

De Brevitate Retis

V1.33, *diffidate da imitazioni*

Raccolta di cose che SR ha detto a lezione, grazie al COVID-19.

Il nome è da intendersi come “Compendio” e non come “File da 10 paginette per darsi la materia”, perché questa materia non è affatto corta.

Il file va visto appunto come una raccolta delle informazioni che sono reperibili (facilmente) solo tramite le lezioni.

Questo documento non vuole sostituire le lezioni, i libri di testo o qualsiasi altro metodo di didattica ufficiale.

Ultima **IMPORTANTE** premessa, le prime sei lezioni circa sono state poco curate per vari motivi, tra cui l'assenza di tempo e il fatto che le informazioni sono facilmente reperibili tramite canali ufficiali, per cui ci si è concentrato di più su altre parti del programma.

Tutte le altre lezioni, fino all'ultima lezione, la N°23, sono ben curate.

Il documento è sprovvisto di esercizi con calcoli, dovete rivolgervi a un'altra fonte per quelli.

Piccolo addendum: cercando “Poveri Studenti” o “Bocciato/i” nel documento si trovano alcune domande che vengono fatte spesso in sede d'esame, ma che spesso richiedono dettagli che vengono esposti solo a lezione e non si trovano affatto (o difficilmente) sui libri.



Application Layer (3~ lezioni)

Lezione 1 – 11/03/2020

Affidabilità – Introduzione

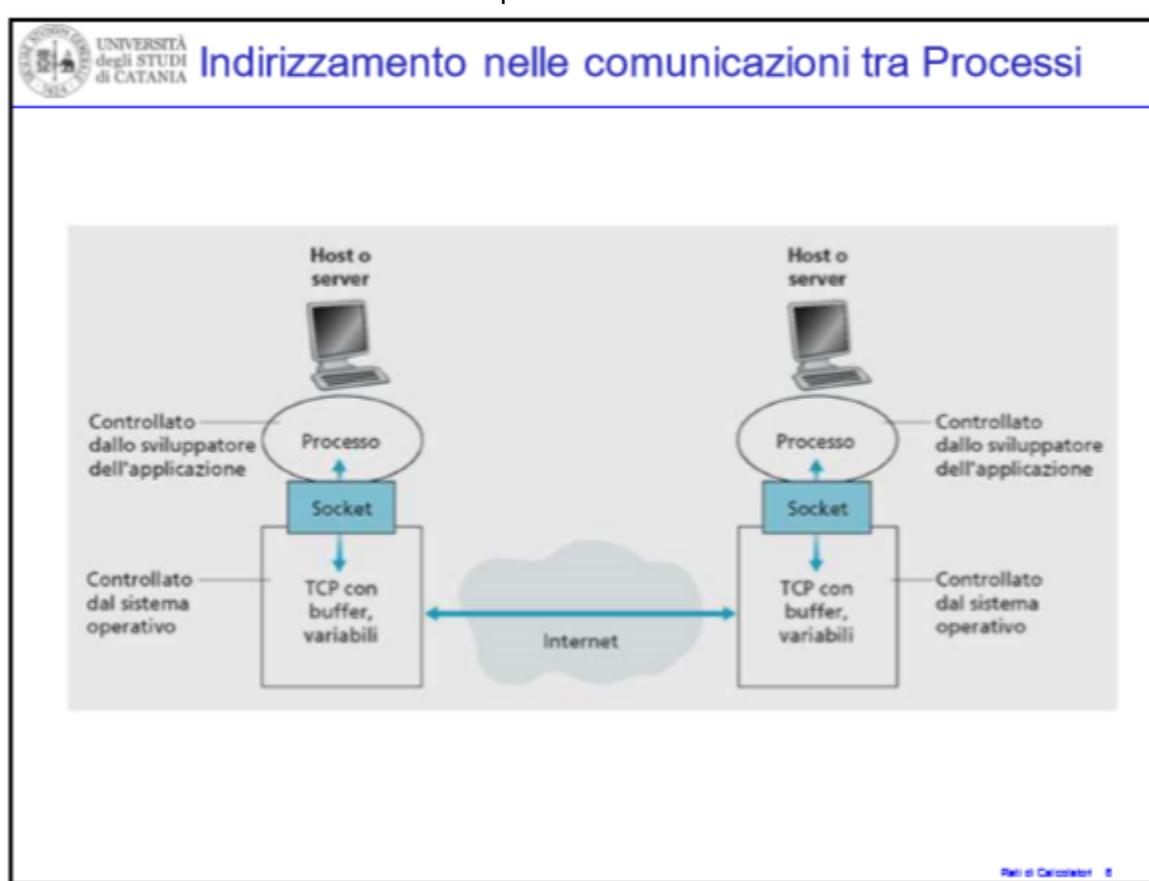
Il timer può essere una soluzione ma non è una soluzione perfetta, in quanto non abbiamo la certezza sui tempi di consegna, soprattutto per le lunghe distanze. Può capitare una situazione di questo tipo : l'Ack di risposta arriva in ritardo, non va perso, arriva semplicemente in ritardo, causando una non corretta sincronizzazione tra gli ACK. Si può tentare di risolvere il problema, fornendo un'associazione esplicita tra il messaggio, o la copia del messaggio, con l'ACK. Nello specifico, i messaggi 1 e 2 devono essere identificati come uguali, così l'identificativo può essere riportato nell'ACK. In modo tale che il terzo messaggio venga identificato come falso ACK. Questo si può fare aggiungendo dei bit di ridondanza, ma comporta una riduzione della

capacità trasmissiva in tutto il sistema.

Ritornando al concetto di protocollo, il protocollo è un insieme di regole, impone una comunicazione a priori, comunicazione che non riguarda i due (processi, persone, una cosa del genere) che vogliono comunicare, riguarda tutto il sistema che sta sotto, e sono tutte informazioni che vanno a diminuire la banda di comunicazione disponibile per i due partecipanti. Sappiamo che non possiamo dare informazioni precise sui tempi di risposta : quanto tempo aspettare l'ACK, perché il tempo di transito di un pacchetto sulla rete è molto variabile. La distribuzione dei tempi segue una Gaussiana, il tempo di attesa quindi dovrebbe essere messo in fondo alla curva, solo che significherebbe aspettare tempo prima di poter intervenire in caso di perdita; quindi non si tratta di una soluzione ottimale.

Primitive di servizio

Abbiamo due host, A e B, A manda una comunicazione verso B e B risponde. Non è però realmente così che funziona, le operazioni non sono atomiche come crediamo; queste comunicazioni sono suddivise in due operazioni distinte: due in andata e due in ritorno



Host A genera una request e Host B riceve una indication; poi l'host B risponde con una response e host A riceve una confirm. Perché questa suddivisione? Perché in mezzo c'è internet. Andiamo a vedere cosa succede all'interno di ogni singola macchina. Ogni macchina dispone(macchina A e macchina B) della linea del tempo, due generici livelli : N ed N+1.

Quando il computer A(o 1) genera una richiesta, questa operazione è effettuata tra il layer N+1 e il layer sottostante N, quindi è un'operazione interna alla singola macchina. Poi c'è la comunicazione, il layer N comunicherà con tutti i layer sottostanti, fino ad arrivare finalmente alla macchina B. La comunicazione arrivata alla macchina. Salirà tutta la pila protocollare fino ad arrivare al layer N della macchina di destinazione, che effettuerà una comunicazione con il layer N+1.

Modello OSI

Il modello di riferimento della pila protocollare è costituito da tanti livelli. Ogni livello è costituito da un protocollo, significa che vengono introdotte primitive di comunicazione tra i livelli, che vengono chiamate API: Application Program Interface. Sono primitive che servono, per esempio, al livello application per eseguire delle funzionalità del livello di presentazione (il sottostante). Ogni livello usa le primitive del livello sottostante e fornisce informazioni al livello sovrastante; e comunica contemporaneamente con il livello corrispondente dell'altra macchina tramite un protocollo dedicato. La scelta verte su quanti livelli introdurre, e cosa fa ogni livello. Se si usano pochi livelli, ogni livello fa troppo, ma se si usano troppi livelli, significa introdurre troppa burocrazia all'interno del sistema. Il modello OSI è suddiviso in sette livelli.

Il primo livello è il Livello Fisico che si occupa del trasporto dei segnali, dalla sorgente alla destinazione, attraversando tutti i nodi intermedi che esistono nella rete, per arrivare alla destinazione. Il livello fisico si occupa di stabilire quali sono i segnali, ossia che tipo di segnali utilizzare. Se si tratta di segnali infrarossi, onde radio, pezzi di carta, qualunque sistema. Si preoccupa di stabilire come modellare questi elementi, per esempio che colore deve essere la luce, con che frequenza e con che intensità, al fine di mettere d'accordo il mittente e il destinatario. O ancora i segnali elettrici, andare a stabilire il livello di tensione che bisogna avere, altrimenti può capitare che il ricettore non sente, o semplicemente si brucia, perché arriva troppa tensione. Il livello fisico si occupa anche di stabilire la forma dei connettori.

Esistono connettori, per esempio di tipo Ethernet. Tra il livello fisico e il livello Data Link si presenta il problema di associare i bit ai segnali, e come riconoscerli. Data Link si occupa di trovare una soluzione a questo problema : come deve essere interpretato un bit, quando un bit deve essere considerato 0 o deve essere considerato 1, quando esiste la trasmissione. Esiste il problema di suddividere l'informazione in frame e capire quando si tratta di trasmissione, e quando invece si tratta di rumore di fondo. Data Link si occupa di capire se ciò che è arrivato può essere interpretato come corretto o come sbagliato, o se è da scartare. Il livello Data Link riesce a capire se un frame ritenuta da buttare in realtà potrebbe considerarsi corretta. Il Data Link si può anche occupare di correggere eventuali errori, ci sono tipologie di rete che sono soggette a parecchi errori. In alcuni casi Data Link si occupa anche di trovare la strada per la destinazione, effettua un particolare tipo di routing. Funziona come il livello fisico, su connessioni dirette, da computer a computer. Livello 1 (Fisico) e livello 2 (Data Link) sono livelli base che una rete di comunicazione deve avere, per cominciare a scambiare informazioni. Al di sopra del livello 2, comincia una serie di livelli più specializzati; in particolare il Livello Di Rete è quello che permette di trovare la strada, ossia di fare routing in maniera simile a Data Link ma su scala geografica, molto più grande. Se due nodi al Livello Fisico saranno collegati in maniera

diretta tra di loro, avranno collegati in maniera diretta anche i livelli Data Link e i Livelli Di Rete. Al di sopra del Livello Di Rete iniziano i protocolli End to End, ossia, i protocolli per la comunicazione fra macchine dello stesso livello non comportano più la presenza di nodi, ma comunicano direttamente Host mittente e Host destinatario. Il Livello Di Trasporto si occupa di stabilire alcune connessioni tra i due livelli, in particolare stabilire se si tratta di comunicazioni orientate alla connessione oppure senza connessione, oppure se c'è bisogno di affidabilità nella connessione o no. Queste sono delle caratteristiche particolari del livello di trasporto. Il Livello Di Sessione è stato introdotto per ovviare ad alcuni problemi che possono nascere nel livello di trasporto, ovvero : se per un qualche motivo la connessione al livello di trasporto dovesse essere interrotta bruscamente senza la volontà da parte del Livello Applicativo, il Livello Di Sessione si occupa di aprire una nuova sessione al Livello Di Trasporto e rendere tutto trasparente al Livello Applicativo. Esempio : Stiamo facendo una telefonata, ad un certo punto si interrompe il livello fisico e a cascata si interrompono tutti gli altri livelli, in particolare il Livello Di Trasporto si accorgerà che manca la connessione, quindi nota che c'è un problema, il Livello Di Sessione si occupa aprire una nuova comunicazione verso la destinazione, significa quindi che prova a usare se disponibile un altro livello fisico, aprire una nuova connessione Data Link, trovare una nuova strada verso la destinazione, aprire una nuova connessione a Livello Di Trasporto rendendo il tutto nascosto ai livelli superiori; in modo tale che non si accorgano che è avvenuta una "catastrofe" sotto il livello di sessione, ma nota semplicemente un ritardo nella comunicazione; proprio perché in quel determinato intervallo di tempo non c'è stata comunicazione. Questo era uno dei compiti previsto al Livello Di Sessione ma è stato delegato ad un livello superiore. Il Livello Di Presentazione si occupa di rappresentare i dati in maniera uguale, per esempio quando bisogna rappresentare un treno di byte, c'è il problema di stabilire se il byte più significativo è messo prima o è messo dopo nell'allocazione della memoria. è un problema banale, riguarda la struttura del processore, però bisogna comunque dire qual è l'ordine. E quindi, il Livello Di Presentazione di occupa di stabilire uno standard per i dati, difatti si usa uno standard chiamato ASN1 che dice come devono essere rappresentate queste informazioni. Il livello più banale è il Livello Applicativo, in esso si trovano le applicazioni che l'utente può utilizzare, tipo i browser, un protocollo di scambio di file tipo FTP, un sistema di posta elettronica, insomma ci sono tutte le applicazioni che servono all'utente. Anche se il modello OSI è uno standard, ha preso sempre più piede il modello TCP/IP, che è stato sviluppato in origine in ambito universitario. In questo modello, il livello più basso è costituito dal livello Network Interface Layer; lo standard stabilisce che sotto il livello Internet, si trova una Network Interface, ma questo non specifica nulla in particolare, quindi potrebbe essere inserito qualsiasi cosa nello strato in fondo. Sopra si trova un Internet Layer, in questo livello troviamo l'IP, che sta per Internet Protocol, viene utilizzato per testare tramite un ping se c'è qualcosa che non va nella connessione, se il ping riceve risposta negativa vi è un problema ai livelli inferiori all'IP, viceversa si trova al di sopra. Partendo dal semplice fatto che il ping è un'operazione innoqua che non coinvolge alcuna porta, implica che si tratta di una violazione rispetto al modello OSI, dato che ping è un'azione che l'utente può svolgere tranquillamente nel Livello Applicativo. Questo succede perché TCP non è stato pensato a livelli, è stato ideato come una funzione di libreria. In TCP non sono presenti il Livello Di Sessione e il Livello Di Presentazione,

quindi semplicemente il Livello Applicativo si trova direttamente al di sopra del Livello Di Trasporto.

Tipologie Di Rete

La più semplice tipologia di rete, è la connessione Punto a Punto. Consiste di due Host : A e B; sono connessi senza intermediari. È il sistema più semplice perché bisogna solo capire se la connessione è unidirezionale, half duplex, o full duplex. Nel caso dell'unidirezionale, A può parlare solo con B, ma B non può rispondere. Nel caso di half duplex A può parlare con B e quest'ultimo può rispondere però in tempi differenti. In full duplex A e B possono parlare contemporaneamente senza problemi. Un altro tipo di rete è il modello broadcast. In questo modello vi sono una serie di macchine interconnesse tra di loro, anche questo è un modello abbastanza comune. Quando la macchina parla, comunica con tutti, si definisce per questo comunicazione broadcast. La difficoltà sta nel dover far rispondere una macchina alla trasmissione di un'altra mentre è ancora in corso. Nel broadcast bisogna anche specificare la differenza tra le trasmissioni: alcune trasmissioni sono per tutte le macchine in ascolto, altre previa specificazione sono solo per determinati dispositivi. Qui si introduce il concetto di indirizzo, il cui scopo è specificare il destinatario. Nella comunicazione il mittente mette il proprio indirizzo, mette l'identificativo(o indirizzo) del destinatario, e tutto questo mentre gli altri dispositivi restano in ascolto. Semplicemente chi non è interessato a questo messaggio, si preoccuperà di ignorare la comunicazione, perché capiscono che non è rivolta a loro. Questo meccanismo crea un problema però, perché anche se le macchine non interessate alla comunicazione non interagiscono, restano comunque bloccate per tutta la durata della comunicazione da cui sono state escluse, non fanno assolutamente niente. Invece, se dovessero parlare tutte in contemporanea, avviene una sovrapposizione delle informazioni. Dal punto di vista teorico si parla di sovrapposizione nel vero senso della parola, in realtà il segnale trasportato va trasformato in bit; questa trasformazione non è sempre possibile, poiché per realizzarla bisogna capire il segnale in arrivo. Per risolvere il problema, va stabilito un protocollo per gestire l'ordine di chi deve parlare e quando. Un modo può essere definito come un'interruzione spontanea del broadcaster, per chiedere parola agli altri; anche se in realtà questo protocollo non è efficiente, perché si torna comunque a una sovrapposizione. Una soluzione banale è quella di Ethernet, che nonostante prenda delle precauzioni, in termini generali possiamo dire che permette la parola a chiunque abbia qualcosa da comunicare, senza curarsi delle possibili collisioni. Un altro modo può essere fare a turno, le macchine tra loro si passano il Token, se non hanno niente da dire passano la parola alla successiva. Un token non è un oggetto fisico, è un'informazione che se passa di macchina in macchina correttamente tutto fila liscio, però capita che il token si perda per interferenze o motivi simili, quindi l'informazione arriva falsata. In un caso simile la macchina che riceve l'informazione in questo stato non sa se ha o meno il permesso. Bisogna rigenerarlo dato che è considerato perso. Il tipo broadcast è suddiviso in 3 categorie : a bus condiviso, ad anello, a stella. In caso di collisioni, il broadcast su bus non è un buon sistema. Nel caso della comunicazione tra due macchine all'interno di un anello, se le macchine sono l'una di fronte l'altra, l'informazione impiega praticamente tutto l'anello per arrivare a destinazione, e questo rallenta quindi tutta la struttura.

Il modello più adatto è quello a stella che permette a una macchina di mandare l'informazione nell'unità centrale, quest'ultima manda l'informazione alla macchina destinataria.

Commutazione a circuito

Veniva utilizzato quando i telefoni utilizzavano il disco. Tramite questo sistema l'utente, che prima di allora lasciava gestire la comunicazione alle centraliniste, ha finalmente la possibilità di realizzare il circuito. Tramite una serie di informazioni di controllo a un primo elemento, viene selezionata da esso una particolare uscita, e avviene in questo modo finché non si trova la destinazione, in base una serie di passaggi. Il vantaggio di questo sistema fu poi meccanizzato in modo da non avere più bisogno di attivazione da parte di un umano. Un altro vantaggio era la possibilità di resettare il sistema alla chiusura della connessione. Tra gli svantaggi possiamo notare che, nel caso in cui un circuito veniva dedicato ad una determinata telefonata, risultava bloccato su quella per tutta la sua durata; significa che in una centrale il limite di telefonate era limitato in base al numero di circuiti presenti nelle centrali. Un altro svantaggio era il fatto che la linea di comunicazione veniva bloccata nel caso di una certa telefonata.

Commutazione di pacchetto

Utilizza una distribuzione a grafo, rispetto alla distribuzione ad albero della commutazione a circuito. Torna un modello utile perché un guasto nel modello ad albero causava un blocco totale nel sistema, invece nel grafo, l'interruzione di un nodo avrebbe bloccato il collegamento solo ad un determinato sottografo, per un blocco totale bisognava avere molti più guasti. Si parla di una comunicazione tramite nodi intermedi dalla sorgente alla destinazione, e proprio come un grafo, si trova un cammino che arriva fino al destinatario desiderato. Nacque quindi ARPANET.

Lezione 2 – 13/03/2020

Comunicazioni tra processi – Livello Di Applicazione

In un sistema distribuito, due processi remoti possono procedere in parallelo solo se riescono a sincronizzarsi tra loro. Lo scambio di messaggi consente la sincronizzazione. Se abbiamo processi paralleli lanciati come thread da un processo generatore, essi convergono verso un risultato finale quando tutti terminano la propria elaborazione; quindi si riprende il processamento seriale. Quando abbiamo due processi presenti su due macchine differenti, separate fra di loro, questi processi non vanno avanti indipendentemente l'uno dall'altro, ma devono scambiarsi delle informazioni. Quel che tipicamente avviene è che un processo rimane fermo in attesa dell'input dei dati provenienti dall'altro processo. Viene eseguita un'elaborazione e tipicamente ne segue una risposta. Se una delle comunicazioni salta, la sincronizzazione si perde, quindi dobbiamo trovare un modo per ripristinarla.

Modello Client Server

È il modello principale di riferimento. È un modello abbastanza funzionale, che ritroviamo quasi sempre nei sistemi di comunicazione. Il sistema è caratterizzato da una serie di processi su una macchina che sono in attesa di input provenienti dall'esterno. Questi processi appena ricevono

l'input vanno avanti con la loro elaborazione, preparano dei dati che poi saranno restituiti ai client. Notiamo che il server, dato che deve restare in attesa di un input dall'esterno, deve sempre essere attivo. Il client può essere spento, e si accende quando ha necessità di informazioni. Spedisce la richiesta al server, successivamente potrebbe tornare nello stato di spento e accendersi solo quando deve ricevere la risposta da parte del server. Quindi mentre il server deve sempre restare acceso, attivo; o in attesa o in elaborazione, il client ha un comportamento molto più variabile nel tempo; deve essere disponibile alla comunicazione solamente quando ci deve essere lo scambio delle informazioni. Chi fa l'elaborazione tipicamente è il server, il client non fa gran che di elaborazione. Possono esserci tanti client connessi allo stesso server come macchina fisica. I livelli applicativi li troviamo solo nella macchina sorgente e nella macchina di destinazione. Ci ritroviamo 5 livelli nella pila protocollare : Livello Fisico, Livello Di Collegamento, Livello Di Rete, Livello Di Trasporto, Livello Di Applicazione. Dato che i livelli applicativi si trovano solo sulla macchina sorgente e su quella di destinazione si parla in questo caso di comunicazione End-To-End. Si salta totalmente ciò che c'è nel mezzo e parlano solamente le due macchine remote. Il livello applicativo non sa e non vuole sapere cosa si trova nella rete sottostante. Ovviamente si passa negli altri livelli per effettuare la comunicazione, ma essenzialmente il livello applicativo vuole parlare con l'altro livello applicativo. Per poter dialogare con il mondo esterno, nello specifico, il processo deve dialogare con un oggetto chiamato Socket. La socket è un'interfaccia, fornisce le funzionalità per poter accedere a tutta la rete sottostante. Essenzialmente in TCP/IP è realizzata con funzioni di libreria. La prima cosa da dire parlando delle socket, sono le porte di comunicazione. Il processo A deve parlare con il processo B, in un sistema operativo ogni processo viene identificato da un process ID denominato PID; all'interno del sistema i processi possono comunicare tra loro indicando l'ID del destinatario del messaggio; invece tra processi remoti la comunicazione avviene tramite le porte, poiché in due macchine diverse non è facile che la macchina possa prevedere il PID del processo destinatario. Entrano quindi in gioco le porte che possono essere paragonate al nome e al cognome del destinatario all'interno della struttura di destinazione. Le socket vengono numerate con un sistema separato rispetto al sistema del pid; il sistema operativo si occupa di effettuare una mappatura tra numeri relativi alle socket e numeri relativi ai processi. Questo consente una cosa molto interessante. La mappatura può essere fatta seguendo delle regole differenti, ovvero; le port numbers, che sono numeri assegnati alle socket, sono numeri a 16 bit e hanno 3 gruppi separati. Abbiamo le Well Known Ports (Servizi di Sistema) che sono i valori compresi tra 0 – 1023. Sono porte assegnate ai servizi di sistema. In base al tipo di comunicazione che il processo A vuole stabilire con il processo B, nonostante abbiano due pid diversi, andrà specificata la porta richiesta per il tipo specifico di comunicazione. Il numero di porta è fisso e conosciuto da tutti i sistemi, cosa che non avviene con i pid, essi vengono assegnati sequenzialmente ad ogni nuovo processo creato e sono differenti in ogni computer, per questo non ci si può basare su questo per la comunicazione. La socket rappresenta il modo in cui i processi possono comunicare tra di loro. Nella categoria delle well known ports, abbiamo porte per servizi critici, tutte le porte richiedono l'autenticazione da parte dell'utente. Cioè l'utente deve fornire le sue credenziali di accesso alla macchina remota. Solo un amministratore può richiedere l'assegnamento a una porta compresa tra 0 – 1023 perché sono tutti processi critici. Poi ci sono le Registered Ports (1024 – 49151),

che sono assegnate da Internet Corporation for Assigned Names and Numbers per qualche uso specifico; e le Dynamic and/or Private Ports (49162 – 65535).

Requisiti delle applicazioni

1) Fault Tolerance

Tolleranza guasti e problemi di comunicazione

2) Throughput

Banda richiesta per la comunicazione

3) Time Constraint

Limiti temporali per la comunicazione

4) Security

Sicurezza

Servizi forniti da IP ??

??

UDP e TCP

Sono due protocolli di trasporto, UDP sta per User Datagram Protocol e TCP sta per Transmission Control Protocol. UDP è un protocollo senza connessione e non è affidabile, TCP è orientato alla connessione, affidabile, congestione della banda (ottimizzazione della banda), ottimizzazione del flusso. TCP impiega tempo e risorse, è il più utilizzato e si potrebbe dire che in confronto a UDP ha proprio tutto, è il più complesso. Se l'applicazione non richiede una particolare affidabilità utilizza UDP perché più snello ed è “più veloce” nella sua esecuzione.

