link State - Dijkstra

Devo sapere tutta la topologia e tutti i costi.

E' un algoritmo iterativo: dato un nodo, facendo k iterazioni, conoscerò il cammino minimo per k nodi.

Routing Gerarchico

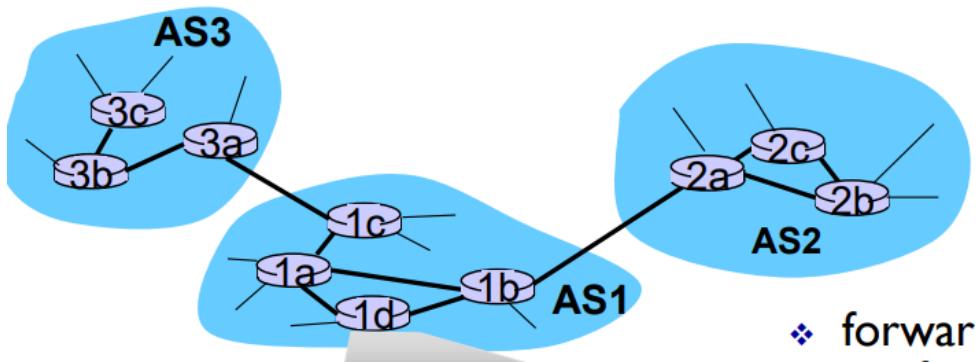
Un router non può memorizzare le informazioni per milioni di possibili destinatari.

Autonomous System

Aggregati di router i quali eseguono lo stesso protocollo di routing, detto "intra-AS" routing protocol.

I gateway Router sono i router di frontiera di un AS.

Gli AS diversi possono essere interconnessi.



Quando un router di AS1 riceve un pacchetto destinato fuori da AS1 esso deve inoltrarlo verso il corretto Gateway Router, ma quale?

Il router di AS1 deve sapere i range di destinatari raggiungibili mediante AS2 e AS3.

Questo è il lavoro degli algoritmi di routing "Inter-AS".

Algoritmi Intra-AS

Detti anche IGP (Interior Gateway Protocols).

Uno di questi è OSPF.

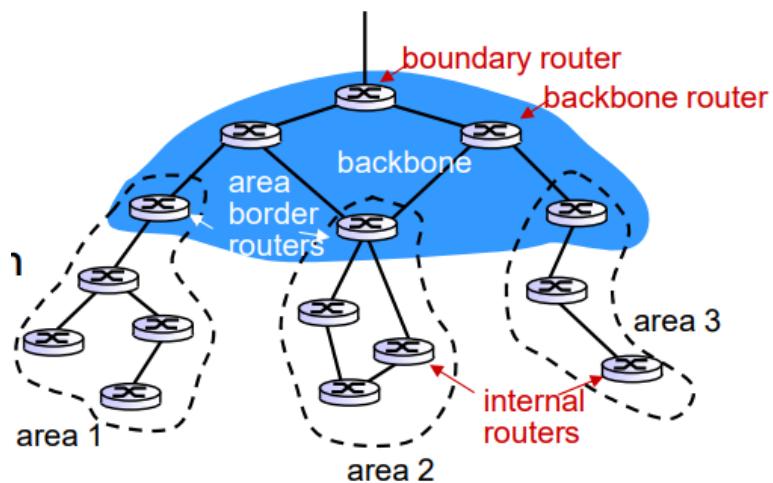
OSPF - Open Shortest Path First

Usa protocolli di tipo Link State.

Calcola l' albero dei cammini minimi per ogni percorso usando un metodo basato sull'algoritmo di Dijkstra.

Le politiche di instradamento OSPF per la costruzione di una tabella di instradamento sono regolate dalle metriche di collegamento associate a ciascuna interfaccia di instradamento.

I fattori di costo possono essere la distanza di un router (tempo di andata e ritorno), il throughput dei dati di un collegamento o la disponibilità e l'affidabilità del collegamento, espressi come semplici numeri senza unità.



Ciò fornisce un processo dinamico di bilanciamento del carico del traffico tra rotte di pari costo.

Area di Routing - Local Area

I router in una Local Area eseguono l'algoritmo Link State solo sui router dentro l'area.

Ogni nodo di una area possiede una dettagliata topologia della rete, OSPF divide la rete in aree di routing per semplificare l'amministrazione e ottimizzare il traffico e l'utilizzo delle risorse.

Le aree sono identificate da numeri a 32 bit.

Area Backbone

Per convenzione, l'area 0 (0.0.0.0) detta "backbone", rappresenta il nucleo di una rete OSPF.

Mentre le identificazioni di altre aree possono essere scelte a piacimento, gli amministratori spesso selezionano l'indirizzo IP di un router principale in un'area come identificatore di area.