Telnet

Telnet è un modello, la cui pila protocollare è costituita da 4 livelli. In cima abbiamo il Livello Applicativo, subito sotto il Livello Di Trasporto, a seguire il livello Internet e il Network Access. Al livello di trasporto si utilizza TCP. Telnet sta per Teletype Network. C'è un client al livello applicativo che si collega alla porta 23 e suddetto client ha la possibilità di collegarsi a qualunque porta del sistema remoto. In Telnet c'è un processo client che controlla il terminale del client, chi manda le informazioni è invece il vero e proprio Telnet Client, che utilizzando TCP comunica con il processo Server di Telnet, il quale, volendo, può comunicare con un'applicazione presente nel server.

HTTP

Http è un protocollo che serve a trasferire file da una macchina all'altra sotto richiesta. Il client ha un suo applicativo, cioè il browser, che spedisce una richiesta al server web, specificando un indirizzo particolare determinato : URI (Uniform Resource Identifier). In origine il server andava a cercare il file da restituire in ritorno al processo browser. Il sistema è stato potenziato con il concetto di Hyperlink in maniera tale che, se al server veniva richiesta un'informazione aggiuntiva, o il server stesso, o comunque il browser, potevano ricevere informazioni anche da altre macchine. HTTP usa protocollo TCP. Il client invia la richiesta dei file che vuole ricevere, questa richiesta arriva al server che risponde con la lista completa dei file. Tipicamente a richiesta soddisfatta la connessione viene chiusa. La prima versione del protocollo (1.0),

prevedeva una chiusura della connessione diretta ogni volta che veniva mandato un singolo file; nel caso quindi della richiesta di più file, veniva utilizzata una connessione per ogni file. La variante successiva (1.1), permette di specificare da parte del client, di tenere aperta la connessione finché il client capisce di aver caricato interamente la pagina web, e solo allora la connessione viene chiusa.

Formato richiesta HTTP

Nella prima linea di testo c'è un campo dove c'è il metodo richiesto dalla connessione HTTP, dopo c'è uno spazio e a seguire la URL (Uniform Resource Locator), a seguire un altro spazio, successivamente la versione del protocollo e i caratteri utilizzati per il ritorno a capo (cr : carrier return; lf : line feed).

Sotto la prima linea ci sono una serie di altre linee chiamate Linee Di Intestazione, dove sono espressi un nome e un valore, variabile e contenuto. La lista di questi nomi seguiti da valori termina quando si ha una linea vuota (contenente solo i caratteri cr e lf). In fondo ci sono altre informazioni aggiuntive (Corpo).

Risposta HTTP

La risposta ha lo stesso tipo di strutturazione della richiesta, ma cambia la linea in cima.

Abbiamo Versione /spazio/ Codice di stato /spazio/ Frase /cr/lf/.

Metodi HTTP

1) Get

Richiesta per leggere una pagina web

2) HEAD

Richiesta per leggere un header di una pagina web

3) PUT

Richiesta per conservare una pagina web

4) POST

Fare un APPEND a una risorsa nominata

5) DELETE

Rimuovere la pagina web

6) TRACE

Fare una ECHO alla richiesta in arrivo

7) CONNECT

Conservare per uso futuro

8) OPTIONS

Fare una query per certe opzioni

DHTML

Il protocollo Dynamic HTML è un protocollo che permette all'utente, tramite il browser, di effettuare una richiesta, ma essa è dinamica; significa che il server non troverà subito la pagina pronta, ma interagendo con degli script o con dei processi sul server, prepara l'elaborazione della pagina, riceve la risposta dal database, prepara la pagina in formato HTML, infine risponde al browser. Questo processo è utilizzato sempre, è inutile avere pagine statiche, sono tutte dinamiche. Se voglio fare una ricerca subito dopo un'altra, non ha senso mandare di nuovo

le credenziali, sfruttando la struttura dell'HTTP , che sfrutta tante connessioni TCP ad alto livello, si può trovare una soluzione. Dato che le richieste fatte dallo stesso client allo stesso server sono trattate come richieste separate, serve un metodo per dire ad alto livello che tutte le richieste sono collegate fra di loro, sono parte di una stessa sessione di lavoro. Serve un metodo per legare fra di loro tutte le credenziali, questo metodo si chiama Cookie. Il cookie è un numero che viene generato e spedito dal server host, viene mandato al client, rendendo tutte le operazioni collegate alla prima semplicemente trasportando lo stesso cookie. Il server si ricorda che il cookie è legato ad un certo utente o sessione di lavoro, quindi non chiede più l'autenticazione.

Proxy Server

Molto spesso i client chiedono l'accesso a pagine sempre uguali. Invece di fare richieste che vanno sempre al server web, si usa un Proxy Server, che usa una sorta di memory cache all'ingresso del sistema. In modo tale che la prima richiesta della serie, trova la cache vuota, e la pagina viene memorizzata sul proxy server e distribuita. Le successive richieste, se rispettano certi requisiti, saranno gestite direttamente dal proxy server senza andare su internet.

Lezione 3 – 16/03/2020

FTP

FTP (File Transfer Protocol) è uno dei protocolli più vecchi che è stato pensato per internet. Serve per trasferire file da una macchina all'altra. È un protocollo bidirezionale, significa che l'utente poteva sia caricare sul server sia prendere dal server file da caricare nel filesystem locale. Non è un vero e proprio filesystem di rete, nel senso che l'utente non vede il filesystem remoto come se fosse una parte del suo filesystem locale, ma deve esplicitamente trasferire file da una macchina all'altra facendo una copia. Non lavora quindi sulla copia remota, ma su una copia locale. Successivamente se necessario quest'ultima va ricaricata sul server. FTP utilizza due connessioni separate : una sulla porta 20 e una sulla porta 21. La porta 20 serve per il trasferimento dei dati, la porta 21 per la connessione di controllo. La prima operazione che viene fatta con il protocollo FTP consiste nel far eseguire all'utente tramite l'interfaccia, l'operazione di log-in su un server remoto. La connessione rimane aperta per tutta la sessione di trasferimento dei dati, viene poi chiusa solo in due occasioni : per time-out(circa 900 secondi di non comunicazione tra client e il server), in cui il server chiude la connessione per inattività dell'utente; oppure viene effettuata in maniera attiva dall'utente con i comandi di uscita. La sessione di lavoro, la connessione, è unica, rispetto ai protocolli web quindi rappresenta una grande semplificazione. Qui non sono necessarie le sessioni di lavoro di HTTP. Si dice di HTTP che è un protocollo senza stato, FTP è invece un protocollo con stato. HTTP è composto da una serie di connessioni TCP separate che sono collegate da un'unica sessione logica di livello superiore tramite i cookie. Sono infatti i cookie che realizzano lo stato dentro HTTP. In FTP questo problema non c'è, l'utente al momento di creare la connessione TCP si autentica e mantiene la connessione per tutta la durata della sessione FTP. La sessione TCP e FTP coincidono, mentre così non avviene in HTTP, infatti sono stati introdotti i cookie. Il protocollo

FTP è definito protocollo sia push sia pop, si possono mandare e si possono prendere dati dal sistema.

SMTP

Simple Mail Transfer Protocol. È il protocollo della posta. Prima la posta funzionava in modo differente da ora. Ogni utente aveva un account completo caricato su una macchina (server); avevano un account diretto. Ogni utente aveva all'interno del proprio server un file unico messo in una zona controllata dall'amministratore con il nome del log-in dell'utente, dove erano messe una di seguito all'altra tutte le e-mail dell'utente, per cui l'interfaccia che aveva l'utente per accedere, al momento della scrittura, era scomoda poiché conteneva lo storico di tutte le e-mail ricevute. Questa interfaccia permetteva di separare le e-mail e presentarle come se fossero tutte dei messaggi separati. Al momento di trasferire l'email, l'utente, tramite la sua interfaccia, dava le informazioni base (chi era il destinatario, il subject, ecc..) e scriveva un pezzo di testo che veniva messo in append sul file remoto dell'utente. Era (molto genericamente) un modo per fare transfert di file da una macchina all'altra. Per identificare la macchina remota, l'utente doveva comporre un indirizzo composta da log-in dell'utente remoto, @, nome della macchina remota. Si pensò di rendere il sistema più semplice per l'utente. Fu aggiunto un altro applicativo oltre il server di posta, che permetteva di entrare sul file della posta ed effettuare dei trasferimenti ad un altro applicativo separato, chiamato Agent di Posta. Si avrà quindi un computer remoto, con un suo agent di posta che si connette tramite un protocollo di comunicazione via rete, al server di posta, interagisce con il file e poi invia il file. Fu poi rimosso il login, inserendo i profili dentro il server, trasformando l'utente in account locale al server, e furono aggiunti una serie di account virtuali autorizzati con log-in e password.

POP3 e IMAP

Il protocollo SMTP necessita di tenere gli account nei server, è un grosso rischio per la sicurezza. Si pensò di togliere quindi l'utente dal servizio di posta e farlo connettere in remoto da uno user agent, connesso a un altro processo chiamato POP3 (Post Office Protocol). POP3 serve per interagire con il file, chiedere l'e-mail, chiedere di leggere la n-esima e-mail, preparare a spedire un file a un certo indirizzo. Serve allo user agent per interfacciarsi con la macchina server per gestire l'ingresso e l'uscita dalla macchina. Non spedisce le e-mail, dà dei comandi per farlo. Anche POP3 è un protocollo che trasferisce le informazioni in chiaro. Funziona in due modalità. Può chiedere di leggere il messaggio e spostarlo totalmente sullo user pc, oppure può lasciarlo come letto nel server. Questa modalità funziona bene nel caso di un utente che dispone di un solo PC. Ad oggi abbiamo tanti pc separati ma una sola casella di posta, serve quindi un nuovo protocollo. IMAP fa al caso nostro. Con POP3 sul server l'informazione viene rimossa, non rimane nulla. Col protocollo IMAP l'utente accede al server, la mail può cancellarla se vuole ma si può lasciare sul server, in modo tale che se ci si collega da terminali separati, la mail sarà sempre raggiungibile sul server. Da qui nasce la flessibilità di IMAP. Anche IMAP manda i messaggi in chiaro. POP3s e IMAPs sono due varianti che prevedono la cifratura delle informazioni. Inoltre sia POP3 sia IMAP prevedono lo scambio delle credenziali di accesso dell'utente. Gli utenti devono prima di tutto autenticarsi.

DNS

Il DNS (Domain Name System) è molto importante per la memorizzazione degli indirizzi IP. È un sistema che traduce i nomi dei siti in indirizzi. Ogni macchina ha un suo server DNS, che ha una base di dati locali, a volte limitata, e ha almeno un riferimento verso il mondo esterno. Un determinato host chiede la traduzione di un indirizzo, il suo DNS di riferimento può sapere come rispondere a questa richiesta, e avviene immediatamente; oppure deve chiedere al suo riferimento principale. Si continua a salire di livello fino a quando si incontra un server che sappia cosa fare. I nomi sono lunghi massimo 255 caratteri e sono sequenze di lettere separate da punti. In caso di mancanza di informazioni, la cosa principale da fare è risolvere l'ultimo pezzo. L'ultimo pezzo ci dice dove dobbiamo andare (esempio nomeindirizzo.fr ; fr sta per Francia ed è quindi un primo suggerimento di dove iniziare la ricerca del server), si comincia a salire fino a quando si trova una macchina che gestisce quel dominio; la macchina trovata sa come gestire il dominio interessato, la macchina successiva a cui farà riferimento sa anche a che indirizzo si trova la macchina che si chiama DNS, per cui la risposta torna indietro a ritroso. La macchina che ha fatto la richiesta in origine potrebbe mettere nella cache l'indirizzo che voleva risolvere, in maniera tale che operazioni successive rivolte a quell'indirizzo, non richiedano il dover risalire tutto l'albero di DNS per trovare la destinazione. Ci sono 14 macchine root server, in tutto il mondo; queste macchine sanno come risolvere la parte finale degli indirizzi. Il server radice potrebbe chiedere al server secondario, ma potrebbe anche rispondere a chi ha fatto la richiesta, dicendo a chi si deve rivolgere. Rispetto al modello ad albero è preferito un collegamento in parallelo, a grafo. Poiché risalire sempre sino alla radice e scendere in un sottoalbero della gerarchia intasa il sistema. I collegamenti trasversali sono migliori. Per ogni dominio locale esiste un server locale che conosce i nomi di tutte le macchine che ci sono nella sua organizzazione. Per non intasare il sistema non si usa un server soltanto, causerebbe un intasamento dato che il DNS di un'organizzazione è un servizio critico. Il traffico è diviso quindi tra tutte le macchine secondarie, che sono altri DNS. Copiano l'informazione dal server primario e la salvano. Nel momento in cui viene fatta una modifica dell'informazione, essa viene fatta nel server primario, è un errore effettuarla su un server secondario perché limiterebbe l'informazione solo a un gruppo ristretto del dominio. Dopo l'aggiornamento sul server primario, passato un po' di tempo (tipicamente mezz'ora), l'informazione viene trasferita a tutti i server secondari. Le altre informazioni sono soggette a scadenza perché messe nelle cache. I Record DNS sono specificati da delle lettere, che servono a specificare il tipo di informazione. Un record che inizia per A è la risoluzione di un nome corrispondente a un indirizzo ipv4, mentre un nome che inizia per AAAA è un nome ipv6. Per ogni nome possono esserci degli alias, quindi l'informazione inizia per CNAME, se è un server di posta inizia per MX, e così via. È un sistema molto comodo perché, per esempio, con il Record MX, è possibile evitare di indirizzare la singola macchina. Ricapitolando il concetto di SMTP, c'è un server di posta, una macchina specifica presente in rete; è più comodo avere una farm di server piuttosto che averne uno solo, in modo di poter gestire meglio la connessione da parte dell'utente (sceglie il più vicino, il meno carico, ecc..).

Grazie ad MX non è possibile soltanto gestire la connessione al server, ma non è più

necessario inserire un indirizzo completo ma basta inserire un indirizzo di dominio. ES. gmail.com non è una macchina, è un dominio; con MX si può avere una corrispondenza tra un dominio e una macchina specifica, ossia un indirizzo IP completo. Questo record MX può variare in base al traffico, in funzione del posto, per cui il DNS può rispondere in maniera differente, grazie a questo record, a seconda da dove ci troviamo, o a seconda del carico delle macchine che ci sono. Se ad un certo punto il sistema si accorge che c'è una precisa macchina troppo carica, una richiesta di risoluzione (es gmail.com) viene indirizzata a una macchina differente.

È quindi possibile effettuare la tecnica di Load Balancing, in base al traffico e alla posizione geografica.

DNS - Recap (Preso dalla Lez.4)

La struttura DNS ad albero si può trasformare facilmente in una struttura a grafo. Ogni partecipante ha il proprio agent DNS al quale gli applicativi chiedono di risolvere i nomi. L'agent ha due possibilità: o conosce già la risposta, perché l'ha scoperta in passato, oppure deve chiedere a qualcun altro. Ogni agent deve avere almeno un riferimento di un DNS ad un livello superiore, in modo tale che si possa risalire l'albero e arrivare ad uno dei DNS radice che possono procedere alla risoluzione dei nomi. Il funzionamento è semplice, il DNS locale conosce tutto ciò che sta al di sotto della sua rete locale, quindi è l'unico autorizzato a dare una risposta certa per tutto quello che è il dominio sottostante. I DNS di livello superiore, conoscono dell'indirizzo completo, solamente la parte finale, non quella iniziale; e via via salendo, il server di livello superiore, essenzialmente uno di quelli radice incluso tra i 14 che esistono al mondo, conoscono quasi esclusivamente il modo di rintracciare le parti finali del nome; ossia a chi rivolgersi per risolvere un determinato nome. Man mano che si scende di livello si risolve anche la parte centrale del nome. L'ultima parte viene risolta esclusivamente dal server locale, ossia quello che conosce tutto.

Transport Layer (5 lezioni)

Lezione 4 – 18/03/2020

Livello Di Trasporto

Il livello di trasporto fornisce una comunicazione logica tra gli host, saltando tutte le problematiche della connessione di rete. Anche con il livello di trasporto parliamo di un protocollo end-to-end, perché non coinvolge nessuna delle macchine sottostanti, anche se ciò rappresenta un problema dato che non è possibile reperire informazioni utili dal livello sottostante. Il livello di trasporto più parlare solamente con il livello di trasporto corrispondente. TSAP e NSAP. TSAP sta per Transport Service Access Point, ossia punto di accesso al servizio di trasporto. Il suo corrispondente è NSAP, ossia Network Service Access Point. Il TSAP coincide l'identificativo della porta, NSAP coincide con l'indirizzo IP della macchina. L'idea è la

seguente. Abbiamo una macchina server, B, con tanti processi attivi contemporaneamente al livello HTTP, immaginiamo di avere due host separati, host A e host C. Sull'host A c'è solo un web client, sull'host C ci sono due finestre aperte sullo stesso server web, quindi due connessioni logiche separate. Dobbiamo identificare per ogni browser, ossia per ogni processo presente sulle due macchine, tre processi separati. Si fa come segue. L'indirizzo IP di destinazione della macchina B, l'indirizzo di sorgente coincide con la macchina, avremo quindi due processi con lo stesso indirizzo IP, e la porta che sarà differente. Per instaurare la connessione bisogna avere i parametri con almeno un valore differente. Si parla quindi servizio di Multiplexing e Demultiplexing. Guardando i 4 parametri, i quali sono IP sorgente, Porta Sorgente, Porta di Destinazione e IP di destinazione; è possibile individuare il processo corretto che deve gestire la determinata informazione. Le informazioni di un processo non devono finire nel canale dell'altro, devono finire perfettamente separate. Se non ci fosse questo meccanismo, il rischio è che la risposta, quando arriva all'host C, non saprebbe a chi mandarla dei due processi. Invece così, con il meccanismo appena descritto, il sistema permette di conoscere perfettamente quali sono i processi che devono essere messi in comunicazione tra di loro.

Indirizzamento con un Name Server

Abbiamo un processo identificato come Process Server, che è in ascolto su tanti elementi, tante porte differenti. L'utente si connette a questo processo effettuando un'operazione, mandando una richiesta o qualcos'altro. Il processo del server indirizza ad un servizio che svolge una qualche attività. Per questioni di sicurezza conviene disporre solo un processo che rimanga in ascolto su tutte le porte. Al momento in cui arriva una richiesta, il processo identifica il tipo di richiesta che è arrivata, lancia un processo separato, un processo figlio per esempio e gli passa la connessione. Il solo processo ascolta le richieste in arrivo e capisce se le richieste sono lecite o meno.

UDP

UDP fornisce un servizio di Datagram per le applicazioni. I segmenti dati vengono passati immediatamente al livello di rete, aggiungendo un piccolo preambolo. UDP non ha un protocollo end-to-end (a parte il mux-multiplexing-delle porte). Ci sono due servizi al livello di trasporto, il servizio UDP e il servizio TCP. UDP è un servizio Datagram senza connessione e senza affidabilità. Quando arriva un messaggio dal livello superiore, viene preparato un pacchetto UDP e viene spedito al livello di rete, aggiungendo solamente le informazioni base che sono Source Port Number e Destination Port Number. Non è corretto definirlo protocollo end-to-end, perché non esiste nessuna connessione. Dispone di controllo sul flusso di dati da parte del protocollo, non dispone nessuna connessione, nessuno stato, ha un piccolo overhead di gestione. Il protocollo si limita a dire qual è la porta sorgente e qual è la porta di destinazione, non fa altro. Se si vuole uno stato ci si deve riferire al livello superiore. Se alla porta di destinazione non c'è nessuno in ascolto, il pacchetto viene banalmente perso. Questo dipende dal fatto che non c'è nessun controllo sul flusso dei dati da parte del protocollo. UDP è una comunicazione unidirezionale, ma possibilmente la risposta può essere necessaria, difatti troviamo una request, e una response, solo che non è detto che la response ci sia. L'applicativo si fa dare e si fa assegnare una porta sorgente (o la richiede), indica necessariamente la porta

di destinazione, e aggiunge il suo payload, i dati, a destinazione; la destinazione, se ha necessità di rispondere può utilizzare la stessa porta sorgente, però non è detto che ci sia alcuna risposta perché UDP è un processo unidirezionale. Se si perde un pacchetto in UDP non c'è un modo per saperlo, non c'è affidabilità alcuna. Serve un livello applicativo che può gestire la perdita di pacchetti, o semplicemente non c'è bisogno di garanzia che i pacchetti arrivino a destinazione. Lo schema UDP è usato dalla telefonia, perché essendo una comunicazione in tempo reale non ha senso inviare di nuovo pacchetti vecchi perduti.

Reliable Data Transfer – rendere affidabile una connessione di rete

Al livello di trasporto vogliamo un canale affidabile, ma al livello di trasporto il canale è inaffidabile. Serve un modo per rendere affidabile il canale. Sono stati ideati una serie di protocolli base che hanno dato vita al protocollo TCP, almeno, una parte di esso.

RDT 1.0

Immaginiamo di avere al livello fisico un canale affidabile e senza perdite. Si definisce di base perché da qua si iniziano a ostruire i pezzi aggiuntivi, togliendo le caratteristiche di affidabilità e mancanza di perdita. Immaginiamo una macchina a stati finiti che ha un solo stato, difatti ritorna su se stessa. Ogni volta che arriva un messaggio dal livello superiore, si crea un pacchetto con i dati ricevuti, e viene mandato attraverso la funzione `rdt_send()`, interfaccia tra livello applicativo e di trasporto; poi c'è una primitiva definita tra livello di trasporto e livello di rete chiamata `utd_send()`, la prima è affidabile, la seconda no. In pratica se si stabilisce che il canale al livello di rete è affidabile e senza perdite, la funzione `rdt_send()` equivale a `utd_send()`, e vanno aggiunte semplicemente le porte che servono al protocollo di trasferimento. D'altra parte, per la ricezione ci sarà `rdt_rcv(packet)`, che estrae i dati dal pacchetto ricevuto, e li spedisce al livello superiore. Il canale può inserire errori, per questo definito inaffidabile. Il ricevente, per risolvere il problema, potrebbe mandare un ACK di risposta se il pacchetto è stato ricevuto bene, o un NAK in caso negativo, ossia se è stato ricevuto in maniera errata.

RDT 2.0 sender

In questo caso la macchina ha due stati differenti. Uno stato di attesa di informazioni dal livello superiore, l'altro invece, dopo che ha scritto i dati, se è in attesa resta in attesa di un ACK positivo, o un NAK negativo. Una volta effettuata questa trasmissione di dati, il trasmittente si mette in ascolto, passa allo stato bloccato, in attesa di ricevere un ACK positivo o un NACK, negativo. Se viene ricevuto un NACK, viene rispedito lo stesso pacchetto. **Nel caso di ricezione pacchetto e ACK positivo, non esegue nessuna istruzione,**

RDT 2.0 receiver

Il ricevitore rdt 2.0, consta di un solo stato; riceve un pacchetto, che se è errato, corrotto, restituisce un NACK. Se il pacchetto ricevuto contiene dei dati corretti, non corrotti, allora estrae i dati, li spedisce al livello superiore e manda indietro un ACK. Resta sempre in attesa di ricevere pacchetti dalla parte corrispondente.

RDT sender 2.0 e RDT receiver 2.0 funzionano bene, sono quindi sincronizzate, solo se il sender non rimane bloccato in stato di attesa di ACK o NACK. Ciò accade perché nello stato di attesa ACK/NACK resta bloccata in attesa appunto che arrivino, ma se ciò non accade rimane bloccata all'infinito.

RDT è affidabile e si occupa dell'interfaccia tra livello applicativo e di trasporto, UTD può non essere affidabile e si occupa dell'interfaccia al livello di trasporto e rete. RDT sta per reliable data transfer. Se l'errore è nella fase di ritorno , il sender rimane bloccato.

RDT 2.1 sender

Ci sono 4 stati, gli stati sono equivalenti a due a due, due stati attendono ACK o NACK, gli altri attendono chiamate dal livello superiore. L'unica differenza tra i due stati di attesa chiamata sono i bit, una chiamata attesa è 0 l'altra è 1. A differenza di RDT 2.0 la macchina va a controllare se il pacchetto che è stato ricevuto è arrivato correttamente oppure no. Se non è arrivato correttamente, rispedisce l'ultimo pacchetto e si rimette in attesa. Invece, se è stato ricevuto un pacchetto ed è corretto, ed in particolare è un ACK del pacchetto, allora va nello stato di transazione. 0 e 1 i, quando usati come in questo caso, servono a distinguere se i pacchetti sono pari o sono dispari.

RDT 2.1 receiver

È costituito da due stati: attesa di pacchetto pari, attesa di pacchetto dispari. Se ricevo un pacchetto non corrotto 0, da 0 si passa a 1, viene spedito un ACK con il suo checksum (è una sequenza di bit che, associata al pacchetto trasmesso, viene utilizzata per verificare l'integrità di un dato o di un messaggio che può subire alterazioni durante la trasmissione sul canale di comunicazione.), e viene spedito il pacchetto all'utente. In caso di errore, ossia un pacchetto corrotto, si spedisce un nak al mittente; può arrivare un pacchetto non corrotto ma con il numero di sequenza sbagliato, significa che ha ricevuto un pacchetto vecchio; tutto ciò che fa è rimettersi in attesa per il pacchetto corretto, bisognerà sbloccare il mittente quindi invierà un ACK.

Possiamo evitare di utilizzare un nak, possiamo sostituire i nak mettendo il bit a cui eravamo interessati negli ACK, ed è una grossa semplificazione.

Abbiamo sinora considerato che il canale sia senza perdite, cioè che i pacchetti o arrivano corretti, o arrivano sbagliati. Però in realtà il canale può perdere qualcosa, abbiamo di norma quindi un canale con perdite e quando vuole introduce degli errori di comunicazione(Lossy Channel with Bit Errors).

Lezione 5 – 20/03/2020

RDT 3.0 Sender

Viene introdotto un timer per stabilire quando rieseguire la ritrasmissione. Dopo che viene mandato un pacchetto, rispetto al modello 2.1, inizia il timer d'attesa per l'ACK. Se tutto filo liscio e arriva l'ACK, possiamo vedere se il pacchetto no né corrotto, se è l'ACK che ci aspettavamo, allora se sono rispettate queste condizioni il timer viene interrotto e passiamo allo stato successivo(da ATTESA ACK ad ATTESA CHIAMATA 1 DA SOPRA), dove attendiamo la ricezione di un pacchetto dall'alto. Altrimenti, se le condizioni non sono favorevoli, ossia se il pacchetto è corrotto, o l'ACK non era quello atteso, rimaniamo ancora in attesa; questa fase non comporta nessun reinvio, si rimane fermi in attesa, poiché abbiamo un timer che è ancora in corso, nel caso che il timer scada il pacchetto viene rispedito e si avvia un timer nuovo. Questa macchina non è perfetta perché se il pacchetto non arriva mai, ci ritroviamo sempre all'interno dei cicli all'infinito. Succede principalmente se non c'è più comunicazione con il destinatario.

RDT 3.0 Receiver

È identico al receiver della versione 2.2.

Facciamo due conti

Un determinato bit occupa un certo lasso di tempo, è un segnale che si sposta lungo il canale, con una velocità pari a 2/3 della velocità della luce. Ogni bit oltre a una loro velocità hanno una certa "larghezza", una certa durata temporale. Supponiamo di avere un canale che viaggia a 10 Mbps, ma il termine corretto è dire : un trasmittente che invia i dati, genera i bit, alla velocità di 10 Mbps. Non è il canale a essere veloce, è il trasmittente che genera i bit a una certa velocità. Supponiamo che il sender spedisca un bit, 10 Mbps significa che ogni secondo il sender genera dieci milioni di bit (10000000), ma se un secondo è costituito da 10 milioni, ognuno di questi bit vorrà dire che hanno la durata di 0.1 microsecondi. Se vengono generati 1500 byte, il tempo totale sarà 1200 microsecondi. Altro esempio. La velocità di propagazione del segnale è all'incirca 2/3 della velocità della luce come sappiamo, ossia 200000 km/s. se la distanza fisica tra mittente e destinatario è di 1km, significa che quando A manda un segnale, affinché l'inizio del bit arrivi a destinazione, ci vogliono 5 micro secondi. Se dovessimo trasmettere una frame, in un tempo di 5 microsecondi potremo generare 50 bit. Per completare la trasmissione, il tempo che ci serve è il tempo che ad A serve per generare tutti i bit + il tempo necessario per farli arrivare tutti alla destinazione B, e quindi 5 micro secondi. Il sistema può essere visto come una cosa, nella quale A inserisce l'informazione e B la estrae. È più corretto definirla pipeline. Più lungo è il canale, maggiore è il ritardo di propagazione.

Stop and Wait

È la tecnica di invio frame e attesa ACK di ritorno. È implicito che si utilizzi il timer. Dato che il round trip time (RTT) è elevato, accade che il sender rimanga fermo per tantissimo tempo.(stessa cosa il receiver)

Dato che la macchina sta ferma in attesa per un certo lasso di tempo, può impiegare il tempo di attesa nel continuare a trasmettere pacchetti. Sfrutta il canale libero, per questo può continuare a farlo, a patto che riesca a identificare la sequenza dei pacchetti, e quindi anche numerare i vari ACK. Questo sistema prende il nome di :

Pipeline

Significa inserire nel canale, nella pipeline di comunicazione pacchetti uno di seguito all'altro. Questo è possibile quando il canale è così lungo, e la frame è piccola rispetto al canale, che posso inserire sul canale più frame consecutive.

Go Back N

Immaginiamo di avere i pacchetti uno di seguito all'altro, anche numerati, distinguiamo tra i pacchetti per i quali abbiamo già ricevuto l'ACK. N è la dimensione della finestra massima di trasmissione, cioè il segmento massimo di spedizione dei pacchetti. Posso spedire N pacchetti, dopo averlo fatto si ferma. Ad ogni elemento analizzato, il range si trova un pacchetto più a destra, come a traslare il limite ogni qual volta che un pacchetto viene analizzato

Go back N sender

È una macchina a stati finiti con un solo stato di attesa.

Go back N receiver

Come la macchina di invio, ma è di ritorno.

Problema: se si perde un pacchetto cosa succede?

Semplicemente si torna indietro e si ritrasmette dal primo pacchetto della nostra N, all'inizio del segmento attuale.

L'utilizzo tra ripetizione selettiva e go bak n, dipende da come sono distribuite le perdite. Se sono perdite sporadiche conviene ripetizione selettiva, se sono concentrate in un certo punto si usa go back n.

Con Go Back N si attiva solo il timer relativo al primo pacchetto, e li ritrasmette tutti a prescindere se li abbiamo correttamente ricevuti o meno(i successivi) , con la ripetizione selettiva si attiva un timer per ogni pacchetto. La ripetizione selettiva ritrasmette solo i pacchetti persi.

TCP - Introduzione

Si occupa di fare l'indirizzamento della porta, il multiplexing. Stabilisce la connessione, la gestisce e la chiude. Si occupa di gestire i dati e di fare il packaging dei dati. Garantisce l'affidabilità e una sorta di qualità del servizio. Gestisce il controllo del flusso e il controllo della congestione. Cerca ovviamente di evitare la congestione, anche se c'è differenza tra controllo del flusso e controllo della congestione.

Cosa non fa TCP

Non si preoccupa troppo delle applicazioni, quindi bisogna mettere delle applicazioni sopra TCP per lavorare meglio. Non è molto sicuro. Non gestisce i limiti dei messaggi, prevede un flusso ininterrotto di byte. Non garantisce la comunicazione.

Lezione 6 – 23/03/2020

Quello che fa TCP è fornire un trasferimento affidabile, controllando il flusso del traffico e controllando la congestione; quello che non fa è garantire che la comunicazione avvenga, fornire i meccanismi di sicurezza, e fornire casistiche specifiche per certe applicazioni.

Il Protocollo TCP

Da un punto di vista del programmatore, il processo che deve mandare dei dati si interfaccia esclusivamente con un modulo con delle funzioni di libreria che utilizzano le socket; le quali si interfacciano con il vero modulo TCP che ha al suo ingresso un Buffer di ricezione. Questo vuol dire che il processo non controlla direttamente la comunicazione TCP tramite le socket, le socket depositano le comunicazioni in un buffer, poi sarà il protocollo TCP a svuotare periodicamente il buffer creando dei segmenti e facendoli arrivare a destinazione.

A destinazione i dati vengono messi in un nuovo buffer e a seconda del funzionamento del sistema remoto vengono passate al processo remoto.

Quindi il tutto può essere immaginato come un qualcosa per cui il processo mittente e il processo di destinazione, sono legati tra di loro da un "tubo" dove vengono inseriti singoli byte senza alcuna suddivisione ben precisa, un flusso di dati tra mittente e destinatario insomma. Il flusso non è regolato dai processi ma è regolato dal protocollo TCP a seconda del funzionamento di tutto il sistema. I dati vengono inviati a blocchi, vengono poi accodati, man mano che transitano potrebbero arrivare al destinatario con uno spezzettamento che non coincide con quello generato dal mittente. Può capitare che il mittente abbia necessità di mettere ordine tra i dati spediti in un qualche modo, questo compito però va fatto eseguire dall'applicativo.

Modello Client-Server

TCP è bidirezionale ed è quindi attivo sia in ricezione, sia in trasmissione contemporaneamente. Si inizia dallo stato closed. Il sistema può mettersi in ascolto oppure può cercare di attivare una comunicazione, ovviamente il primo ad essere attivato può essere la macchina remota messa in ascolto, mettendosi in uno stato di listen, dopo di che rimane bloccato in attesa di ricevere un'informazione dalla controparte. La controparte può inizialmente essere in uno stato di chiuso, e in maniera attiva cerca di attivare la connessione e manda un segnale, in particolare chiamato SYN, alla controparte. Se tutte e due a questo punto sono d'accordo nel continuare, allora la connessione è stabilita, le due macchine sono in contatto fra di loro; a questo punto la comunicazione è perfettamente bidirezionale. Nella fase di chiusura, essa può essere fatta da entrambe le parti, sia quindi da chi ha aperto la connessione, sia chi ha ricevuto la connessione. Di fatto, succede che una delle due macchine decide di mandare il segnale di chiusura e procede, l'altra macchina riceve l'informazione di richiesta di chiusura, fa alcune operazioni, poi di fatto chi ha iniziato la richiesta, entra nello stato di wait, e poi torna nello stato di chiuso.

Formato pacchetto TCP

Non è definita la dimensione, dispone, oltre che di un campo variabili, anche di un campo opzioni, la cui dimensione non è prestabilita, ma è variabile. La dimensione dei campi N.porta sorgente, N.porta destinazione, Numero di sequenza, numero di riscontro, Lunghezza intestazione, finestra di ricezione, checksum, puntatore di dati urgenti, sono di lunghezza fissa, 32 bit; 5 parole da 32 bit. Dovrebbero essere 20 byte, che sono 4 per ogni riga.

Abbiamo il numero di sequenza. Esso è un intero a 32 bit che identifica ogni singolo byte della connessione. Il tutto viene servito come un flusso di byte tra mittente e destinatario. Di fatto dobbiamo avere quindi un identificativo per ogni singolo byte presente nella connessione, per questo hanno deciso di mettere questo intero a 32 bit. Successivamente abbiamo un campo ACK, che è relativo al numero di sequenza in direzione opposta. Il numero di sequenza identifica il primo byte dei dati che vengono spediti, questo vuol dire che (esempio) 100, identifica il primo byte di una sequenza di (esempio) 2048 byte. Se 100 è il primo, l'ultimo sarà 2147, ma non ha senso mandarlo visto che sapendo la posizione del primo, si sa automaticamente la posizione dell'ultimo. L'ACK, che viene spedito indietro, fa riferimento esattamente al numero di sequenza spedito in direzione opposta indicando quel è il byte successivo che il ricevitore è disponibile a ricevere. Di questi 2Kbyte, ricapitoliamo, il primo è identificato con il valore 100, l'ultimo è identificato con il valore 2147; dato che partiamo da 0 l'ultimo è il ValoreTotale – 1. Quindi il ricevente sta dicendo "va bene, li ho ricevuti tutti, vorrei che il prossimo byte da ricevere abbia il numero di sequenza 2148". Infatti nella comunicazione successiva il numero di sequenza inizia con 2148, esattamente quello riportato dall'ACK. Il numero di sequenza identifica il primo byte presente nei dati, il numero di riscontro non è relativo a questa comunicazione ma è relativo alla comunicazione precedente che è legata al pacchetto preso in esame. Ricapitolando, il numero di sequenza identifica i dati che stiamo mandando, ne identifica il numero del primo byte(numero di sequenza del primo byte presente

nei dati), sapendo quanti dati ci sono, si può avere anche l'informazione sul quale sia il valore di sequenza dell'ultimo byte. Il numero di riscontro invece è relativo al numero di sequenza della comunicazione in verso opposto, fa quindi riferimento al pacchetto che è stato spedito al contrario rispetto alla comunicazione. Non sono quindi riferimento tra loro, l'uno con l'altro, ma sono numeri relativi a due comunicazioni differenti. Si usa questa tecnica per mischiare la comunicazioni da A verso B, e da B verso A; in particolare la comunicazione da A verso B trasporta le risposte della comunicazione da B verso A, per questo motivo sono presenti nella stessa intestazione. La Lunghezza dell'intestazione dice quanto è grande la sezione, in valori da 32 bit. Ci sono poi alcuni bit non usati, 6 bit non usati; seguono successivamente una serie di bit, che sono denominati flag, perché sono divisi in bit singoli. In particolare, abbiamo syn, che avvisa che sta per essere avviata una connessione con la controparte, viene settato per due pacchetti, e poi non viene settato più (rimane a 0). FIN è il flag che indica la volontà di chiudere la connessione. RST (reset) è un flag di reset, per resettare in caso succeda qualcosa di non previsto. PSH (push) è un flag che viene inserito per dire di svuotare il prima possibile il buffer di spedizione, viene utilizzato quindi per tutte le macchine che seguono la catena di comunicazione. ACK è un flag che serve per dire che il campo acknowledgment number, è da considerare come valido, altrimenti viene totalmente ignorato, significa che la grandissima maggioranza delle volte, il flag è settato a 1. URG (urgent) è un flag che serve per dire alla controparte che ci sono dei dati urgenti da valutare prima di tutti gli altri; è legato anche allo urgent pointer, presente nell'intestazione. Dato che ho un flusso costante di byte, può capitare a un certo punto che il mittente inserisca delle informazioni che devono essere valutate immediatamente annullando, o bypassando tutto ciò che era stato spedito prima, però se quello che c'era prima non viene valutato dal ricevente, i dati urgenti non vengono mai valutati; avendo previsto questa cosa, viene quindi inserito un contatore dati urgenti, un flag relativo a questo campo, che serve a dire alla controparte di ignorare ciò che veniva prima del dato urgente, fa prendere subito i dati segnalati e si procede a seconda di quello che viene indicato dai dati in questione. Il campo checksum è identico ad UDP come struttura, ma potrebbe non essere controllato durante le varie implementazioni, soprattutto dai router di transito perché, intanto i router di transito non vanno sempre a vedere nel livello di TCP, il problema principale è che il controllo sulla correttezza dei campi viene fatto al livello di Data Link, quindi questo controllo risulta inutile. Il campo window size serve per realizzare il controllo del flusso. Il campo opzioni serve per negoziare la massima dimensione del segmento (MSS) accettabile da parte di tutta la comunicazione. Il checksum controlla tutto, sia l'intestazione IP, sia l'intestazione TCP, sia i dati, e fa alla fine un checksum totale.

Creazione della connessione TCP

La macchina A vuole aprire la comunicazione con la macchina B, se la macchina A è il client la macchina B è il server; il server deve essere già attivo prima che il client comincia a spedire il primo pacchetto. A mette un numero di sequenza, non è importante il contenuto del numero, il campo ACK deve essere relativo alla comunicazione precedente, solo che non c'è stata una comunicazione precedente, quindi in qualche modo deve poter dire all'host di destinazione "guarda che questo campo non lo devi prendere in considerazione", e lo fa con il flag ACK, che

in questo caso viene messo a 0. Nella prima comunicazione viene messo il flag syn a 1. Quindi abbiamo syn=1, ACK=0. L'host B riceve la comunicazione, risponde impostando il campo ACK con il ValoredellaSequenza+1, il numero di sequenza sceglierà un suo numero, e risponde dicendo "ok, anch'io voglio aprire la connessione, però il flag ACK viene posta a 1", per dire che quel campo deve essere preso in considerazione (è un campo valido). La risposta è il numero di sequenza per il ACK precedente, il numero di ACK è il valore del NumeroDiSequenzaPrecedente+1, e il flag ACK ovviamente è posto pari a 1, il flag syn a questo punto è 0. Da questo momento la comunicazione può essere considerata aperta.

Scenario che può capitare

Sia 1, sia 2, vogliono aprire la connessione; queste vengono considerate contemporanee perché la creazione della connessione da 2 verso 1 viene fatta qualche istante prima che arrivi la richiesta da 1 verso 2. Quindi anche se non sono temporalmente coincidenti, vengono considerate coincidenti (nel protocollo) perché non sono collegate tra loro le operazioni, per ciò vengono considerate contemporanee. Viene definito quindi Handshake a tre vie; se fossero 4 vie, abbiamo che, quando l'host 1 riceve una sequenza, dice "io ho chiesto di aprire una connessione, col valore x, la controparte contemporaneamente mi ha chiesto di aprire la connessione con il valore y", questi sono i due valori necessari per aprire la connessione, e risponde con il suo numero di sequenza e ACK+1. Flag syn settato, ACK settato. L'host 2 fa esattamente lo stesso tipo di lavoro, ovviamente invertendo i valori rispetto all'altra connessione.

Handshake a tre vie

A un certo punto può capitare che spunta un messaggio perso nella rete, una vecchia richiesta di creazione della connessione spunta fuori quando in realtà l'host 1 non ha più interesse ad aprire la connessione verso l'host 2. È uno scenario raro, ma è uno scenario possibile. Non possiamo fidarci del fatto che non esistano messaggi che si perdono nella rete e che spuntino improvvisamente senza motivo. Ovviamente l'host 2 non può sapere che questo messaggio è vecchio, non c'è un timestamp dei pacchetti e in ogni caso non avrebbe diritto a negare una richiesta del genere. Quindi risponde correttamente come dice il protocollo, l'host 1 invece rigetta la connessione. Non la apre, informando che l'host 2 non è più disponibile ad aprire la connessione.

Altro scenario che può capitare

Arriva una richiesta vecchia, host 2 tenta di aprire; c'è un altro vecchio messaggio che tenta di aprire una accettazione di richiesta di connessione. Capita quindi che ci sono due richieste col numero di sequenza (esempio) x, il problema sta nel valore dell'ACK, che è relativo al numero di sequenza della connessione precedente. Il vecchio messaggio avrà un numero di ACK diverso dal numero di sequenza x. A questo punto l'host 1 rigetta la connessione.

Viene chiamato Handshake a tre vie perché comporta una sfida, tra la macchina 1 e la macchina 2. La sfida è banale, ed è un qualcosa che è alla base di alcuni protocolli di verifica di correttezza. La sfida banale, è la seguente. "io ti sto mandando un numero x, tu mi devi rispondere con lo stesso numero x oppure con un altro valore che è legato in ogni caso al valore x, per esempio x+1". Questa sfida può essere vinta solamente se questo messaggio

arriva prima della risposta. La sfida diventa particolare quando abbiamo che due messaggi non sono collegati tra loro. Se un messaggio contiene un certo valore di ACK, notiamo che comunque il flag ACK viene settato a 0 per dire "lascia perdere perché quello è un valore che non devi considerare". Quindi i due messaggi che si scambiano gli host non sono legati fra di loro, ma sono totalmente indipendenti. I successivi messaggi sono invece legati ai precedenti presi in esame, poiché sono una diretta conseguenza di questi ultimi. Abbiamo un messaggio da A a B, che una volta ricevuto da B, viene lanciata la sfida. La risposta viene dopo la domanda, infatti il numero di sequenza viene riportato nel valore di ACK, questa è la sfida, quindi sicuramente il messaggio è posteriore. Il messaggio che partito da B arriva ad A, dice "ok, ora mi devi rispondere", lancia un'altra sfida, "ora mi devi rispondere con un valore di ACK che contiene il valore y", la sfida ovviamente deve essere persa, o almeno, viene persa perché nel caso di un messaggio vecchio, non sappiamo da dove esso arrivi, e tra l'altro non è legato al precedente, quindi succede inevitabilmente, appunto, che la sfida viene persa. Non solo arriva un messaggio di reject, ma l'host 2, che si vede arrivare la risposta, capisce che è successo un problema. Per questo host 2 chiude la connessione (la sfida non è stata vinta).

L'apertura funziona in questa maniera. Host 1 sfida Host 2, e Host 2 sfida Host 1. La sfida si definisce tale, perché essa è semplicemente : "tu sei attivo in questo momento, sei presente nella rete e rispondi, oppure i messaggi che stanno girando sono messaggi vecchi, e quindi devono essere ignorati?". Le sfide sono una questione di temporalità della sequenza dei messaggi. Il messaggio tre deve essere posteriore al messaggio due, e il due deve essere posteriore al messaggio uno. Se questa sfida viene vinta vuol dire che 1 e 2 vogliono in quell'istante parlare fra di lor e a questo punto inizia la connessione vera e propria. Il protocollo funziona a una risposta binaria. Apertura della connessione – reject (chiusura) della connessione. Se dovessero esserci problemi di comunicazione ovviamente la connessione banalmente non viene aperta. Se non ci sono problemi di connessione e tutto fila liscio, la connessione viene aperta. E può cominciare il dialogo tra le due parti.

Recap sfida

La sfida è :

Host 1 vuole aprire una connessione con Host 2. E Host 2 risponde. Dato che Host 1 non vede l'Host 2 e Host 2 non vede l'Host 1, ma semplicemente possono scambiarsi il messaggio; chi c'è dall'altra parte? Ci sono messaggi vecchi provenienti da Host 2 che si sono persi? Che spuntano improvvisamente senza motivo? Oppure c'è un reale tentativo di aprire la connessione? Dato che non si vedono, la sfida è "io ti mando un numero, se tu ora sei presente, sei online, allora mi devi rispondere con un valore legato al numero che ti sto mandando". Poi la sfida viene invertita. L'Host 2 sfida l'Host 1 a fare la stessa operazione. Qui non è un problema di sicurezza, è un problema solo per garantire che vecchi messaggi che circolano nella rete per qualunque motivo, che non ci interessa, non vadano ad aprire connessioni, quindi a occupare risorse in maniera inutile. Host 2 apre la connessione solamente se la sua sfida, quella che manda verso Host 1, ha successo.

Chiusura della connessione

Immaginiamo uno scenario particolare ma interessante. Abbiamo due eserciti, uno blu e uno bianco, che devono combattere fra di loro, solo che il problema, è che l'esercito blu è diviso in due parti, l'esercito bianco è solo concentrato al centro. Il problema è che le due mezze armate si devono sincronizzare fra di loro, perché se non riescono a sincronizzarsi, sicuramente perdono, ma insieme vincono. È un problema esclusivamente di sincronia tra le due parti. Per vedere questa sincronia basta mandare un messaggio dalla parte di sinistra alla parte di destra, e viceversa. Una volta che il messaggio è stato spedito, non sappiamo se la controparte lo riceve. Se la comunicazione fosse perfetta, non ci sarebbero problemi perché basterebbe un solo messaggio, dalla parte sinistra alla parte destra, e la sincronizzazione avverrebbe tranquillamente. Ma in un sistema reale il messaggio può perdere, la risposta del protocollo può essere un messaggio di conferma che dice : "Ho ricevuto il tuo messaggio principale". Però anche la conferma si può perdere, a questo punto chi ha mandato il messaggio, non vedendo la conferma, non sa qual è lo scenario corretto : se la controparte ha ricevuto, e quindi ha spedito la risposta di conferma, oppure il messaggio originale è stato perso, non ha nessuna idea di che cosa è successo realmente. D'altra parte, la controparte vede arrivare il messaggio, manda la risposta, ma anche la controparte sa che la sua risposta si può perdere, e può avviare un ragionamento su cosa pensa di fare la parte sinistra. Non sa se la parte sinistra ha ricevuto il messaggio e come interpretare la mancanza di una risposta. Ma anche se la risposta dovesse arrivare, quindi la richiesta è partita; arriva la risposta; questa parte dovrebbe essere tranquilla. E invece no, perché anche questa parte fa il ragionamento della controparte, ma essa non sa che la risposta è stata ricevuta. Quindi manda un altro ACK di conferma. Possiamo immaginare un protocollo fatto da una serie del tipo messaggio-risposta-messaggio-risposta costituiti soltanto da conferme? Ovviamente sarebbe un lavoro infinito. Se il messaggio è importante manda una conferma, altrimenti viene scartato e l'ultimo messaggio importante, prima di quest'ultimo (ossia il suo posteriore) diventa il messaggio importante. Non c'è una soluzione completa e definitiva a questo problema. Come soluzione, il client chiude la connessione, manda il segnale di finalize, e il server risponde con un ACK. Nonostante si possa concludere qua, perché basta un solo ACK, TCP si comporta diversamente. La chiusura è unidirezionale, rispetto all'apertura che invece è bidirezionale. Nel senso che, possiamo chiudere la connessione da una macchina verso l'altra, e tenere aperta la connessione dati nel senso opposto. Questo vuol dire che se il client chiude la connessione verso il server, il server può continuare a spedire dati al client, e ottenerne le relative risposte. Chiudere la connessione non significa non inviare più niente, non si mandano più pacchetti contenenti dati, ma continuano a scambiarsi pacchetti contenente informazioni, tipo gli ACK. Questa è la procedura classica di chiusura. Dopo la chiusura di entrambe le macchine, la prima si mette in attesa con un certo timer, allo scadere del quale la connessione viene chiusa del tutto. Si usa il timer per un motivo ben preciso. Capita che la prima macchina mandi un ACK che si perde, la seconda macchina avvia il timer, in attesa di ricevere l'ACK, se non lo riceve, immagina che la connessione sia andata persa, se entro il timer non arrivano ulteriori messaggi, di fatto la connessione è chiusa.