Ogni area aggiuntiva deve avere una connessione all'area backbone OSPF.

Area Border Router

Tali connessioni sono mantenute da un router di interconnessione, noto come router di confine di un'area detto Area Border Router, ABR.

Un ABR ha il compito di mantenere i percorsi per tutte le aree della rete.

Backbone Router

Esegue OSPF limitato all'area di Backbone.

Boundary Router

Serve a connettere altre AS.

Internet inter-AS Routing

BGP - Border Gateway Protocol

Detto anche “La colla che tiene insieme internet”.

BGP fornisce a ogni AS varie informazioni:

- eBGP:
 - si preoccupa di capire quali AS vicino portano quali prefissi di rete (esterni)
- iBGP:
 - si preoccupa di propagare l'informazione all'interno dei router interni
- determinare una buona rotta in base alla raggiungibilità delle informazioni
- Consente ad una nuova subnet di pubblicizzare la sua esistenza in tutto internet.

Domande Pistolesi

ip addr show

Comando che ti permette di visualizzare lo stato e le caratteristiche delle schede di rete della nostra macchina.

Ricorda che in Linux tutto è un file, quindi le periferiche che vedi in realtà sono relative a file nel tuo computer.

La scheda di rete “loopback”, in realtà non esiste, è solo una scheda di rete virtuale per inviare messaggi a sé stessi.

ip addr add

Comando per assegnare manualmente una configurazione di rete o parte di essa ad una certa interfaccia di rete.

```
ip addr add 10.0.2.15/24 broadcast 10.0.2.255 dev eth0
```

Quali sono i requisiti per essere connessi ad internet?

Una macchina per navigare in internet ha bisogno di 4 parametri di configurazione:

1. Indirizzo IP
2. Subnet Mask
3. Default Gateway
4. Indirizzo di Broadcast.

Cosa è Name Service Switch

Il Name Service Switch (NSS) consente la sostituzione dei tradizionali file di configurazione Unix (ad esempio / etc / passwd, / etc / group, / etc / hosts) con uno o più database centralizzati, i meccanismi usati per accedere a queste basi essendo configurabili;

Come vedere se la macchina è connessa a internet e verifica configurazione

Per verificare che la macchina sia correttamente connessa basta inviare un ping a www.google.it.

ping www.google.it

Se questa cosa fallisse, fai in ordine:

1. Niente panico, idiota.
2. Usando il comando `ip addr show`
 - Controlla che la scheda di rete sia Accesa → UP
 - Controlla che la scheda di rete sia Abilitata → LOWER_UP
3. Vai in `/etc/network/interfaces.d`
 - Controlla che la scheda di rete abbia una configurazione valida e completa.
 - Se la configurazione è data dal DHCP, prova a metterne una manualmente e riprova, se funziona allora il Server DHCP è assente.
4. Vai in `/etc/resolv.conf`
 - Controlla che sia impostato almeno un Server DNS.
5. Usando il comando `iptables -L`
 - Controlla nella chain di OUTPUT che non ci siano policy del Firewall che diano noia al ping, ricorda che ping usa il protocollo ICMP.
6. Usando il comando `iptables -t nat -L`
 - Controlla nella chain di OUTPUT e POSTROUTING che non ci siano regole NAT che possano dare noia al pacchetto.

Come vedere se una macchina ha indirizzo statico o dinamico

Nel file in `/etc/network/interfaces` si trova la spunta `dhcp` invece di `static`.

Da dove si vede se abbiamo accesso al router di default? Qual è il suo indirizzo IP?

Vediamo dal file persistente in `/etc/network/interfaces`.

Oppure usa il comando “`sudo route -n`” nella riga con ip di destinazione `0.0.0.0`

Da dove si vede se il DNS è configurato?

1. Guardare in `/etc/resolv.conf` per vedere il DNS server è impostato.
2. Usare poi il comando `nslookup <dominio>` per testare un dominio in particolare.

Comando traceroute - come funziona

Mi dice anche la route che fa il pacchetto dal mittente verso il destinatario.

I pacchetti utilizzano il protocollo UDP:

Questo comando invia al destinatario **terne** di pacchetti alla volta, quindi invia 3 pacchetti alla volta (è rarissimo che quei 3 pacchetti facciano 3 strade diverse , in quel caso si hanno delle statistiche false).

1. Imposta TTL = 1 e manda la terna.
2. Il 1° router che i 3 pacchetti incontrano:
 - a. Prende i pacchetti.
 - b. Decrementa il TTL (TTL = 0).
 - c. Invia al mittente iniziale un messaggio di errore ICMP di tipo *Time Exceeded* con l'indirizzo del router che ha portato TTL a 0.
3. Il mittente iniziale rimanda la terna impostando TTL = 2.
 - a. Il 1° router prende i pacchetti.