MTU e MSS

MTU sta per Maximum Transfer Unit, ed è la dimensione massima di byte accettati dal livello inferiore per la trasmissione. Quindi, TCP bufferizza, di conseguenza il generatore di informazioni dell'applicativo manda una sequenza di byte che vengono messi nel buffer, TCP li prende a blocchi e li invia nella rete. L'MTU è quello che dice "è inutile che vai a prelevare per esempio 2000 byte, perché al livello inferiore più di 1500 non puoi spedire"; quindi se vuoi li prendi a blocchi di 1500, verranno poi scaricati nel suo buffer di destinazione, e poi la destinazione li prende come vuole. Per esempio a blocchi di 100, a blocchi di 2000 se ci sono, e così via. L'MTU è importante ed è un valore che viene passato dal livello inferiore al livello IP, o comunque TC; basta dire come vengo interpretati i dati dai buffer sulla rete. L'MTU viene comunicato, di norma, dalla scheda di rete. Per esempio ethernet comunicherà 1500 byte. E se si dovesse attraversare una rete con un MTU più piccolo, per cui il pacchetto non può più viaggiare? Allora è prevista la possibilità di frammentare i singoli pacchetti, in segmenti più piccoli. Ovviamente il preamble deve essere replicato perché deve essere comune a tutti, però con alcune accortezze, per poter gestire la comunicazione. La frammentazione dato che replica il preamble, ovviamente comporta uno spreco di banda. L'altra cosa importante da dire per quanto riguarda la gestione dei dati, è che di base il protocollo TCP utilizza un meccanismo di sliding window, per gestire il flusso di dati. Ma a differenza di quello visto prima, TCP utilizza gli ACK cumulativi. Ossia tutti i byte spediti dall'inizio di una comunicazione fino a un determinato momento. Se mandiamo due messaggi, ma il primo si perde e arriva solo l'ACK del secondo, non rappresenta un problema perché la conferma del secondo implicitamente conferma pure il primo. È un vantaggio l'ACK cumulativo, perché permette di semplificare queste operazioni. Tuttavia non permette di implementare il meccanismo della ripetizione selettiva. Con essa avremmo potuto dire in modo preciso quali pacchetti sono andati perduti, così invece si può dire che sono arrivati tutti i pacchetti sino ad un certo punto.

Finestra di Ricezione e Finestra di Trasmissione

I dati che arrivano dalla controparte, vengono messi su un buffer, e vengono svuotati a seconda del processo di destinazione, in particolare a seconda di come è disposto a riceverli. Il processo di destinazione segue una chiamata di lettura per svuotare i buffer, se non arriva la chiamata, ovviamente i buffer li mantiene. Per evitare di mandare i pacchetti inutili, ovviamente conviene far mandare dal destinatario delle informazioni per dire quanto è disponibile ancora ricevere.

Lezione 7 - 25/03/2020

Finestra di ricezione

Quando i dati arrivano a destinazione devono essere memorizzati da qualche parte, il sistema operativo ricevente predisponde una zona di memoria, un buffer di memoria; dove poter depositare le informazioni in attesa di essere elaborate, passate all'applicazione ricevente.

Questo spazio ovviamente è di dimensioni limitate, ma soprattutto se non viene svuotato, l'applicazione ricevente si va a riempire. Quindi il ricevente sempre con meccanismi di Piggybacking tipo ACK, comunica al mittente la sua finestra di ricezione, RcvWindow, per dire "ho questo spazio di ricezione, quindi puoi mandare dati massimo per questa dimensione"; non è possibile mandare dati oltre la dimensione perché semplicemente non c'è lo spazio per la memorizzazione.

Una cosa interessante da aggiungere è il cosiddetto Algoritmo di Nagle, a cosa serve? Tornando all'esempio di Telnet, viene mandato un byte, senza informazioni; mandare un pacchetto solamente per un byte rappresenta uno spreco di banda perché semplicemente al livello TCP/IP per un byte dobbiamo spedire 20 byte di intestazione TCP e 20 byte di intestazione IP; quindi 40 byte di intestazione solo per TCP/IP, solo per mandare un byte. Questo fa sì che ci siano una marea di pacchetti ogni volta che viene premuto un tasto.

Per evitare questo, Nagle propose il seguente algoritmo.

Algoritmo di Nagle



Algoritmo di Nagle (RFC 896)

Il telnet produce pacchetti di dimensione minima (1 byte) con spreco di banda.

L'algoritmo di Nagle tenta di ridurre l'overhead, bufferizzando i dati da trasmettere.

```
if available_data > 0 then
    if window_size ≥ MSS & available_data ≥ MSS then
        send_a_MSS_segment
    else
        if waiting_for_an_ack == true then
            enqueue_data /* until an acknowledge is received */
        else
            send_data
        end if
    end if
end if
```

Se ci sono dati disponibili da essere spediti, quindi in pratica dopo aver inviato il pacchetto ho continuato a premere tasti sulla macchina A, allora se la dimensione della finestra di ricezione è maggiore del Maximum Segment Size, cioè ho più dati di quelli che posso infilare in un pacchetto, e i dati disponibili sono più di quelli che posso inserire in un pacchetto, allora spedisco il pacchetto.

Cioè se lo spazio disponibile è maggiore della lunghezza massima di un segmento, e i dati che ho a disposizione non ancora inviati della macchina A sono maggiori di quelli dello spazio di un segmento, allora li invio, perché di fatto sto sfruttando al massimo il pacchetto, l'idea è questa.

Altrimenti, waiting_for_an_ack, cioè; hai spedito questi dati, li accumuli, quindi non continui a spedire pacchetti, fino a che non ricevi un ACK, solo a questo punto avrai un pacchetto abbastanza corposo da potere spedire e mandare a destinazione.

Qual è l'idea di questo algoritmo? Semplicemente non spedire un pacchetto per ogni tasto che è stato premuto, ma se ancora non hai ricevuto un ACK, accumula i byte che hai da spedire e appena possibile li spedisci.

Questo ha un effetto molto deleterio nell'interattività delle macchine.

Quello che succede: dato che con questo protocollo Telnet, per esempio, quando premo un tasto, il tasto poi viene messo a video solamente quando ricevo di ritorno lo stesso carattere, quello che succede è che premo un tasto, viene spedito un pacchetto, dopo

di che tutti i tasti successivi vengono accodati e verranno mandati per esempio con questo pacchetto e torneranno indietro con un altro, quindi l'effetto visivo sarà che io scrivo e non vedo un tasto premuto ogni volta che digito un carattere, ma li vedo comparire a blocchi consecutivi.

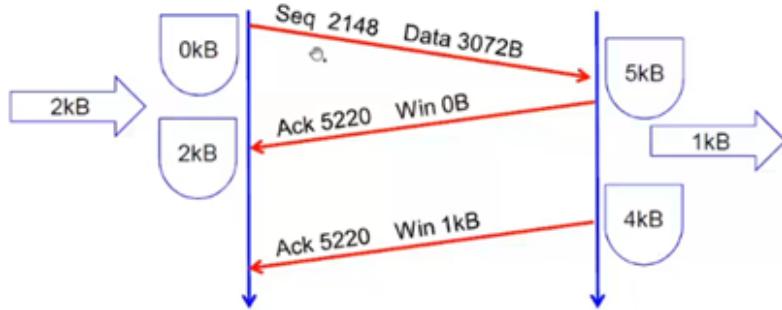
Questo ovviamente fa sì che, da un lato miglioriamo le prestazioni del sistema perché non spediamo pacchetti troppo frequentemente, d'altro canto però l'interattività dell'utente è abbastanza scarsa perché non vede subito quello che sta facendo.

Ci sono alcuni applicativi, tipo terminale remoto grafico, per cui un ritardo di questo tipo è assolutamente deleterio, in questo caso è possibile disattivare l'algoritmo di Nagle da parte dell'applicativo, addirittura c'è un'avvertenza di Microsoft Windows 10 che dice di disattivare l'algoritmo di Nagle perché soprattutto per i giochi, da' una scarsa interattività.

Bisogna però aggiungere anche un altro discorso. La bufferizzazione dipende da quanto dura il tempo tra un invio e un altro, quindi se siamo in rete locale, dove praticamente questo tempo è piccolo, perché per raggiungere la destinazione basta pochissimo, allora l'algoritmo di fatto non ha neanche il tempo di entrare in funzione, per cui quello che succede è che, se l'RTT è basso, di fatto l'algoritmo di Nagle spedisce pacchetti con elevata frequenza, mentre se l'RTT è elevato, cioè il tempo di andata e ritorno è molto elevato, allora la bufferizzazione comincia a funzionare. L'algoritmo comunque è disattivabile dall'applicativo.

Finestra di ricezione e ACK

Finestra di ricezione e ACK



Se il messaggio di «sblocco» dovesse perdere, il sistema rimarrebbe bloccato.

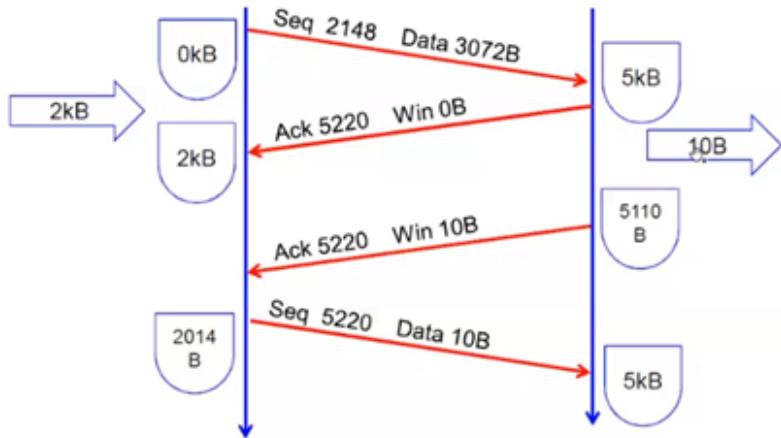
Il sender, ricevuto il pacchetto con `recv_wnd = 0` avvia un timer (*Persistent Timer*), allo scadere del quale invia un pacchetto vuoto di test, per sollecitare una risposta.

Ho mandato un messaggio, il buffer si è riempito, ho la finestra di ricezione chiusa. Dopo di che abbiamo detto, appena la finestra si riapre, il destinatario manda un messaggio con lo stesso ACK, indicando semplicemente che la finestra si è aperta. Che succede se dovesse perdere questo messaggio? Tutti e due sono bloccati, perché il ricevente è messo in ricezione, il mittente non può spedire dati, quindi il sistema è bloccato. Per evitare questo esiste un altro timer TCP denominato Persistent Timer, che viene attivato quando viene ricevuto il messaggio di chiusura della finestra, l'ultimo messaggio di chiusura della finestra. Il timer fa sì che se ad un certo punto il mittente non riceve più messaggi allora manda un messaggio di sveglia, di sblocco, al destinatario mandando le stesse informazioni, quindi lo stesso numero di sequenza, però senza dati. (RWND=0)

Essenzialmente per dire : "Sei ancora vivo?", in attesa di una risposta. È un modo per sollecitare il destinatario. Se non dovesse arrivare niente viene fatto un po' di volte, se non dovesse arrivare nulla allora, così come accade per la chiusura della connessione, il mittente considera interrotta la comunicazione. Quindi presuppone che ci sia stato un problema più serio.

Altro problema sempre per quanto riguarda la ricezione chiamato Sindrome della finestra stupida. Invece di prendere 1 Kb, si prendono solamente 10 Byte,

Silly window syndrome



La «[soluzione di Clark](#)» prevede che il ricevente non aggiorni subito la sua *receive_windows* in caso di piccole variazioni.

Quindi passo da 5 k a 5110 byte occupati nel campo di ricezione. Mando un messaggio, ovviamente il mittente può spedire solamente un pacchetto di 10 byte, cosa abbastanza stupida, diventa ancora più stupida se invece di essere 10 byte, viene consumato un Byte per volta, avrei sempre messaggi di finestre microscopiche da fare andare avanti. Per evitare questo discorso la soluzione è abbastanza banale e dice : “non aggiornare la finestra nel messaggio di risposta se devi comunicare valori microscopici, aspetta che il buffer sia svuotato in maniera seria, quindi non mandare 10 byte, ma per esempio aspetta di avere il buffer svuotato di 1k prima di mandare una informazione di questo tipo”.

In ogni caso il sistema non va in timeout, non si blocca, perché, se manda messaggi con Win 10; dice al mittente “sono ancora vivo, semplicemente l’applicativo è bloccato”, quindi non ci sono problemi, “puoi andare avanti tranquillamente, appena si sblocca la situazione te lo faccio sapere”.

Quindi anche per questo problema c’è una soluzione abbastanza semplice, da implementare.

Round Trip Time

Abbiamo detto, io mando un pacchetto, attendo la risposta, quando mando il pacchetto avvio un timer. Se scade il timer rispedisco lo stesso pacchetto. Perché serve una stima buona del timer? Perché se metto un timer prefissato alto, sicuramente riduco il numero di pacchetti che girano, ma rallento molto la connessione, quindi il risultato per quanto riguarda l'utente finale è: si perde un pacchetto poi aspetto mezz'ora prima di poter andare di nuovo avanti. Mezz'ora si fa per dire, il tempo massimo prefissato è 1 secondo, non si dovrebbe andare oltre il secondo.

Un secondo è troppo, perché devo anche ricordarmi che non so dove si trova l'host B, può essere nella mia stessa sottorete, quindi mettere un secondo è pura follia, l'host B potrebbe essere dall'altra parte del mondo e allora un secondo potrebbe essere un tempo corretto.

Se sono nella mia stessa LAN, mettere un tempo adeguato per il timer significa avere anche una buona interazione con il destinatario, anche nel caso di perdita di pacchetti.

Quindi conviene fare una stima. Il problema è : "come faccio questa stima?". Ovviamente la stima la posso fare solo dopo che ho cominciato ad eseguire il dialogo, tra A e B. fino a che non ho informazioni su come sta andando la comunicazione tra A e B, non posso fare nessuna stima.

Quindi devo cominciare a raccogliere i dati. Come posso fare la stima? Un modo banale, potrebbe essere : mando un pacchetto, avvio il timer, vedo quando arriva la risposta, ho misurato il tempo di andata e risposta del messaggio, con il suo ACK. Però, lo faccio una volta, due volte, tre volte, questi dati come li peso fra di loro? Potrei fare banalmente la media temporale, ma fare la media temporale non è la soluzione migliore perché, per fare una media di questo tipo devo sprecare delle risorse. Non solo, quello che è successo all'inizio della comunicazione può non essere più significativo perché nel frattempo le connessioni della rete sono cambiate, in meglio, o in peggio. La soluzione adottata da TCP è la seguente. **EWMA** (Exponential Weighted Moving Average), pesata, esponenziale, mobile. Funziona in questa maniera.

 UNIVERSITÀ
degli STUDI
di CATANIA

Tempi di Round-trip

Avere valori corretti per il timer consente di migliorare le prestazioni e diminuire la congestione di rete.

TCP effettua una stima dei tempi di Round Trip per fissare un valore di timeout.

Si usa una **EWMA** (Exponential weighted moving average)

$$\text{EstimatedRTT} = (1-\alpha) \cdot \text{EstimatedRTT} + \alpha \cdot \text{SampleRTT}$$

Tipicamente $\alpha = 0.125$

Io prendo un campione misurato, l'ultima accoppiata messaggio, ACK di ritorno. Questo lo peso con un fattore alpha, e lo vado a sommare alla stima precedente pesata a sua volta con un fattore 1-alpha. A cosa serve fare questa cosa?

Prendiamo il campione N; esso quando viene misurato, viene pesato con fattore alpha, al passaggio successivo questo campione N, sarà messo in **EstimatedRTT**, e sarà pesato con un fattore 1-alpha. Al passaggio successivo, questo campione N, che già era stato pesato con alpha, viene ridotto ancora di un altro fattore 1-alpha. Quindi quello che succede è che il contributo di questo campione, sarà alpha, numero minore di 1; viene ridotto di un fattore con un numero minore di 1 varie volte fino ad avere ad un certo punto un contributo praticamente nullo. Questa media per come è combinata, tiene maggiormente conto di quello che è successo di recente piuttosto che non quello che è successo nel passato remoto. Non è che se lo dimentica, tiene conto di tutta la storia passata di ogni singolo elemento della media, ma man mano lo considera sempre di meno, pesato in maniera esponenziale con un esponente minore di 1. Quello che mi interessa, è andare a vedere quella che è la distanza tra la media e ogni singolo campione misurato, ed è quello che facciamo con un'altra stima.



Tempi di Round-trip

TCP effettua anche una stima sulla variabilità dei tempi di Round Trip.

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

β

Tipicamente $\beta = 0.25$

$$\text{RTO} = \text{EstimatedRTT} + 4 \cdot \text{DevRTT}$$

Il **valore iniziale** raccomandato per RTO è 1 secondo (RFC 6298)

In caso di timeout, il nuovo timer viene settato al doppio del valore precedente.

I timer crescono quindi esponenzialmente in caso di timeout.

Si prende in considerazione la distanza tra il valore stimato e il singolo campione. Questa stima viene moltiplicata per un fattore beta, sempre minore di 1, e sommata alla stima precedente moltiplicata per

1-beta. È pesate perché la somma, tra **DevRTT** più **beta**, da' 1. Il valore del timer **RTO** viene calcolato in questa maniera. Prendo la media stimate **EstimatedRTT**, e sommo 4 volte la deviazione calcolata **DevRTT**.

Timer

TCP, quanti timer avvia? La logica vorrebbe, per ogni pacchetto, metto su un timer. Solo che in questa maniera il sistema operativo starebbe solamente a guardare timer, in continuazione. TCP utilizza un solo timer, anche se ci sono più pacchetti spediti, non ancora confermati. Appena viene la conferma, viene lanciato il timer successivo.

Lezione 8 - 27/03/2020

PRIMA PARTE

EWMA

“Dicevamo” (La registrazione è iniziata 10 minuti dopo l'inizio effettivo della lezione)
Avere una sistemazione dei timer opportuna consente di limitare ritrasmissioni inutili a causa di Ack in ritardo e quindi migliorare traffico in rete.

Si utilizza l'EWMA (Media Esponenziale Pesata Mobile). Indica le varie formule nelle slide:

(ho fatto del mio meglio senza Latex, grazie Unicode)

$$\alpha = 0.125$$

S_{RTT} = Sample RTT (Valore reale)

E_{RTT} = Estimated RTT (Valore stimato)

$$E_{RTT_{n-2}} = (1-\alpha) \cdot E_{RTT_{n-3}} + \alpha \cdot S_{RTT_{n-2}}$$

$$E_{RTT_{n-1}} = (1-\alpha) \cdot E_{RTT_{n-2}} + \alpha \cdot S_{RTT_{n-1}}$$

$$E_{RTT_n} = (1-\alpha) \cdot E_{RTT_{n-1}} + \alpha \cdot S_{RTT_n}$$

$$E_{RTT_n} = (1-\alpha) \cdot ((1-\alpha) \cdot E_{RTT_{n-2}} + \alpha \cdot S_{RTT_{n-1}}) + \alpha \cdot S_{RTT_n}$$

$$E_{RTT_n} = (1-\alpha)^2 \cdot E_{RTT_{n-2}} + \alpha(1-\alpha) \cdot S_{RTT_{n-1}} + \alpha \cdot S_{RTT_n}$$

$$E_{RTT_n} = (1-\alpha)^2 \cdot ((1-\alpha) E_{RTT_{n-3}} + \alpha \cdot S_{RTT_{n-2}}) + \alpha \cdot (1-\alpha) \cdot S_{RTT_{n-1}} + \alpha \cdot S_{RTT_n}$$

$$E_{RTT_n} = (1-\alpha)^3 \cdot E_{RTT_{n-3}} + (1-\alpha)^2 \cdot S_{RTT_{n-2}} + \alpha \cdot (1-\alpha) \cdot S_{RTT_{n-1}} + \alpha \cdot S_{RTT_n}$$

$$E_{RTT_n} = 0,6699 E_{RTT_{n-3}} + 0,0957 S_{RTT_{n-2}} + 0,1094 S_{RTT_{n-1}} + 0,125 S_{RTT_n}$$

Il valore in blu si sostituisce con quello in blu sopra. Si svolgono i calcoli (ad esempio $(1-\alpha) \cdot (1-\alpha)$ fa $(1-\alpha)^2 \dots$)

Con $\alpha = 0.125$ viene fuori l'espressione in fondo.

La stima al passo 'n' è pari alla **stima** (estimated) al passo 'n-3' (storia passata pesata con il valore 0.667) + I valori **campionati** (sample) il penultimo è pesato per $\frac{1}{8}$, n-1 è pesato $1/10$, il valore n-2 è pesato per "meno di un decimo", aggiungendo un passo, andando al passo n+1 tutti i valori saranno moltiplicati per $1-\alpha$, quindi scenderanno ancora.

Man mano che i valori vanno indietro nel tempo il loro contributo diventa sempre meno inferiore, dato che vengono tutti sempre moltiplicati per un valore $1 - \alpha$ il loro peso è esponenzialmente sempre di meno.

Per questo si chiama "Media Esponenziale Pesata Mobile".

Se ci sono forti varianabilità mi interessano i valori più nuovi, non i più vecchi.

$$\text{DevRTT} = (1 - B) \cdot \text{DevRTT} + B \cdot |\text{SampleRTT} - \text{Estimated RTT}|$$

$$B = 0.25$$

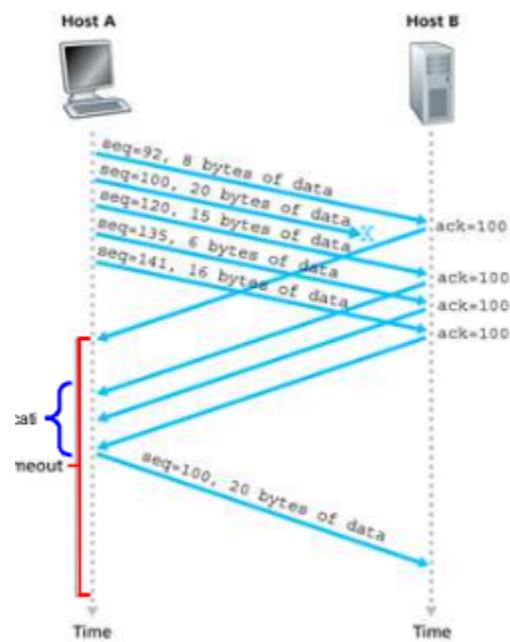
$$\text{RTO} = \text{EstRTT} + 4 * \text{DevRTT}$$

Con questa espressione il valore del timer è molto sovrastimato. Così si evitano di sicuro ritrasmissioni inutili.

Questo significa che in caso di perdita c'è un forte rallentamento e bisogna aspettare che il timer scada prima di proseguire.

Per evitare di bloccare il sistema a causa del timer troppo grande c'è un altro meccanismo, perché a volte potrebbe convenire "metterci una pezza".

Fast Retransmit



Quando un destinatario riceve vari pacchetti manda l'ACK con numero di sequenza dell'ultimo pacchetto ricevuto.

In questo caso si è perso un solo pacchetto però, segnalato dal fatto che all'Host A sono tornati 3 ACK duplicati. Da notare che quando arriva l'ACK duplicato in questione è impossibile sapere quanti pacchetti si sono persi dopo il primo duplicato arrivato. Invece di aspettare tutto il tempo di timeout (in rosso) ritrasmetto il pacchetto perduto, in modo tale da poter avere dopo la ritrasmissione un nuovo riscontro, nel migliore dei casi con numero di sequenza cumulativo (cioè il caso che si è perso un solo pacchetto nel mezzo della comunicazione e tutti quelli dopo sono arrivati correttamente, evitando così ulteriori ritrasmissioni).

Quindi, da un lato attivo il timer come sempre, dall'altro se capitano 3 ACK duplicati posso anche immaginare che sia successo un problema momentaneo e allora rimando il pacchetto perso, nella speranza di recuperare velocemente tutto.

Non è un meccanismo di ripetizione selettiva ma ci assomiglia. È limitato solamente al primo pacchetto perso.

Alla fine viene spedito solo l'ACK cumulativo di tutti i pacchetti arrivati dopo quello perso (Facendo riferimento alla figura, arriva ACK=157 direttamente, **non** ACK=120, ACK=135, ACK=141).

Un pacchetto in ritardo potrebbe essere considerato come perso ma Fast Retransmit funziona anche in questi casi

Domanda: "Perchè 3 ACK"

Risposta: "Perché no?"

"1 è troppo poco, 2 poteva anche andare bene ma 3 sembrava meglio.
4 troppi"

Domanda: "Una volta che si recupera il pacchetto perso il timer viene resettato?"

Risposta: "Quando si rispedisce il pacchetto perso il timer viene resettato, il nuovo timer parte col nuovo pacchetto. D'altro canto è l'ultimo spedito quindi è logico che venga resettato".

"Il problema del fast retransmit è più complicato di come vi sto descrivendo, ma è troppo complicato da spiegare, quindi il Kurose e anche io vi facciamo una versione semplificata perché è quello che vi serve sapere. Prima di spedire l'ACK duplicato succede qualcos'altro"

Domanda: "E se si hanno pacchetti 'alternati'? (uno perso uno arrivato uno perso...)"

Risposta: "Non fa parte del corso. Preso in considerazione da TCP New Reno."

Controllo del flusso vs Controllo della Congestione

(Slide del rubinetto, you know the one [n98-Trasporto6.pdf], non ne vale la pena di metterla. Si riferisce alla parte a sinistra)

Fast Retransmit e Timer sono stati pensati per migliorare la gestione della rete. Nel senso che questo meccanismo evita che il timer troppo grande fermi la connessione e si cerca di riprendere la comunicazione il prima possibile generando un solo pacchetto ritrasmesso in più rispetto ad avere la scadenza del timer.

Un timer efficiente calcolato bene evita anche le trasmissioni inutili di pacchetti che sono solo in ritardo ed evitano di mettere timer fissi troppo grandi che porterebbero come detto poc'anzi a ritardi nella comunicazione.

Non sono direttamente legati alla gestione della congestione ma di fatto servono anche a migliorare / evitare di arrivare troppo presto alla congestione.

Controllo del flusso: Evitare di far viaggiare troppi pacchetti se a destinazione non c'è lo spazio per memorizzarli accuratamente. (Buffer di Ricezione che viene svuotato dall'applicativo). Non è un problema se il buffer è piccolo se l'applicativo lo svuota continuamente senza problemi.

L'informazione sul buffer di ricezione è limitata dal ritardo di comunicazione tra mittente e destinatario, in quanto è inserita in un pacchetto.

Quindi, quando viene mandato dal destinatario il pacchetto con l'informazione "Il buffer è pieno", ci sono ancora dei pacchetti in viaggio nel canale che sono già stati spediti dal mittente, e se il buffer è pieno (e non viene svuotato in tempo, magari per fortuna qualche pacchetto passa e allora vengono mandati ACK duplicati), tutti questi pacchetti verranno buttati via.

Il controllo del flusso, pur essendo molto robusto e logicamente corretto, **soffre** dalla limitazione fisica della **distanza** tra mittente e destinatario (Il problema descritto sopra si manifesta in modo molto pesante nelle comunicazioni da un capo del mondo all'altro). In questi casi non basta il controllo del flusso e quindi non può che essere considerato "Una pezza al problema" invece di una vera e propria soluzione.

Il **problema del controllo del flusso** è che il destinatario non ce la fa a smaltire i pacchetti che ricevono in tempo, e allora il mittente va informato di smettere, momentaneamente, di trasmettere. Perché altrimenti i pacchetti vanno buttati. L'unico sistema che abbiamo è di usare il canale di comunicazione per dire al mittente "smettila che il ricevente non ce la fa".

Non è perfetto perché se c'è un forte ritardo tra i due si perdono tutti i pacchetti in viaggio mandati dopo la richiesta di pausa. Questi pacchetti genereranno tutti

timeout nel mittente, ma il mittente è bloccato e quindi dei timeout se ne frega perché non ha più il permesso di spedire.

Definizione di Congestione

(Si riferisce sempre alla slide del rubinetto, ma la parte a destra)

Il ricevitore ha una capacità large ma il canale ha una “strozzatura”. Il traffico arriva, ma arriva piano piano.

È una congestione o un rallentamento? **Le due cose sono molto differenti.**

La congestione non è rallentamento.

[Qui metterò il riassunto di circa 4 ore di spiegazione, perché per ora ha detto solo che la congestione è complicata]

Tentativo di definizione n1:

“Il Controllo di Congestione è un meccanismo end-to-end che tenta di capire cosa succede nel canale, cosa che (le macchine che comunicano) non possono fare, sperando che le decisioni prese influiscano a livello di rete.
Questo è il problema grosso del controllo della congestione.”

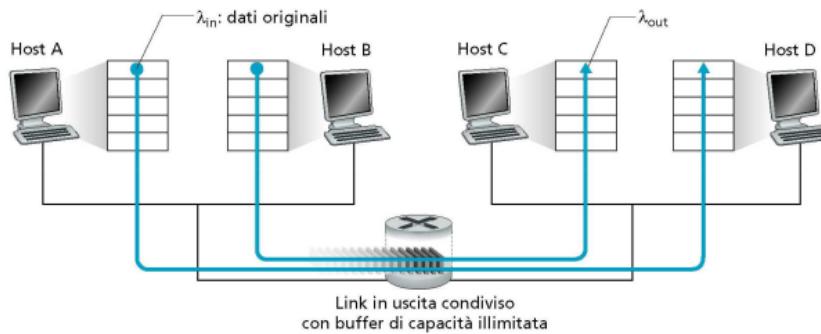
Tentativo di definizione n2:

“La **congestione** si ha quando il traffico di livello di trasporto è fatto da pacchetti ritrasmessi all’infinito” (che non contribuiscono affatto alla comunicazione ndr)”

Tentativo di definizione n3 (dalla lezione 9):

“La congestione è dovuta al fatto che ci sono tanti pacchetti ritrasmessi e pochi pacchetti nuovi che arrivano a destinazione. Cioè la rete è piena di pacchetti (inutili) che sono copie, di copie, di copie...”

Primo Scenario: buffer illimitato



Ci sono due comunicazioni, A<->C e B<->D. Tutti i dati di queste due comunicazioni vengono scambiati tramite un collegamento condiviso.

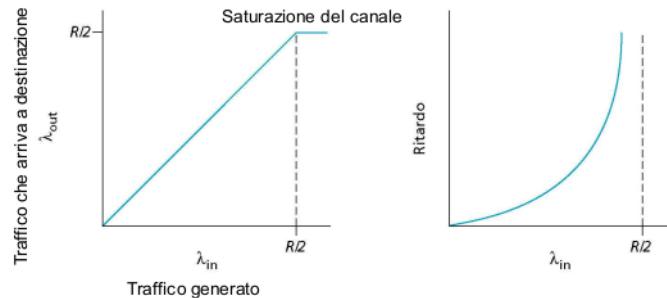
La prima ipotesi è che il router in mezzo ha un buffer infinito. I pacchetti non possono perdersi per capienza massima.

Il link in uscita ha comunque una larghezza di banda, quindi i pacchetti vengono comunque accodati se vengono mandati troppo velocemente.

In pratica, il buffer è infinito, ma l'uscita è limitata. I pacchetti potrebbero quindi essere accodati all'infinito.

Questo è un problema di teoria dei grafi. Ecco la soluzione:

Immaginando che le due connessioni si suddividano equamente la banda a disposizione (R) avremo:



Supponendo che le connessioni siano “intelligenti”(???) si suddividano equalmente la banda a disposizione

λ_{in} : Traffico generato

λ_{out} : Traffico che arriva a destinazione

Fino a quando siamo al di sotto di $R/2$ (cioè metà della banda a disposizione perché se la stanno dividendo) allora tutto il traffico generato arriva in uscita senza problema alcuno.

Quando il traffico in ingresso diventa superiore alla larghezza di banda in uscita allora λ_{out} si ferma al valore massimo.

λ_{in} può comunque crescere ma se dovesse farlo allora i pacchetti verranno accodati all'infinito. Ciononostante in un tempo infinito usciranno dal buffer.

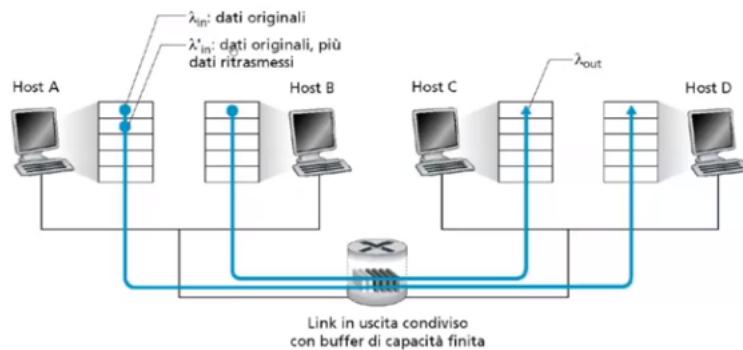
Finché il traffico generato è inferiore al traffico in uscita funziona, seppur con ritardo enorme di comunicazione.

BENE QUESTA NON È CONGESTIONE È SOLO UN SEMPLICE RITARDO.

Si hanno ritardi enormi ma non è congestione.

=====SECONDA PARTE (con registrazione)=====

Secondo Scenario: buffer reale



Abbiamo:

λ_{in} : Traffico generato a livello applicativo

λ'_{in} : Traffico generato a livello applicativo + livello di trasporto

Questi dati non coincidono.

Infatt $\lambda'_{in} = \lambda_{in} + \text{dati ritrasmessi}$.

$\lambda'_{in} \geq \lambda_{in}$ Sempre.

Saranno uguali se non ci sono ritrasmissioni. Se ci sono dati che vengono ritrasmessi a causa di una perdita allora λ'_{in} sarà maggiore.

Queste ritrasmissioni avvengono perché adesso il buffer è limitato e quindi dei pacchetti verranno buttati nel caso in cui si dovesse riempire.

Il tempo di transito per un pacchetto che ha la fortuna di trovare uno slot libero nel buffer ha un massimo ben preciso che dipende dalla:

- Grandezza del buffer
- Velocità di elaborazione del router.

Non c'è più il problema del ritardo infinito.

“Paghiamo” questo tetto massimo al ritardo di comunicazione con delle ritrasmissioni (aka spreco di banda). I tempi di transito sono finiti.

I pacchetti duplicati non sono utili per il λ_{out} , anche se il traffico a livello di trasporto aumenta, il traffico a livello applicativo si ferma perché non ci sono più nuove informazioni utili.

La **congestione** si ha quando il traffico di livello di trasporto è fatto da pacchetti ritrasmessi all'infinito che non contribuiscono affatto alla comunicazione.

L'Applicativo continua a mandare ACK duplicati dicendo che il pacchetto è arrivato, ma il sistema continua ad andare in timeout, e quindi a rispedire pacchetti vecchi, e man mano che la coda si riempie di duplicati l'applicativo non riesce ad andare più avanti. La congestione quindi può arrivare al punto critico di alimentare se stessa.

Una situazione dove quindi $\lambda'_{in} \gg \lambda_{in}$ e λ_{out} è prossimo allo zero.

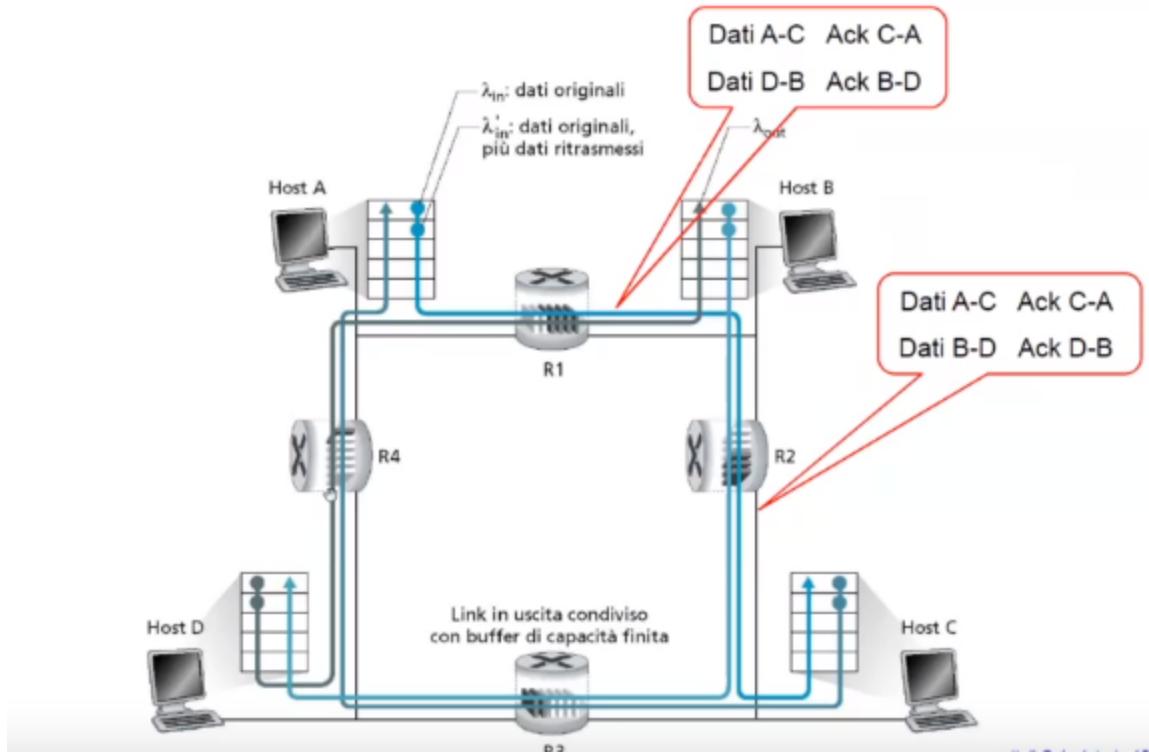
I pacchetti duplicati ci sono perché scade il timer (anche se il povero pacchetto era solo in ritardo).

Il pacchetto duplicato va a fare ancora più confusione nella rete e fa ancora più confusione.

In questo scenario paradossalmente se aumentiamo la memoria al buffer peggioriamo questa situazione. Contrario a quanto si potrebbe pensare inizialmente, converrebbe avere buffer piccoli. (Cosa che comporta altri problemi...)

(-- Nel caso in cui lo chiedesse, esiste una politica di gestione delle code chiamata RED: Random Early Detection ([Wikipedia](#)), che butta "quasi a caso, in realtà non proprio a caso" i pacchetti, e migliora leggermente la situazione. --)

Terzo Scenario :4 Host che si creano congestione a vicenda



Ante Scriptum: Tutto questo paragrafo va letto in chiave di quest'immagine, molte frasi sono prese 1:1 dalla registrazione su MS Stream.

Il traffico in andata e in ritorno seguono via diverse molto particolari e strane.

Sostanzialmente le connessioni si disturbano tra di loro, perché attraversano stessi router e buffer.

È importante sottolineare la definizione di connessione Asimmetrica (ADSL => la A sta per Asimmetrica): Di norma in una connessione asimmetrica le velocità di download e upload sono differenti.

La velocità di upload contiene gli ACK delle connessioni in download. Se si dovesse saturare il canale di upload gli ACK non riescono ad arrivare a destinazione: il mittente che usa il canale di download per noi invia duplicati perché gli ACK non stanno arrivando.

Quindi il traffico utile sarà limitato perché il canale di upload viene saturato da qualcun altro. Questo fatto blocca la comunicazione su ENTRAMBI i canali, anche se separati.

Controllo della Congestione ATM (accenno)

Dunque si ha congestione quando i buffer dei router si riempiono e il traffico viene limitato e infine bloccato.

I router non potrebbero dunque avvertire gli host che i propri buffer si stanno riempendo criticamente e dunque di rallentare la velocità di trasmissione?

ATM permetteva di farlo (non è parte di programma), funzionava bene ma TCP non lo può fare perché TCP mette in diretta comunicazione (logica) i livelli di trasporto: non vede il livello di rete.

AIMD

Quando posso pensare che c'è un problema? Quando scade un timer.

Qui subentra la fase AIMD: **Additive Increase, Multiplicative Decrease**.

Le due connessioni di trasporto non hanno idea della velocità a cui possono andare.

Quando inizia a spedire pacchetti, l'ideale è riempire il più possibile il canale senza saturarlo.

Il problema sta nei router che stanno in mezzo e i loro buffer: idealmente vorremmo il **canale tutto pieno** (massimo throughput) e i **buffer in ingresso dei router vuoti** (minima congestione).

Con AIMD mando i primi pacchetti e va tutto bene, arrivano correttamente. Allora provo a mandarne qualcuno in più. Ogni step aumento di una quantità fissa (lineare) il numero di pacchetti mandati finché non succede qualcosa, ad esempio la perdita di un pacchetto.

Appena succede questo qualcosa mi fermo e decremento dividendo per una certa quantità, ad esempio per 2.

AIMD è una delle tante politiche utilizzate nel controllo di congestione TCP.

Ha una crescita lenta ed è molto cauto.

Nella fase di incremento additivo sto molto al di sotto della capacità massima possibile per molto tempo.

Slow Start

Un altro approccio che si usa in concomitanza con AIMD

All'inizio mando un pacchetto, arriva l'ACK. Ne mando due. Mi arrivano ambo gli ACK.

Ne mando quattro, e così via...

Questo è un approccio esponenziale e molto più "brutale". Quindi si usano insieme.

(prima aveva detto che Slow Start era un'implementazione di AIMD, sbagliando platealmente)

Accenno di controllo della Congestione in TCP

Inizio in Slow Start: cioè vado in incremento esponenziale: 1,2,4,8,16... fino a raggiungere una certa soglia (che verrà spiegato dopo, don't worry), dopodichè vado in approccio AIMD lineare. Al primo timeout torno di nuovo a 1 e ricomincio, ma ora ho modo di calcolare la soglia. L'idea è che la soglia la pongo a metà di dove c'è stato il timeout: Se avviene a '24' (unità generiche per ora), allora la pongo a '12'. Appena la supero proseguo di nuovo in AIMD.

3 ACK duplicati sono un altro problema, viene discusso dopo.

Questo è l'approccio in Tahoe.

Dettagli successivamente.

Lezione 9 - 30/03/2020

Recap lezione precedente e Q&A:

Il **controllo del flusso** è un controllo end-to-end tra mittente e destinatario.

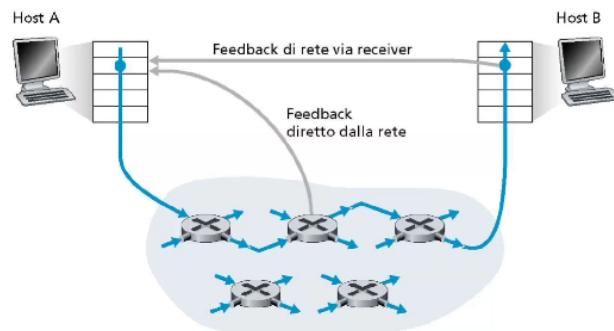
In particolare il destinatario manda in tutti i pacchetti di ritorno quanto spazio ha ancora libero per memorizzare i dati (nel buffer). La rete funziona perfettamente e non genera alcun problema nel trasferimento di pacchetti. L'unico problema può essere il ritardo di comunicazione tra mittente e destinatario, ma non ci possiamo fare niente. Questo problema fa sì che quando il destinatario manda indietro il pacchetto dicendo di fermarsi, tutti i pacchetti che già sono stati immessi nella rete potrebbero non trovare spazio all'arrivo ed essere scartati. Di certo non vengono fermati una volta che hanno attraversato la "soglia" del router mittente e sono nella rete. Se non vengono perduti e arrivano al destinatario, potrebbero venire scartati se la receive window (rwnd) è ancora piena, o essere fortunati e trovare posto nel buffer.

Il **controllo della congestione**: La congestione è dovuta al fatto che ci sono tanti pacchetti ritrasmessi e pochi pacchetti nuovi che arrivano a destinazione. Cioè la rete è piena di pacchetti (inutili) che sono copie, di copie, di copie... - Più pacchetti ci sono fermi nei buffer di transito più la rete diventa congestionata.

(parentesi: I buffer di transito NON sono i buffer di mittente <-> destinatario, ma tutti i buffer dei router che stanno nella rete nel percorso da mittente a destinatario, per questo la congestione è un problema così ostico: non ha a che fare con mittente e

destinatario, che sono collegati logicamente a livello di trasporto, ma ha a che fare con i router a livello di rete [e quindi invisibili a mitt-dest]).

Idealmente vogliamo che il mittente stia sempre a spedire e quindi riempire il canale, ma tenere vuoti i buffer dei router intermedi (cioè che riescano a smaltire il traffico). Tutti questi buffer intermedi sono indipendenti e non coordinati tra loro e potrebbero anche avere traffico di "disturbo" da altre fonti.



I segnali d'allarme della congestione sono il timeout o 3 ACK duplicati.

Il mittente può recepire questi due segnali e capire che la comunicazione non sta andando 'liscia'.

TCP-Mittente inizia a rallentare quando riceve questi segnali, perché ovviamente non può intervenire sulla rete direttamente, sperando che questo rallentamento abbia un effetto positivo (buffer che si svuotano, etc).

Come riprendere a ri-aumentare il traffico?

Non sapendo qual è il throughput max che può viaggiare in termini di banda / pacchetti può fare solo una prova. Se un certo throughput può essere sopportato dalla rete allora prova ad aumentarlo.

Esso viene aumentato in termini di pacchetti spediti (con le modalità descritte precedentemente: Slow Start e poi AIMD).

Tahoe vs Reno

TCP Tahoe implementa:

- Slow Start
- Congestion Avoidance
- Fast Retransmit

TCP Reno implementa:

- Slow Start
- Congestion Avoidance
- Fast Retransmit
- Fast Recovery

Slow Start: Si parte da 1, poi si raddoppia se tutto va bene, 2, 4, 8 con un incremento esponenziale. Ovviamente se non mi fermo in qualche modo saturerei subito la rete e andrei incontro ad un timeout / triplo ACK. Allora rallento una volta superata la Slow Start Threshold (ssthresh) entro in Congestion Avoidance (CA) e aumento linearmente (AIMD).

Ovviamente arriverò ad avere dei problemi anche così, ma molto più lentamente.

Dato che le due versioni di TCP prese in esame implementano il meccanismo di Fast Retransmit posso trattare in modo particolare i 3 ACK duplicati e prenderli come sintomo che è successo qualcosa nella rete.

La differenza grossa tra Tahoe e Reno è il meccanismo di Fast Recovery, che Reno implementa.

Tahoe in entrambi i casi (timeout / 3 ACK dup) si ferma totalmente e riparte in Slow Start, setta ssthresh a metà del valore precedente e riparte.

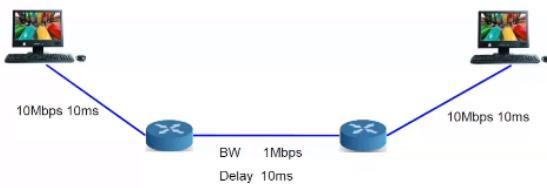
Reno invece fa una considerazione differente, dice "Ok, è successo qualcosa ma non è grave, perché gli ACK comunque mi sono arrivati, non è un timeout vero e proprio."

Si rallenta per dare possibilità alla rete di smaltire un po' di traffico, ma si riprende a partire dalla soglia con AIMD.(??? Controlla kurose perché ha spiegato tutto senza le formule di CWND e MSS)

Scenari Esempio Congestione

Scenario 1: Un Flusso TCP in un canale senza perdite ma con ritardo costante

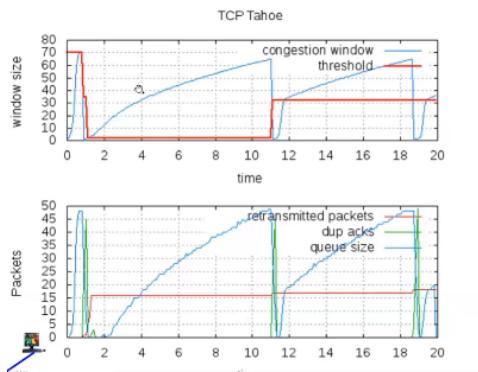
Scenario 1
Un flusso TCP in un canale senza perdite e ritardo costante



Abbiamo 2 macchine e 2 router nel mezzo.

Le macchine sono connesse ai router con 10 Mbps e 10 ms di ritardo (costante)
I router sono connessi tra loro con un collegamento a 1 Mbps e ritardo costante.

TCP Tahoe: Non ci sono perdite e ritardo costante. Situazione anomala: non ci sono perdite il ritardo di propagazione è costante, significa che non ci sono altri elementi nel mezzo e il percorso non viene mai variato. Situazione ideale per TCP.

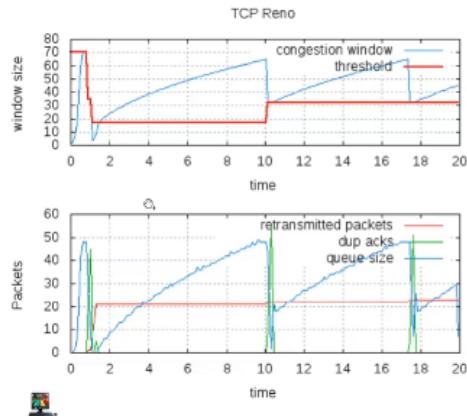


Notiamo che all'inizio c'è un valore di ssthresh a caso, va in timeout ovviamente, rimane a 0 per un po' di tempo e poi si stabilizza. Dopo la stabilizzazione si vede l'incremento esponenziale.

L'incremento non è esattamente lineare: Perché il tempo che impiego a spedire **n** pacchetti non è lo stesso tempo che serve a spedire **n+1** pacchetti. Non è un errore di protocollo o di comunicazione.

Il mittente vede un canale a 10 Mbps, ma nel router in realtà i pacchetti si accumulano perché ha una capacità trasmissiva di 1 Mbps. I pacchetti si accumulano e la situazione collassa. Successivamente ai primi ACK duplicati ricevuti poi si stabilizza essendo una situazione ideale.

TCP Reno: Più o meno le perdite sono allo stesso punto (con lievissime differenze perché sono due simulazioni differenti ovviamente, ma irrilevanti)



Si vede che gli eventi non sono mai di timeout ma di ACK duplicati, infatti il sistema riesce a reagire e recuperare velocemente grazie alla Fast Recovery, ripartendo non da 0 ma dalla ssthresh.

Nel caso di queste simulazioni numero di sequenza più alto => più throughput e quindi connessione che ha trasferito più dati.

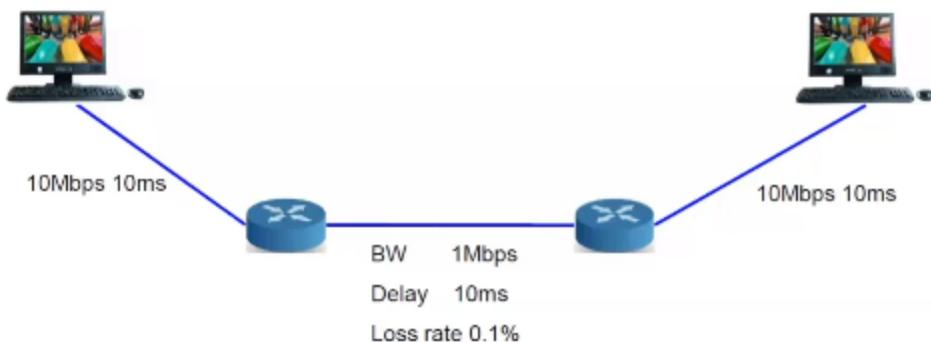
Si vede che Reno riprende più velocemente di Tahoe in questo caso, ma non sempre è così (si può dimostrare con *"carta e penna in realtà"*).