- b. Decrementa il TTL (TTL = 1) e manda i pacchetti al prossimo router.
4. Il 2° router prende i pacchetti.
 - a. Decrementa il TTL (TTL = 0).
 - b. Invia al mittente iniziale un messaggio di errore ICMP di tipo *Time Exceeded* con l'indirizzo del router che ha portato TTL a 0.

Così via fino alla destinazione , la quale manderà al mittente iniziale un messaggio particolare ossia *ICMP Destination Unreachable*.

Il messaggio *ICMP Destination Unreachable* seguirà tutto il percorso all'incontrario fino al mittente iniziale. Il mittente iniziale capirà che il pacchetto è arrivato alla fine.

Comando traceroute - quando si usa e quali sono i suoi vantaggi

Ha lo stesso scopo del comando ping, ma mi offre la possibilità anche di vedere il percorso che effettuano i pacchetti che ho inviato.

Comando traceroute - come facciamo a capire da cosa deriva il timeout, ovvero se siamo stati noi a scartare il pacchetto oppure i destinatari

Dipende se va avanti, se procede semplicemente non è implementato per rispondere a ICMP altrimenti non abbiamo ricevuto risposta e non è raggiungibile.

Comando traceroute - spiega l output cosa significa se mostra gli asterischi

Il pacchetto ha toccato un Host che ha inoltrato il pacchetto ma che non ha risposto.

iptables - come aggiungere un record alla tabella.

```
iptables -t filter -A chain -p protocollo -d destinazione -j ACTION
```

NAT e PAT - Inserimento di una regola in una chain.

```
iptables -t nat
-A chain
-p protocollo
-d destinazione
-j SNAT/DNAT
```

Esercitazione 1 Pistolesi

1. Visualizzare la configurazione delle interfacce di rete
 - a. *ip addr show* → ti mostra tutte le tue interfacce di rete e i loro dati.
2. Visualizzare solo la configurazione di eth0 *ip addr show eth0*
 - a. Che indirizzo IP e Mask sono impostate? → *Non sono ancora state impostate.*
 - b. Che indirizzo MAC ha la scheda di rete? → *08:00:27:47:fe:bc*
 - c. Cos'e' l'MTU e quanto vale? → *MTU Esprime la dimensione massima di un pacchetto IP , e vale 1500 bit.*
3. Impostare come indirizzo IP dell'interfaccia eth0 l'indirizzo 10.0.2.15, maschera 255.255.255.0
 - a. *sudo ip addr add 10.0.2.15/24 broadcast 10.0.2.255 dev eth0*
4. Abilitare l'interfaccia di rete
 - a. *sudo ip link set eth0 up*

5. Visualizzare la tabella di routing → `ip route show`
 - a. Qual e' il default gateway? → *Non è stato ancora impostato.*
6. Impostare come default gateway l'host 10.0.2.2
 - a. `sudo ip route add default via 10.0.2.2`
7. Visualizzare la rotta usata per raggiungere 192.168.0.27
 - a. `ip route get 192.168.0.27` → *192.168.0.27 via 10.0.2.2 dev eth0 src 10.0.2.15*
8. Visualizzare l'indirizzo del server DNS in uso
 - a. `cat /etc/resolv.conf` → *nameserver 10.0.2.3*
9. Cambiare l'indirizzo del server DNS in 8.8.8.8
 - a. `sudo nano /etc/resolv.conf` → *scrivi manualmente l'indirizzo DNS voluto*
10. Scoprire l'indirizzo IP di Imgify.com → `nslookup imgify.com` → *18.205.222.128*
11. Rimuovere l'indirizzo IP di eth0 tramite ip
 - a. `sudo ip addr delete 10.0.2.15 / 24 broadcast 10.0.2.255 dev eth0`
12. Disattivare eth0 → `sudo ip link set eth0 down`
13. Fate una copia di backup del file interfaces (es. interfaces.bak)
 - a. `cd /etc/network`
 - b. `touch interfaces.bak` → *Se non esiste crea il file*
 - c. `cp interfaces interfaces.bak`
14. Impostare l'indirizzo 10.0.2.15, la netmask 255.255.255.0, e attivare la scheda di rete usando ifup e il file interfaces (Cancellare le impostazioni di eth0 presenti)
 - a. `nano interfaces`
 - b. `iface eth0 inet static ; address 10.0.2.15 ; netmask 255.255.255.0 ; broadcast 10.0.2.255.`
 - c. `iup eth0` → *Se lo fai prima del punto b ti dirà che eth0 non esiste.*
15. Aggiungete al file la sezione per impostare la scheda all'avvio
 - a. `nano interfaces`
 - b. `auto lo eth0`