(NDR: Mancano alcune immagini in tutta questa sezione per non appesantire troppo il documento)

Scenario 2: Un Flusso TCP in un canale con perdite e con ritardo costante

Scenario 2

Un flusso TCP in un canale con perdite e ritardo costante



Ai dati precedenti si aggiunge una perdita dello 0.1%, cioè una ogni mille. I pacchetti non si perderanno solo a causa della coda piena ma anche a caso, con le code vuote.

In **Tahoe** la soglia varia radicalmente nel tempo, non si ha più uniformità dopo un certo periodo. L'andamento è molto più confuso.

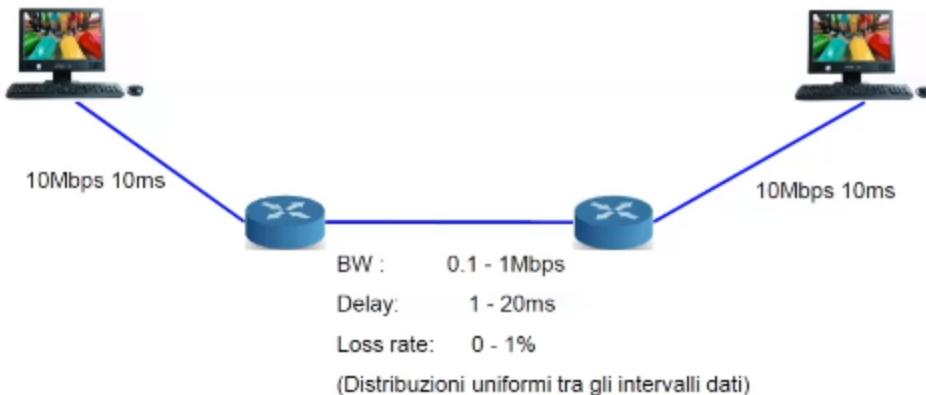
Idem con **Reno**, in alcuni casi addirittura è andato a 0: perché è scattato un timeout (probabilmente più di un pacchetto perso). Andando in timeout riparte da 0 con la SS come fa Tahoe in ambo i casi.

Come prima è importante il grafico dei numeri di sequenza: in questo caso la differenza tra Reno e Tahoe (vince Reno) è molto più accentuata. Quindi Reno sta facendo vedere prestazioni migliori, ovviamente perché il numero di volte che va a 0 è molto inferiore rispetto a Tahoe.

Scenario 3: Un Flusso TCP in un canale con perdite, bandwidth e ritardo variabile.

Scenario 3

Un flusso TCP in un canale con perdite, bw variabile e ritardo variabile



In questo caso la bandwidth non è più costante e neanche il ritardo. Anche le perdite invece di essere sempre di una bassa percentuale, ora possono variare e di molto (da nessun pacchetto perso a 1 pacchetto perso ogni 100).

La variazione di banda è dovuta al fatto che nella realtà i router non sono dedicati solo a due macchine come nella simulazione ma a più macchine. C'è altro traffico (simulato) sul canale

La variazione di ritardo di propagazione è dovuta alla simulazione di altri router intermedi qui invisibili.

Si vede che la threshold ha valori di picco molto più bassi rispetto ai casi precedente. Questo perché se la banda e il ritardo scendono ai loro valori minimi allora la connessione di sicuro andrà in timeout o manderà ACK duplicati.

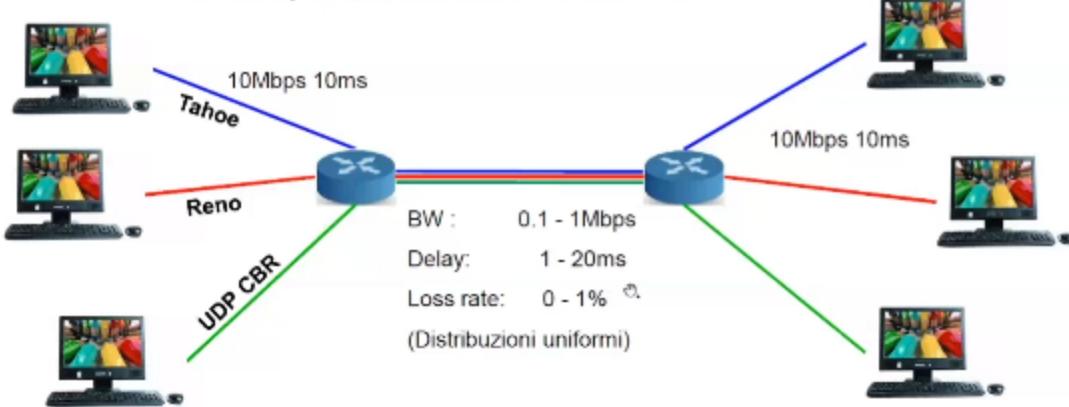
La differenza tra Tahoe e Reno (Reno vince di nuovo) si vede molto prima. Anche se prestazioni piuttosto simili

Scenario 4: Tre flussi concorrenti sullo stesso canale

Scenario 4: tre flussi concorrenti sullo stesso canale

- un flusso UDP CBR
- un flusso TCP Tahoe
- un flusso TCP Reno

Canale con perdite, bw e ritardo variabili



- Un flusso UDP Constant Bit Rate
- Un flusso TCP Tahoe
- Un flusso TCP Reno

Bandwidth, Ritardo e Perdite variabili.

I canali sono uguali a prima come banda.

Le connessioni TCP litigano tra loro.

La connessione UDP CBR prova a mandare traffico a velocità costante.

Non avendo gestione della connessione, se riuscisse ad avere tutti e 10 i Mbit del canale strozzerebbe del tutto le connessioni TCP (Ma il fatto che potrebbe non vuol dire che riesce).

UDP inizia a T=0 e le TCP a T=50.

UDP utilizza 500kbps per volta (la metà del valore massimo del canale tra i router).

Quando iniziano le connessioni TCP il throughput varia di molto. Reno vince quasi sempre su Tahoe, ma a volte Tahoe recupera.

In ogni caso Reno vince sul tempo e la divergenza tra i due grafici diventa sempre più pronunciata.

Il traffico totale fra le tre inoltre non supera mai il valora max il 1 Mbps, ma non è neanche sfruttato al massimo perché quando si raggiunge la saturazione le TCP rallentano per tentare di ovviare la congestione, e durante questo rallentamento il canale non è sfruttato alla sua capacità massima.

UDP invece riesce a fregarsi metà di banda abbastanza consistentemente.

Scenario 4b: Tahoe parte a T=25 e Reno a T=70. UDP spento

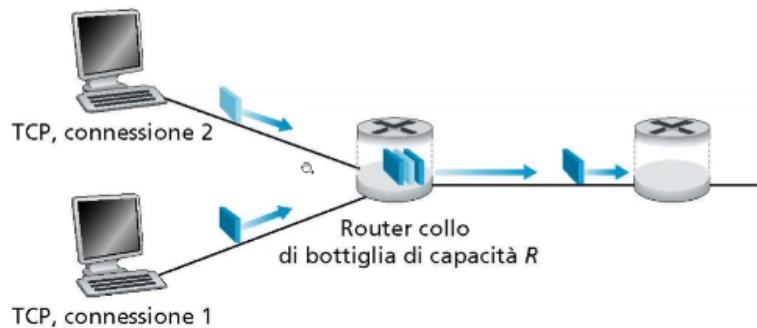
Dopo un po' di tempo si vede che Reno recupera lo svantaggio e supera Tahoe.

Dopo un certo punto però sembra che le connessioni sembrano “collaborare” e condividersi la banda: Tahoe sembra accorgersi della connessione Reno partita dopo e le lascia un po' di banda.

“Sarebbe bello riuscire a dimostrarlo”

Reno riesce ad avere più banda semplicemente perché si comporta meglio in caso di congestione, ma teoricamente se ci sono più connessioni si dividono equamente la banda.

Fairness

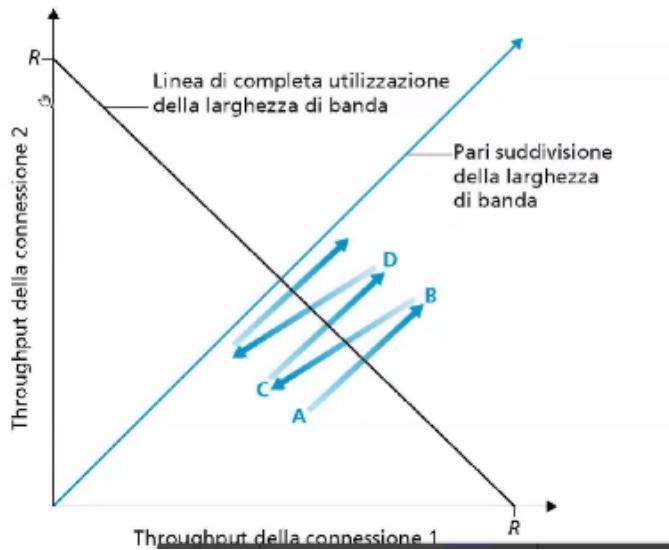


Abbiamo due connessioni TCP, un router centrale che fa da collo di bottiglia e ha una certa capacità R .

L'obiettivo finale è che la banda tra queste due connessioni venga equamente divisa in questo link.

Le due connessioni non sanno che l'altra esiste. Il router non può fare niente: il suo unico compito è di inoltrare i pacchetti fuori dal suo buffer con la sua politica FIFO e scartare i pacchetti che arrivano quando il buffer è pieno.

Dimostriamo che la proprietà della fairness esiste con una "*dimostrazione grafica*":



Andiamo a diagrammare il throughput delle due connessioni.

Abbiamo un punto con delle coordinate (x,y) che rappresenta il throughput delle due connessioni.

Dato che R rappresenta la massima larghezza di banda disponibile in uscita, individuiamo altri due punti: $R^1 = (R, 0)$ e $R^2 = (0, R)$, che indicano rispettivamente che la connessione 1 o la connessione 2 hanno preso tutta la banda disponibile.

Infine, il punto di coordinate $(R/2, R/2)$ indica l'equa suddivisione della banda (al centro del grafico).

Tutta la bisettrice infatti è fatta da punti che hanno le coordinate (x, y) uguali, cioè i throughput sono uguali.

La fairness vuol dire che il throughput sta sulla bisettrice, e l'ideale è stare fermi sul punto a $R/2$.

Il segmento disegnato tra i punti $(0, R)$ e $(R, 0)$ ha tutti i punti di coordinate (x, y) tali che $x+y = R$, perché tracciando una retta che passa per i punti R^1 e R^2 avrà coordinate:

$$y = -x + R \quad \Rightarrow \quad y + x = R$$

Quindi tutti i punti su questo segmento rispetteranno l'equazione $y = -x + R$. -1 è quindi il coefficiente angolare di questa retta (che estende il segmento) e una retta con coefficiente -1 è perpendicolare alla bisettrice (che ha coefficiente 1 e equazione $y = x$ ovviamente, senza termine noto perché passa per l'origine).

Quindi è **importante** dire che il segmento tra i punti R^1 e R^2 è un segmento e non una curva o un'iperbole.

Tutti i punti che sono nel triangolo rettangolo di vertici OR^1R^2 sono punti tali che la somma delle coordinate è minore o uguale a R . Quindi il segmento indica la Linea di completa utilizzazione possibile della banda come dice il grafico. Va da sè dire che all'origine le connessioni sono ferme.

Dato che la somma è al di sotto di R il collegamento non è sfruttato al massimo.

Possiamo avere un throughput maggiore di R quindi? Teoricamente no, ma... invece sì. Come? DEI POVERI STUDENTI SONO STATI BOCCIATI PER NON AVER SAPUTO RISPONDERE A QUESTA DOMANDA.

Consideriamo il fatto che ci sono dei buffer di ingresso che possono accumulare pacchetti, allora può capitare che ci sia un istante in cui il throughput è maggiore di R , semplicemente per il fatto che i pacchetti si stanno accumulando.

Quindi può capitare di arrivare nella zona oltre il segmento $R^1 R^2$. Però arrivare in questa zona del grafico, e quindi di avere un punto in questa zona oltre il max throughput, vuol dire che i pacchetti si stanno accumulando e quindi dato che si accumulano le code si riempiono e prima o poi si arriva alla perdita di qualche pacchetto per timeout o triplo ACK duplicato.

Ora cerchiamo di dimostrare che c'è la fairness.

Supponiamo di trovarci in un punto qualsiasi di questo triangolo, nell'ipotesi che le due connessioni abbiano MSS uguali (che non è neanche una condizione irreale), se sono nella zona di C.A. aumentano tutte e due la banda di 1 MSS ciascuno (LO DICE QUI???) quindi l'aumento del throughput della connessione 1 sarà uguale a quello della connessione 2.

Supponendo di essere sul punto $P(x,y)$, allora il successivo punto P_1 avrà coordinate $P_1(x+1, y+1)$. Per trovare la retta che passa per questi due punti si fa sistema, e il coeff. angolare di questa retta sarà 1. L'unica differenza è che c'è un termine noto ovviamente.

Quindi i punti della connessione si spostano su un segmento parallelo alla bisettrice.

Proprio perché il coeff. angolare è sempre 1.

La distanza (dato che sono paralleli) rimane uguale: non miglioriamo ma neanche peggioriamo, diremmo.

Quindi supponiamo di andare dal punto A al punto B, e una volta che siamo al punto B supponiamo ci sia una perdita, ad esempio 3 ACK duplicati, e, altra ipotesi, le perdite sono contemporanee per le due connessioni. Allora le due connessioni dimezzano i loro valori contemporaneamente, e ci spostiamo al punto C, che vogliamo si avvicini alla bisettrice (altrimenti ci stiamo allontanando dal nostro obiettivo).

Questo punto C si trova nelle coordinate $C(x/2, y/2)$, e come prima dobbiamo fare sistema tra C e B. Risolvendo il sistema:

$$\begin{cases} y = x \\ y/2 = x/2 \end{cases}$$

Quello che viene fuori è che il termine noto dell'equazione della retta è 0. Quindi questa retta ha andamento uguale a $y = mx$. Dato che il termine noto è 0 passa per l'origine.

Quindi il punto C si trova esattamente a metà della retta che parte da B e passa per l'origine (il disegno è sbagliato in questo). Ed è più vicino alla bisettrice rispetto ad A.

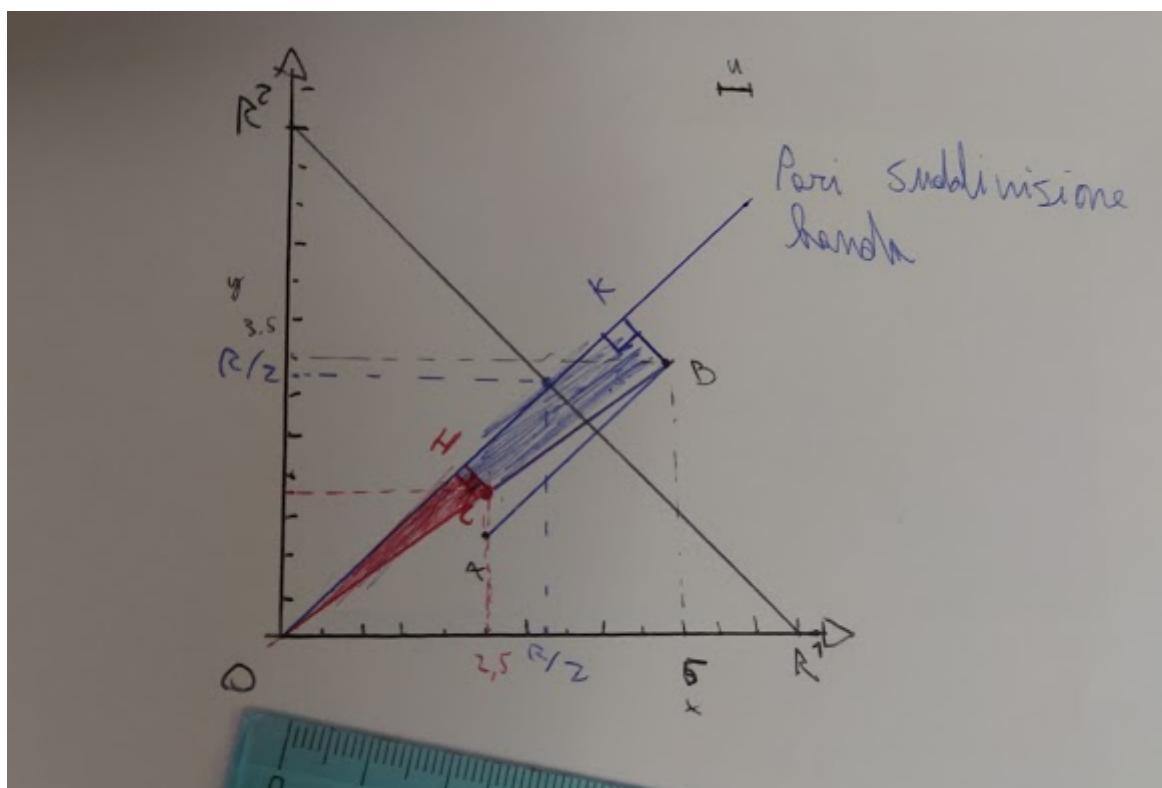
Perché, guardando sia il triangolo **OBK** (dove BK è la distanza tra B e la bisettrice), OK e BK sono i cateti, OB l'ipotenusa, e il triangolo **OCH** (dove CH è la distanza tra il punto

C è la bisettrice, messy drawing below), OH e CH sono i cateti, mentre OC è l'ipotenusa. I due triangoli sono simili, dato che hanno gli angoli uguali fra di loro: l'angolo retto e l'angolo tra la perpendicolare tra C e la bisettrice e la perpendicolare tra B e l'ipotenusa è uguale. Sono due triangoli simili e per questo la distanza tra B e la bisettrice è esattamente il doppio della distanza tra C e la bisettrice.

In altre parole $CH = 2BK$.

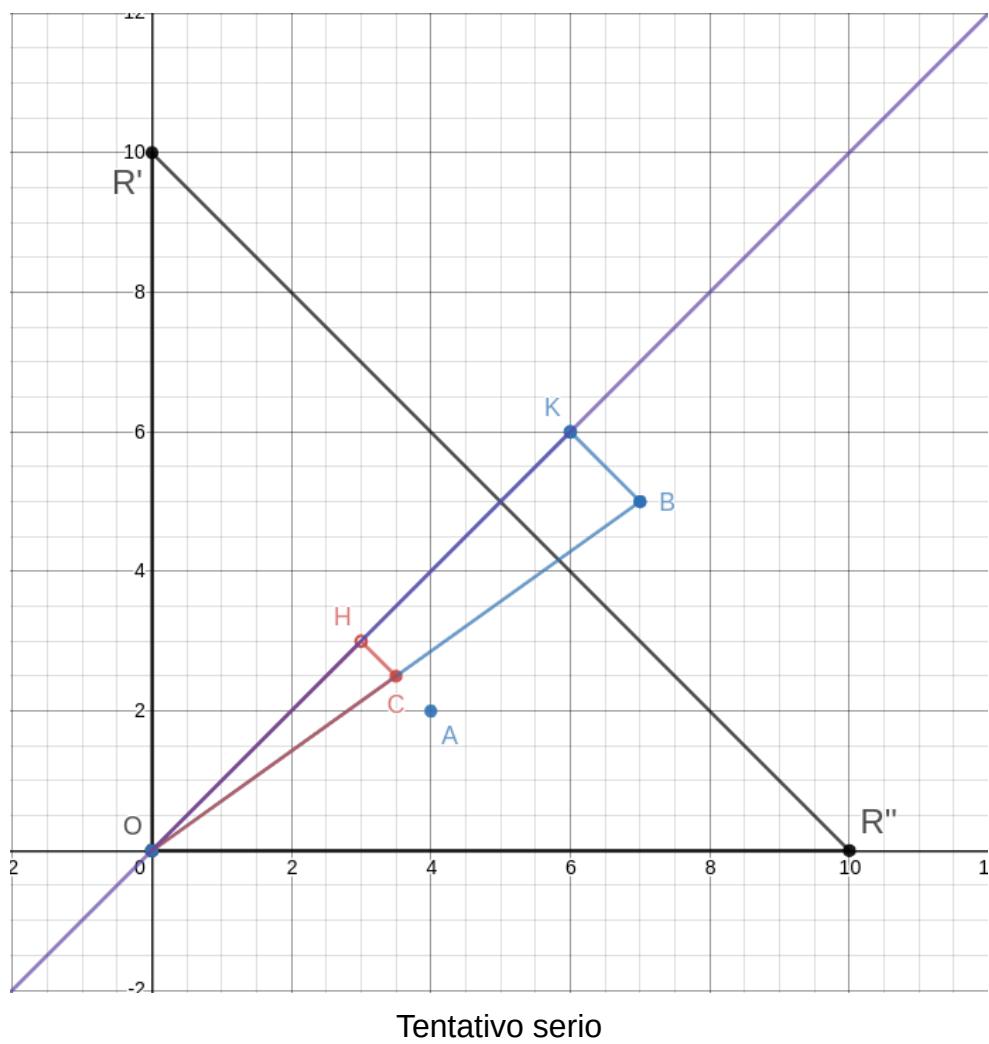
Quindi ci siamo avvicinati all'equità.

Da qui possiamo proseguire a zig zag verso la bisettrice.



Teoricamente se prendessi un righello risulterebbe esattamente come dice Riccobene.

Consiglio davvero di disegnare il tutto seguendo le istruzioni, funziona.



Possiamo superare la bisettrice? Sì, quello che succede però è che ora tendiamo a scendere dall'alto verso il basso: è una zona a “*minimo potenziale, come dicono i fisici*”.

La fairness è importante perché garantisce che anche se una connessione parte dopo un'altra allora la prima connessione dà spazio alla seconda.

Senza la fairness la seconda non troverebbe spazio e non potrebbe comunicare.

Senza la fairness Internet in generale, come idea, non funzionerebbe.

Una connessione che arriva dopo troverà spazio grazie all'esistenza di questa proprietà, per qualsiasi numero di connessioni (per quanto riguarda la dimostrazione, per 3 connessioni si dimostra in uno spazio tridimensionale, per 4 tetradimensionale,

etc...), senza la fairness già una seconda connessione nello stesso canale non potrebbe esistere.

UDP ovviamente non ha neanche la proprietà di fairness e può strozzare le connessioni TCP. Così si potrebbe fare un attacco DOS molto scemo (perché puoi bloccare la connessione)

Usare una politica RED (Random Early Detection, buttare pacchetti “randomicamente”) invece di una politica FIFO può aiutare nel caso di connessioni UDP lecite che stanno strozzando le TCP.

Funziona perché dato che UDP trasmette costantemente allora è più probabile che becco UDP. Se poi per sfortuna becco un pacchetto TCP pazienza, avrò rallentato ulteriormente la connessione TCP (cosa che per quanto riguarda l'affidabilità invece può giovargli).

FINE TRASPORTO

Network Layer (6 lezioni)

Lezione 10 - 27/04/2020

Livello di rete

Il livello di rete riguarda tutti i dispositivi intermedi presenti nella comunicazione end-to-end. Finora abbiamo parlato di **livello applicativo** e **livello di trasporto**, che erano incentrati esclusivamente nella macchina mittente e nella macchina destinazione, il livello di rete è il primo che si occupa invece della comunicazione verso tutti i dispositivi intermedi presenti nella rete.

Abbiamo di due tipi di comunicazione, servizio **datagram**, servizio a **circuito virtuale**, dopo aver affrontato il livello di trasporto è più semplice comprenderne le caratteristiche.

Il **servizio datagram** essenzialmente fa sì che ogni pacchetto segua la strada indicata dal router attraversato, o meglio dire, la strada che il router attraversato ritiene più conveniente, per arrivare a destinazione. Questo significa che i pacchetti spediti al livello di rete possono arrivare in ordine non corretto verso la destinazione.

Nel **sistema a circuito virtuale** si crea un tubo di flusso tra le macchine sorgente e destinazione con tutti i router attraversati, e i pacchetti seguiranno sempre ed esclusivamente questa strada.

Quali sono le conseguenze di queste due scelte?

Intanto, creazione del circuito: nel sistema Datagram non c'è necessità, perché ogni pacchetto può seguire una strada differente. Nel sistema a circuito virtuale invece prima di effettuare la comunicazione, bisogna creare il circuito; quindi bisogna scegliere quali sono i router attraversati, e i router attraversati devono dare il consenso all'attraversamento.

Questo vuol dire anche che l'indirizzo nel sistema datagram deve essere completo, mentre nel circuito virtuale è un qualcosa di molto più semplice. Ogni pacchetto deve portare l'informazione sulla destinazione nel proprio indirizzo per far sì che ogni router attraversato possa decidere autonomamente come far arrivare il pacchetto a destinazione; nel sistema a circuito virtuale invece una volta che è stato creato il circuito non c'è più necessità di avere l'informazione completa, ma i pacchetti semplicemente devono sapere qual è il circuito di riferimento per arrivare a destinazione.

Qual è la grossa differenza tra questi due approcci? In Datagram, dato che ogni router deve decidere più o meno in maniera autonoma, ovviamente non decide solo in maniera autonoma, come far andare avanti il pacchetto, deve fare parecchio lavoro, non può eseguire un'operazione uguale per tutti i pacchetti della serie, perché le condizioni possono cambiare; quindi gli stessi pacchetti che arrivano allo stesso router di ingresso, subiranno una sorta differente a seconda dell'evoluzione temporale della situazione della rete. per cui potranno essere scelte strade differenti.

Nel servizio con circuito virtuale, il router non deve fare nessuna operazione, una volta che è stato creato il circuito, lui deve semplicemente sapere che quel dato pacchetto appartiene a un determinato flusso che già conosce, e quindi inoltrarlo a una strada di uscita.

Quindi, le informazioni che sono presenti nei pacchetti del sistema datagram sono complete (indirizzi dei singoli pacchetti), devono avere tutte le informazioni per arrivare a destinazione; nel sistema a circuito virtuale invece serve solamente un numero (indirizzo di circuito) breve, piccolo, veloce da analizzare, per arrivare a destinazione. Ovviamente legata all'informazione di stato (l'indirizzo espresso poco fa), ci sono le informazioni di instradamento; nel sistema datagram non c'è nessuna informazione a parte l'indirizzo di stato, quindi nessun instradamento; nel sistema a circuito virtuale, ogni circuito deve essere identificato, quindi è presente instradamento; è l'informazione relativa ad ogni circuito che deve essere utilizzato.

Riguardo i guasti nei router, nel caso di datagram, il sistema è stato pensato perché se dovesse guastarsi un router intermedio, ci sono a disposizione altre strade per arrivare a destinazione; una volta identificato il guasto nel percorso, il primo router sa che il percorso del guasto non può essere utilizzato, gli basta sceglierne un altro tra quelli a disposizione, si perderanno pacchetti lungo la strada che si interrompe, mentre tutti quelli che già sono usciti lungo la strada verso la destinazione potranno continuare; ma se il guasto avviene nel circuito virtuale, se dovesse rompersi un router intermedio, allora non è più possibile la comunicazione end-to-end, e il circuito dovrà essere ricreato in qualche modo.

Ci sono delle modalità nel sistema a circuito virtuale (atm), che permettono la ricreazione veloce di questi circuiti.

Ricapitolando per i guasti, non ci sono effetti nel datagram a parte la perdita di alcuni pacchetti, mentre nel sistema a circuito virtuale ci sono parecchi problemi.

Il controllo della congestione, nonostante sia stato trattato al livello di trasporto, si trova al livello di rete; la congestione è caratterizzata da problemi in alcuni dei router attraversati.

Questa informazione non è disponibile direttamente al livello di trasporto ma esso la

deve immaginare in base ad alcune informazioni che riesce a recepire dalla rete, tipicamente perdita di pacchetti o ritardo nei pacchetti.

La congestione quindi si affronta immaginando che ci sia, parlando del sistema **fast retransmit**, dove i due approcci TCP analizzano diversamente l'effetto dei 3 ACK duplicati, uno ritorna un MSS come congestion window, l'altro riparte dalla soglia. Ma sono sempre supposizioni quelle che vengono fatte, non si sa esattamente che cosa è successo, potrebbe essere un pacchetto semplicemente che ha seguito una strada più lunga ed è banalmente in ritardo.

Questo perché non abbiamo informazioni su quello che succede nella rete, ma semplicemente possiamo immaginare che sia successo qualcosa, e l'unica cosa che si può immaginare è che ci sia congestione.

Nel sistema a circuito virtuale invece le cose sono molto più semplici, perché il primo pacchetto che crea il circuito, dice ad ogni router attraversato, quali sono le caratteristiche medie/massime (in base a come è realizzato il sistema) del flusso di dati che verrà utilizzato per la comunicazione. Quanta larghezza di banda, quanti pacchetti al secondo, tante informazioni.

Ovviamente il singolo router a quel punto può fare i conti e dire : "ok, le mie risorse sono impegnate fino a questo livello, ho ancora spazio per gestire il nuovo flusso", quindi dare consenso all'attraversamento, oppure potrebbe dire : "no, già sono vicino alla saturazione, questo nuovo flusso di dati farà sì che vado oltre la mia capacità", quindi negare l'attraversamento e riportare la palla al router precedente, che a questo punto sceglierà una strada alternativa peggiore. Ovviamente in questa fase si fa una ricerca del percorso minimo esattamente come dovrebbe essere fatta nel servizio datagram, però mentre su datagram viene fatta sempre per ogni pacchetto, nel circuito virtuale si fa solo all'inizio.

Una volta che viene stabilita la connessione, tutti i router attraversati hanno le risorse per gestire quel flusso di dati, quindi in teoria non dovrebbe esserci congestione.

Si dice in teoria perché le informazioni che vengono date all'inizio non è detto che siano corrette o complete, se sono valori medi, ovviamente non stiamo dicendo quali sono i valori di picco, possono esserci ancora congestioni, chiaramente questo approccio tende a limitare notevolmente le congestioni già in fase iniziale, a bloccare le connessioni se non c'è la larghezza di banda sufficiente.

In alcuni casi (nel sistema **ATM**), il controllo della congestione è reso ancora più semplice per il fatto che è possibile, da parte dei router attraversati, informare il mittente e il destinatario, sul fatto che si stanno raggiungendo delle condizioni di congestione. Quindi, nel caso datagram il controllo della congestione è complesso semplicemente perché non abbiamo le informazioni dirette dalla rete e dobbiamo augurarci che tutto fili

liscio, la congestione la gestiamo al livello di trasporto ma in realtà è al livello di rete, nel sistema a circuito virtuale la gestiamo al livello di rete, a volte i router informano sull'imminenza della congestione, quindi possono essere prese le opportune misure.

Protocollo IPv4

Cosa c'è al livello di rete? Al livello di rete ci sono tante cose, in particolare il protocollo ip per quanto riguarda gli indirizzi, come è formato un datagram, come vengono gestiti i pacchetti; c'è il protocollo icmp che serve per far dialogare fra di loro i router e per ottenere informazioni dai router di transito, e ci sono i protocolli di instradamento (RIP, OSPF, BGP) che creano le tabelle di inoltro per poter instradare i pacchetti.

Gli indirizzi IP sono formati da 4 byte, uno di seguito all'altro. In una prima suddivisione, avevano deciso di suddividere questi 4 byte in 5 gruppi, chiamate **classi**, e vi erano quindi classe A, B, C, E; di cui in realtà la D e la E non sono stati utilizzati correttamente, A, B, C erano quelli più utilizzati. La suddivisione era abbastanza banale. Gli indirizzi di classe A avevano 1 byte per quanto riguardava la rete e 3 byte per indirizzare gli host; classe B 2 e 2, classe C 3 e 1.

Ovviamente con alcune limitazioni, ovvero, gli indirizzi di classe A cominciano sempre col bit più significativo posto a 0, quindi i numeri vanno da 1 fino a 127.255; la classe B inizia per 10, e utilizza tutti gli indirizzi da 128.0.0 (128 è il primo byte con il primo bit posto a 1 e tutti gli altri posti a 0) fino a 191.255. C e gli altri a seguire. Perché questa suddivisione? Per trovare la strada, si cerca di identificare la rete di destinazione con 1 byte, 2 byte, 3 byte, a seconda della classe; poi una volta arrivati dentro la rete di destinazione, ogni host deve avere un indirizzo differente a seconda della classe potrà avere 16 milioni di host, 65 mila host, oppure 254 host differenti, dentro la stessa rete. L'idea di questa suddivisione iniziale, era che quando si fa routing, si va a guardare solamente l'indirizzo di rete, senza curare l'indirizzo host.

Era una suddivisione comoda perché aveva a che fare solamente con i byte, però la sua comodità era evidente nel momento in cui non si andava a gestire un gran numero di host all'interno di internet, se erano pochi insomma; ma se ci sono alcuni miliardi di host, come al giorno d'oggi, ovviamente questa suddivisione crea squilibri troppo pesanti. I byte di rete non identificano i router; l'indirizzo di rete serve per arrivare al router, e serve principalmente per identificare tutta la LAN collegata al router.

Una volta dentro la LAN non si vanno a guardare più i byte di rete, si guardano solo i byte relativi all'host.

0.0.0.0	This host
00000....xxxxxxxx	Host nella subnet
255.255.255.255	Broadcast
127.0.0.0 – 127.255.255.255	Loopback
10.0.0.0 – 10.255.255.255	IP Privati
169.254.0.0 – 169.254.255.255	Zero conf. net. APIPA
172.16.0.0 – 172.31.255.255	IP Privati
192.168.0.0 – 192.168.255.255	IP Privati

Reti di Calcolatori 7

Alcuni indirizzi sono stati riservati.

This host: è un indirizzo non valido, da non utilizzare mai, anche se poi nel tempo fu concesso di poter utilizzare l'indirizzo 0.0.0.0 per rappresentare lo stesso host, è utilizzato in alcuni protocolli.

Host della stessa subnet: Una macchina ha un indirizzo completo, di classe A, B, C ecc, non è quello il punto. Togliendo una parte di rete, lasciando solamente la parte relativa all'host, la macchina può ancora essere raggiunta, ma solamente da un'altra macchina dentro la stessa subnet. Posso utilizzare quindi, o tutto l'indirizzo completo, per raggiungere un dato host, oppure, se in un qualche modo so che si trova nella mia stessa LAN, posso mettere a 0 tutta la parte relativa alla rete e lasciare solo la parte di indirizzo dedicata all'host, così da poterlo comunque identificare.

Broadcast: se si utilizza l'indirizzo, tutte le macchine riceveranno il pacchetto. L'indirizzo è però bloccato dalla maggior parte dei router, visto che non è possibile che tutti lo utilizzano senza regole.

Loopback: gli indirizzi loopback servono per identificare realmente this host. Quasi sempre l'indirizzo si trova nel formato 127.0.0.1, ma in realtà va benissimo qualunque forma che inizi per 127. Da ricordare che 255 equivale a dire 1.

Se nell'indirizzo della subnet si sostituiscono i valori finali, con l'indirizzo di broadcast, si ottiene 000.255.255.255; ossia un broadcast nella subnet. Stessa situazione si ha nel loopback quando si ha 127.255.255.255, equivale a un broadcast dentro un loopback. Non ha senso però visto che è una macchina sola.

127.0.0.1 identifica la singola macchina, la propria macchina,

IP privati: visto che iniziano con 10, significa che sono di classe A; tutti i router bloccano questo tipo di indirizzo, non possono navigare proprio su internet. A cosa servono? Immaginiamo che per un qualche motivo si vuole creare una LAN interna, con la cablatura sempre presente per andare su internet, ma di fatto non ci si deve andare. Una connessione interna alla propria abitazione, alla propria struttura. Sono indirizzi privati, non sono raggiungibili dal mondo esterno, per cui se una macchina volesse comunicare con un'altra avente un IP privato, di fatto nessun router di quelli da attraversare permetterebbe una cosa del genere. Può servire per fare esperimenti, per fare LAN protette.

Zero Configuration Network: configurazione fatta in automatico dal sistema operativo quando mancano informazioni dal mondo esterno. Nessuno avvisa il sistema sul quale sia il suo indirizzo IP, nessuno ha modo di ricevere questa informazione dal mondo esterno, allora la macchina invece di lasciare tutti 0, decide in qualche modo di prendersi un indirizzo di questo gruppo. Anche questo indirizzo non naviga su internet.

IP Privati: fanno parte del gruppo di classe B ma l'utilizzo è limitato, non è frequente trovarli, se si parla di indirizzi che iniziano con 172.16.0.0; è più frequente trovare indirizzi con 192.168.0.0 . Sono spesso indirizzi forniti dai router domestici.

Subnet Mask (Maschere di Sottorete)

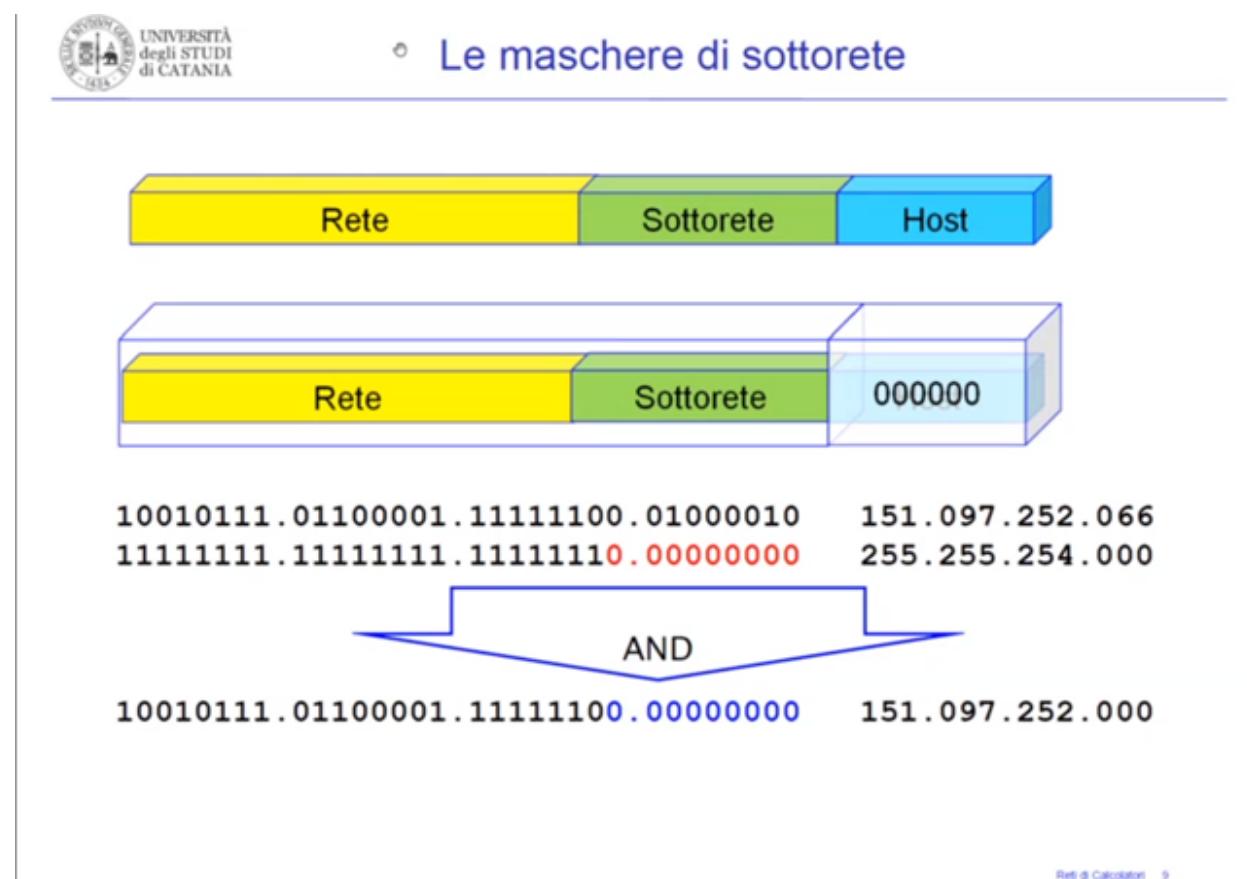
VLSM (Variable Length Subnet Masking)

Sistema con una dimensione variabile di maschera di sottorete

CIDR (Classless Inter-Domain Routing)

Formato di gruppi di indirizzamento (si pronuncia "sider")

a.b.c.d/x -> formato a 4 byte, con valore x compreso tra 0 e 32.



La rete è stata suddivisa in due parti, la parte di rete vera e propria, e la parte di sottorete. La parte di rete e sottorete, insieme rappresentano la singola LAN. Non c'è più l'obbligo di avere due byte esatti, con indirizzo di rete e sottorete non c'è più l'obbligo di avere un byte per la sottorete, possiamo andare a ragionare sui bit singoli. Quindi avere un indirizzo di rete e sottorete fatto non più da 16 bit, ma 17, 18, 19, 20, insomma il numero che vogliamo noi; ovviamente scelto dall'amministratore di rete.

La maschera è un elemento importante, rappresenta un modo per poter, dato un indirizzo, **dividere** la parte di rete e sottorete, e la parte di host.

La maschera è fatta da una serie di bit 1, poi una serie di bit 0.

La cosa importante da sottolineare è che non c'è più il blocco, il vincolo, di avere tutto il byte o niente dello stesso byte, ma il numero di bit 1, sono tutti uno di seguito all'altro seguiti da una serie di zeri, uno di seguito all'altro. All'atto pratico, la maschera si applica con l'operatore AND.

Applicata la maschera esce il valore della Rete+Sottorete, viene "azzerata" nel risultato la parte dell'indirizzo che identifica l'Host.

Quindi con la maschera o prendo solo l'Host o Rete+Sottorete.

Ad esempio, se la maschera è '23', i primi 23 bit indicano la Rete e gli ultimi 9 l'Host (Infatti con 9 bit dedicati agli host ho un totale di $2^9=512$ indirizzi -> 510 host utilizzabili, non possiamo utilizzare indirizzi che terminano con .0 o .255).

Assegnamento indirizzi IP

Esiste un'autorità, **IANA** (Internet Assigned Numbers Authority), che distribuiva gli indirizzi IP, man mano che la struttura internet è cresciuta, non si poteva chiedere più alla stessa organizzazione, per cui sono state realizzate delle **RIR** (Regional Internet Registry) che servivano per gestire gli indirizzi e i nomi di dominio per le varie regioni. A queste RIR si passa anche i **LIR** (Local Internet Registry), che sono quelle locali.



Assegnamento Indirizzi IP

Oggi esistono cinque RIR:

- **APNIC** (Asia Pacific Network Information Center): www.apnic.net.
- **ARIN** (American Registry for Internet Numbers): www.arin.net.
- **LACNIC** (Latin American and Caribbean Internet Addresses Registry): www.lacnic.net.
- **RIPE NCC** (Réseaux IP Européens Network Coordination Centre): www.ripe.net.
- **AFRINIC** (African Regional Internet Registry): www.afrinic.org.

Reti di Calcolatori 11

Il LIR per la rete Italiana è il **GARR-LIR**. GARR è l'organizzazione italiana che gestisce la rete per quanto riguarda le strutture universitarie, GARR= Gruppo di Armonizzazione Reti di Ricerca. Prima struttura internet realizzata in Italia.

Gli indirizzi IP si compra(va)no a blocchi. Ora sono finiti e quindi si cerca di riciclare tutto ciò che è stato male assegnato o non utilizzato

Formato Datagram IPv4

Formato da blocchi di 32 bit ciascuno



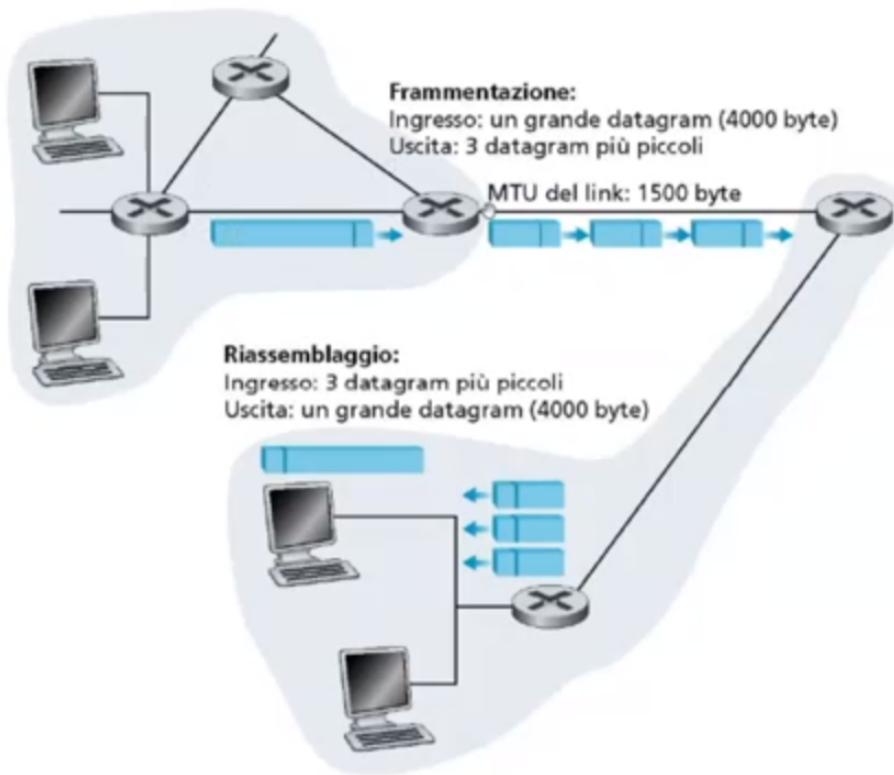
- **Prima Riga:**
 - **Versione:** 4 bit
 - **IHL:** Dimensione del preambolo (Header Length), in gruppi da 4 Byte, cioè ogni “riga” sono 4 Byte
 - **ToS:** Tipo di Servizio: 3 bit di precedenza e 3 flag, non usato
 - **Datagram Length:** Lunghezza totale del pacchetto: 16 bit -> 64 kB max, intestazione+payload.
- **Seconda Riga (Frammentazione):**
 - **ID:** Identifica il Datagram stesso ed è anche necessario per la Frammentazione, se ci sono problemi a mandare un pacchetto troppo grosso i router intermedi possono frammentarlo per migliorare la trasmissione (potrebbero anche riassemblarlo, ma di fatto lo fa solo la macchina di destinazione).
Vedi sotto per spiegazione più dettagliata
 - **Flags:** 3 bit, uno non usato, Don't Fragment e More Fragment.
 - **Offset:** 13 bit, per indicare la posizione (relativa) del pacchetto nella sequenza di frammenti, dettagli sotto. 13 bit -> posso creare 8192 frammenti da almeno 8 Byte

- **Terza Riga:**
 - **TTL:** Time To Live, 8 bit, l'idea è che un pacchetto non deve girare in eterno nella rete, va buttato dopo un tot. Troppo complicato indicare una vera data o quantità di tempo. In realtà è un banale contatore di salti, ogni router decrementa di un'unità il contatore e una volta arrivato a 0 viene scartato. Valore max: 255 salti.
Una delle applicazioni più importanti è rettificare eventuali errori nel routing, si evita che un pacchetto giri all'infinito. (vd. Flooding)
 - **Protocol:** Specifica il protocollo usato a livello di trasporto
 - **Header Checksum:** Checksum del solo preambolo, per vedere se ci sono errori.
- **Quarta+ Riga:**
 - **Source IP:** 32 bit, indirizzo sorgente
 - **Dest IP:** 32 bit, indirizzo del destinatario
 - **Options:** Campo assolutamente facoltativo, motivo per cui esiste il campo IHL. Se questo campo dovesse essere vuoto allora IHL = 0.
 - **Data:** Contiene tutto quello che contiene il pacchetto a livello di trasporto. (Intestazione del protocollo di trasporto + dati applicativi).

La minima dimensione dell'intestazione è 20 Byte (cioè con campo opzioni vuoto).

Quindi 20 Byte + 20 Byte (TCP) = 40 Byte per effettuare una comunicazione in rete.
Con UDP sarebbero 28 Byte.

Frammentazione



La frammentazione si fa perché esiste il valore MTU (Maximum Transfer Unit), che rappresenta la dimensione massima della frame che può viaggiare su un dato collegamento (a livello Data Link).

Ad esempio, dato che l'MTU di Ethernet è 1500 Byte un pacchetto più grande va spezzettato (Ad esempio, 4 kB viene separato in 3 pacchetti da 1.5, 1.5 e 1.0 kB).

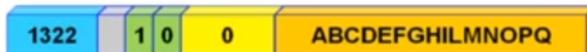
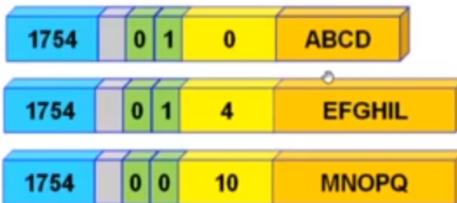
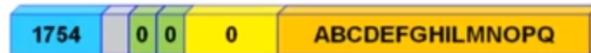


Le Flag sono 3 bit: un bit vuoto, un bit **Don't Fragment** e un bit **More Fragment**.

DF vuol dire non frammentare nei nodi intermedi: O trovi un'altra strada o frammenti a monte.

MF invece serve a frammentare, e serve a dire che il pacchetto in questione non è l'ultimo in una serie di frammenti. Dopo un pacchetto con bit MF=[1] vuol dire che ce n'è sicuramente un altro.

Offset rappresenta la posizione del frammento dentro il messaggio originale.
Un valore 0 significa che è il primo frammento.
l'ID rimane lo stesso per tutto il frammento.



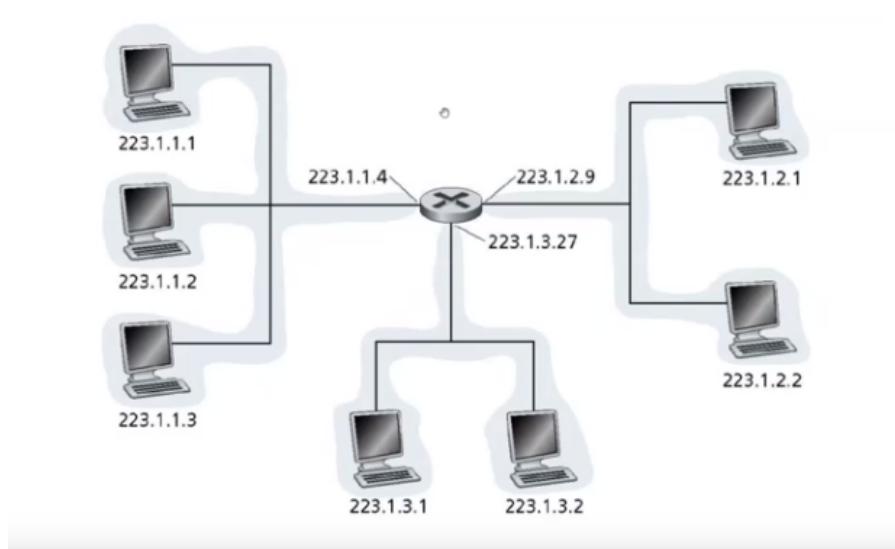
Un esempio di frammentazione

Con questo sistema anche se i frammenti arrivano in ordine sparso ho comunque tutti gli strumenti per ricostruire il datagram originale.

Se un frammento ha sia valore MF e sia valore di Offset siamo certi che è nel mezzo della sequenza, non è né il primo né l'ultimo.

È un'operazione non banale che fa perdere tempo, e soprattutto se si perde anche un solo frammento allora si è perso tutto il pacchetto. Si preferisce evitare la frammentazione il più possibile perché si risparmia molto lavoro sia alle macchine intermedie che al destinatario.

Indirizzi IPv4



Questa è una piccola rete di 3 LAN e un solo Router. All'interno della LAN ci devono essere tutti indirizzi congrui tra di loro. La parte a sinistra ad esempio è sempre "uguale", cambia solo il quarto Byte. (Come le altre).

Ogni macchina deve avere un indirizzo UNICO. Internet non ammette indirizzi duplicati. Gli indirizzi privati sono di fatto duplicati, ma ciò è possibile perché sono bloccati dal router.

Il router in questo caso ha 3 interfacce di rete, e ogni interfaccia deve avere il suo indirizzo di rete, compatibile con la LAN a cui è collegato.

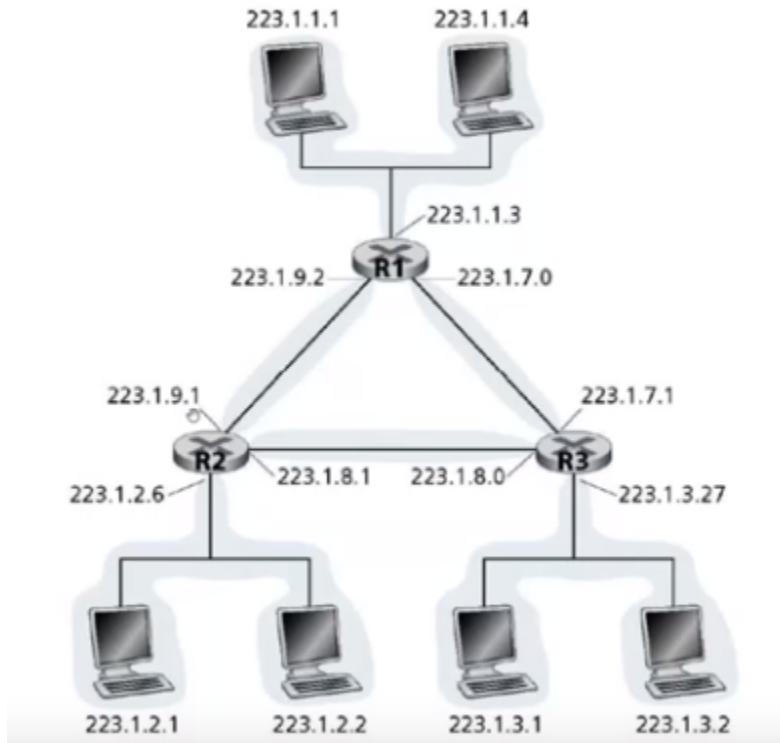
Lo schema sopra può essere visto in modo compatto, dove ogni sottorete è identificato con il seguente schema:

LAN1: 223.1.1.0/24 (Contiene: {223.1.1.1, 223.1.1.2, 223.1.1.3})

LAN2: 223.1.2.0/24 (...)

LAN3: 223.1.3.0/24 (...)

Cioè per identificare tutta la rete si mette a 0 tutta la parte Host. In questo caso possiamo mettere tutto l'ultimo Byte perché la maschera divide perfettamente i primi 3 Byte dall'ultimo. Se fosse /23 avremmo 223.1.0.0/23 ad esempio, perché l'1 farebbe parte dell'Host.

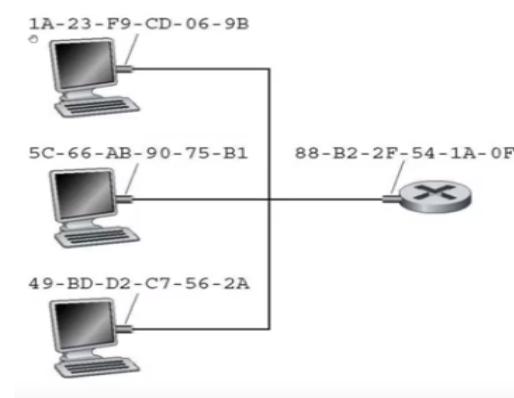


Un esempio un po' più complicato dove oltre alle 3 LAN con degli host collegati ci sono altre 3 LAN "di comunicazione" tra un router e l'altro.

Tutti gli esempi danno informazioni mancanti: Quando rappresentiamo una rete ogni host va identificato sia con l'IP che con la maschera, nel formato $a.b.c.d/x$. La maschera dentro una sottorete ovviamente non può cambiare.

C'è una sorta di errore: 223.1.8.0 e 223.1.7.0 non sono indirizzi validi (così come non lo sarebbero con .255), potrebbero esserlo se il collegamento fosse punto-a-punto di tipo seriale, ma dato che non ci viene data questa informazione presupponiamo che la figura sia sbagliata. Meglio ovviamente usare indirizzi che terminano in .1 e .2.

A livello LAN abbiamo i MAC Address:



Che hanno un formato diverso, e vengono scritti dal costruttore della scheda di rete.

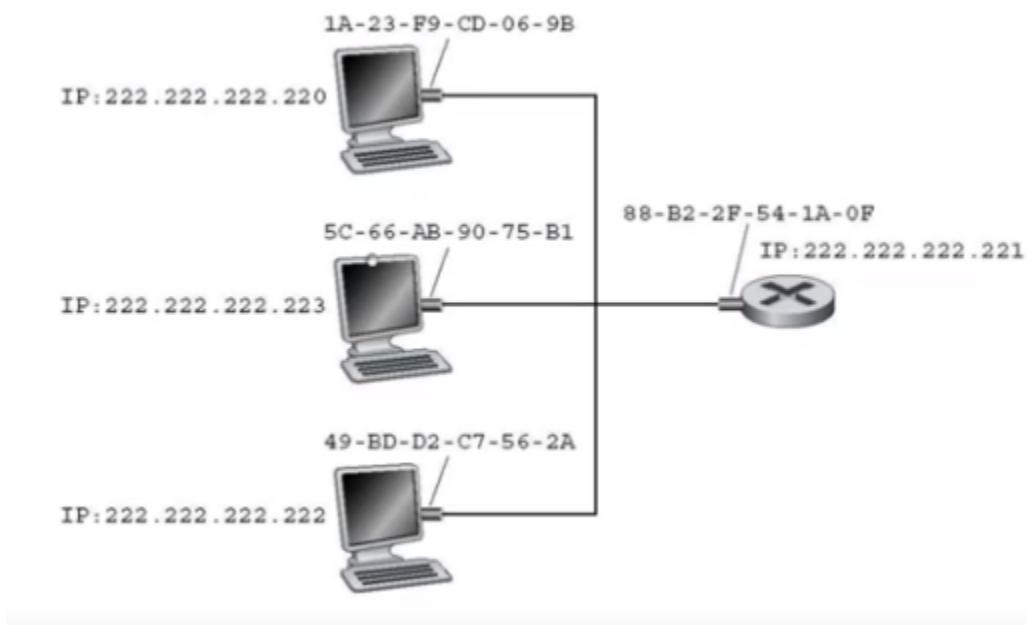
(Fun fact: mettendo i primi 3 Byte su google trovi chi ha fabbricato la scheda di rete).

Devono essere univoci in tutto il mondo. Ogni costruttore si riserva un blocco di indirizzi. Il problema di esaurimento è molto più lontano perché abbiamo 6 Byte, quindi abbiamo 2.81474977e14. "Un bel numeretto". L'unica cosa è che gli indirizzi devono avere il bit più significativo posto a 1.

L'Indirizzo quindi è scritto nel Firmware, quindi non dà nessuna informazione riguardo la LAN di appartenenza, da questo indirizzo non si può risalire alla posizione del computer nella LAN, però è necessario per fare comunicare le Frame fisicamente a livello Data Link.

Quindi ora il problema è: Mi serve un indirizzo che identifica la macchina **DENTRO** la LAN e un indirizzo MAC che serve per fare viaggiare la comunicazione vera e propria che non è legato alla LAN.

Quando una macchina vuole comunicare con un'altra, la macchina di destinazione viene identificata dall'IP, ma deve avere un modo per poter passare da IP a MAC Address e viceversa. Dentro la Frame poi vado a mettere il pacchetto che usa IP, quindi devo poterli usare tutti e due.

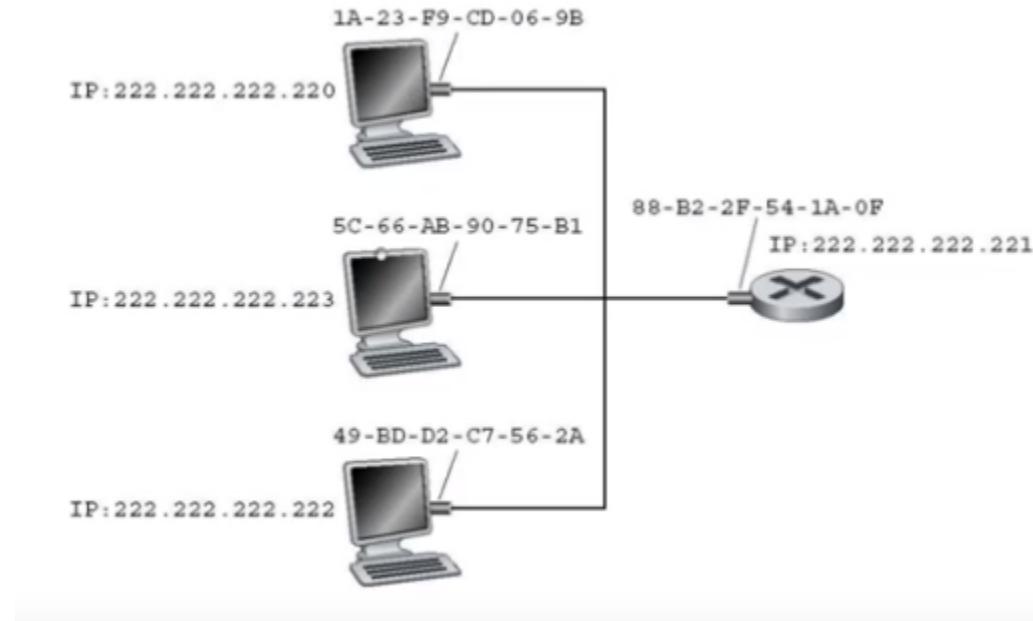


La grossa differenza è che mentre l'IP è legato alla posizione geografica della LAN ed è compatibile con tutti quelli presenti nella LAN, il MAC address invece no, è slegato dalla LAN (è legato solo al costruttore della scheda di rete della macchina).

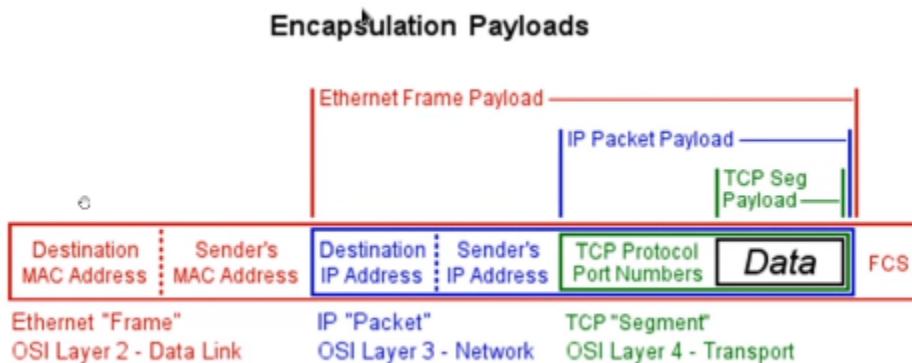
La soluzione a questo problema è ARP e se ne parla nella prossima lezione.

Lezione 11 - 29/04/2020

Indirizzi IPv4 Recap



Questa è una lan dove ci sono macchine che possono comunicare in “broadcast”, chiunque può parlare e chiunque può ricevere quello che viene scritto sul canale. Quello che realmente succede (se non ci sono collisioni), è che c’è una frame Ethernet che viaggia nel canale, al suo interno c’è il pacchetto IP, che a sua volta contiene il paccheto TCP che, infine, contiene i dati applicativi.



Quello che a noi interessa è che nella LAN non interessano in maniera principale gli IP address ma i MAC Address, che è quello che dice verso dove è diretta la frame.

Ovviamente ci deve essere un legame, un link di qualche tipo tra il MAC Address e l'IP Address all'interno di uno stesso pacchetto, altrimenti se sono totalmente slegati tra di loro la comunicazione non avrebbe senso.

[piccola_digressione_start]

Fa vedere una schermata wireshark che dubito abbia senso incollare
Catturando una frame Ethernet troviamo il MAC della destinazione nei primi 6 Byte, a seguire altri 6 Byte del MAC della sorgente, poi c'è una coppia di Byte che indica che tipo di frame Ethernet è (non l'abbiamo ancora studiata), poi ci sono 20 Byte di header IP, 20 Byte di header TCP, valori esadecimali per G E T (47 45 54) ...

Fa vedere la struttura dell'header IP paragonandolo alla figura con i dati in hex

In generale è una spiegazione del flusso di dati che vediamo con wireshark paragonato alla teoria, un modo per far vedere che effettivamente il pacchetto studiato nella L10 esiste ed è utilizzato, e non è un costrutto puramente teorico.

[piccola_digressione_end]

ARP

Abbiamo le macchine che tra di loro parlano tramite indirizzo IP, ma le frame viaggiano usando il MAC address, serve quindi un protocollo per passare dal MAC Address all'IP e viceversa

Questo protocollo è l'**Address Resolution Protocol** (ARP).

È un protocollo distribuito, cioè non vi è alcun server che dà la risposta corretta.

Il mittente conosce l'IP della destinazione (tramite DNS per esempio), ma non il MAC Address.

Prima cosa che fa è mandare in giro una ARP Request dove c'è scritto:

From: 192.168.0.1
(00:00:1A:3E:43:55)
To: 192.168.0.3
(FF:FF:FF:FF:FF:FF)

Mettendo tutte FF al posto del MAC Address sconosciuto essenzialmente sta mandando la richiesta in broadcast sul Link (Esattamente come nell'IP mandare a 255.255.255.255).

Tutte le schede di rete non solo riceveranno la frame (questo avviene sempre), ma in particolare verrà mandata ai livelli superiori. Cioè tutte analizzeranno questa richiesta ARP.

È una richiesta che sta a livello IP.

Tutte analizzano, ma solo una **non** scarta la frame perché le altre notano che l'IP di destinazione indicato nella richiesta non è il proprio.

La risposta viene mandata esclusivamente dalla macchina che possiede quell'IP, e in particolare contiene:

From: 192.168.0.3
(D1:01:CA:13:37:B7)
To: 192.168.0.1
(00:00:1A:3E:43:55)

L'unica informazione nuova è il MAC Address che ci mancava, il destinatario quindi ha già tutte le informazioni per rispondere al mittente.

Tramite questo primo messaggio viene riempita una struttura detta Tabella ARP:

Indirizzo IP	Indirizzo LAN	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Il Time To Live in questo caso è un tempo vero e proprio (e non un contatore di hop come il TTL del pacchetto IP), perché è una tabella gestita all'interno della singola macchina, quindi i tempi possono essere monitorati tranquillamente senza problemi di sincronizzazione tra router o quant'altro. È presente infatti in ogni macchina che ha una struttura IP e degli indirizzi MAC a livello sottostante.

Serve per scartare informazioni troppo vecchie perché potrebbero non essere più valide.

La tabella può essere considerata una sorta di cache, infatti ha anche una dimensione limitata. Oltre al TTL una voce potrebbe essere cancellata perché la cache è piena ed è quella che è stata aggiunta per prima (FIFO).

I PROBLEMI DI ARP

DEI POVERI STUDENTI SONO STATI BOCCIATI PER NON AVER SAPUTO RISPONDERE A QUESTA DOMANDA.

- ARP non prevede autenticazione.

Il protocollo è del tipo: Una macchina fa richiesta e **qualcuno** risponde, il problema è proprio questo, chi è il qualcuno?

Se tutto funziona perfettamente allora dovrebbe essere la macchina corretta, ma nessuno assicura che questa operazione venga fatta realmente dalla macchina corretta.

Funziona sulla buona volontà e correttezza delle singole macchine partecipanti, non è stato pensato con nessun livello di sicurezza. La macchina che risponde dice "Io sono io", quindi chiunque può rispondere, e non c'è nessuna possibilità di verificare che quella risposta sia corretta.

- ARP è un protocollo senza stato.

Una ARP Reply può essere mandata anche senza una precedente ARP Request.

La macchina di destinazione potrebbe mandare la Reply senza nessuna richiesta, che è ancora peggio di ricevere una risposta falsa.

Questo perché questa Reply "autonoma", ovviamente usata a scopi fraudolenti, dice di essere una determinata macchina, e tutte le altre macchine accettano l'informazione come corretta senza nessun controllo.

- Un Host che riceve un pacchetto ARP deve aggiornare la sua ARP Cache.

Per pacchetto ARP si intende sia una Request che una Reply (anche la Request contiene info utili)

Domanda: "Si tratta di una vulnerabilità Man in the Middle?"

Risposta: Non è una vulnerabilità del tipo "Man in the Middle" perché essendo una LAN broadcast i pacchetti non vengono bloccati nel mezzo, ma arrivano anche al vero destinatario.

Si può fare "*una sorta di man in the middle su una cosa strana che succede che vedremo tra poco*"

Questo attacco ARP è a livello DLL sostanzialmente, ingannando il livello IP. Il Man in the Middle avviene a livello IP e TCP.

Conseguenze:

È possibile dirottare il traffico IP.

Immaginiamo lo scenario: Una macchina fa una richiesta e la macchina reale risponde con una reply veritiera. Qualche istante dopo anche l'altra macchina risponde (e può veramente farlo pochi istanti dopo in quanto tutte le macchine connesse nella LAN vedono il messaggio), con il suo MAC Address.

Cosa succede nel mittente? Ha ricevuto due messaggi, entrambi con la stessa associazione IP<->MAC.

Dipende dall'implementazione, ma quello che più frequentemente succede è che il secondo messaggio va a sovrascrivere il primo (semplicemente perché è più nuovo). A questo punto la macchina ingannatrice può fare di tutto: Potrebbe ricevere i messaggi e tenerli per sè, oppure potrebbe anche riceverli e inoltrarli comunque al destinatario reale, realizzando così abbastanza facilmente un attacco Man in the Middle.

Ovviamente ci deve essere anche la risposta, quindi l'ingannatrice farà un secondo attacco, nei confronti del destinatario spacciandosi per il mittente.

Questo attacco funziona solo se queste due macchine non vanno a guardare in profondità i messaggi che arrivano: infatti basterebbe non scartare i pacchetti a livello Data Link per scoprire che c'è un attacco in corso.

Comunque è un attacco troppo banale anche da realizzare, e quindi non si può affidare la sicurezza della comunicazione solo al fatto di essere sicuri che il MAC Address sia corretto.

Cioè, in alcuni router domestici (ADSL) ci sono dei parametri di configurazione (sul wifi specialmente) che permettono di bloccare collegamenti WiFi in base al MAC Address. Basare la sicurezza, l'autenticazione (termine "più corretto") solo sul MAC Address è una cosa assolutamente sbagliata.

Anche perché a dire il vero il MAC Address può essere cambiato su riga di comando in qualsiasi OS, in quanto è solo un campo di memoria.

Questo schema è stato pensato senza livello di sicurezza ed è abbastanza strano mettercelo, quindi non possiamo fidarci della risposta che ci arriva.

Normalmente non c'è nessun problema comunque, perché, guardando con visione d'insieme, la comunicazione sta avvenendo molto probabilmente perché due applicativi stanno comunicando: ma se io faccio una richiesta a questo applicativo mi dovrà pur rispondere in qualche modo. Se la risposta è errata chiudo la connessione.

Se serve un'autenticazione magari con questo attacco si riesce a passare la parte di livello più basso, ma poi baso l'autenticazione su qualche altro sistema, (una chiave condivisa, un certificato, etc...) a livello applicativo.

Il protocollo è stato lasciato senza sicurezza perché anche se si riesce ad ingannare il mittente inizialmente, poi l'attaccante deve continuare a riuscire ad ingannare correttamente tutti gli altri sistemi.

Infatti, le possibilità di fare danno con questa vulnerabilità sono estremamente basse.

"Però si può fare qualcos'altro"



Rifacendoci a questa figura: Abbiamo un Router che vede almeno fino al livello di Rete (in quanto ha un indirizzo IP), se le tre macchine connesse ad esso vogliono comunicare col mondo esterno devono passare per questo router.

Ovviamente il router potrà analizzare il traffico di queste macchine, via software. Supponendo che la macchina in mezzo sia un'attaccante, non potrebbe solo sniffare tutti i pacchetti che passano per questa rete (essendo connessa in LAN, nessuno se ne accorgerebbe tra l'altro), ma potrebbe anche fare qualcosa di più sofisticato.

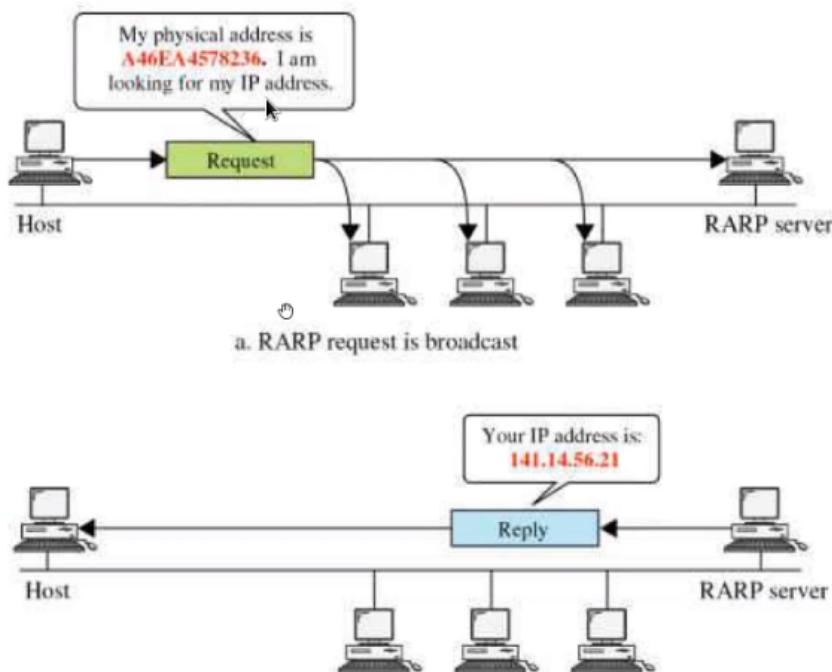
Se l'attaccante riesce a fingersi per il **router di uscita** allora questa macchina potrebbe modificare qualcosa, farli ritardare, far cadere la connessione, filtrarli, o fare davvero qualsiasi cosa.

Il mittente dei pacchetti non se ne accorgerà mai. Questo è un vero e proprio man in the middle.

Per essere definito un **attacco** ci deve essere una qualche modifica o manipolazione dei pacchetti, altrimenti è solo uno **sniffing**.

Un'altra differenza che possiamo fare è che facendo lo sniffing banale l'attaccante potrebbe essere saturato da tutto il traffico di rete (dato che in una LAN vera è più probabile che ci siano centinaia o migliaia di macchine collegate, e convogliare tutto il traffico di centinaia di macchine ad una ovviamente avrà esiti catastrofici), mentre con l'attacco man in the middle allora il suo compito di analisi del traffico diventa molto più semplice, dato che già di default gli arriva il traffico che gli interessa.

RARP



Reverse ARP: a differenza di ARP, non è un protocollo distribuito.

Prevede la presenza di un server ARP che conosce tutti.

Io, macchina, conosco solo il mio MAC Address e non il mio IP. Mando la richiesta a qualcuno che conosce questa associazione. La richiesta ovviamente viene mandata in broadcast perché non so chi è il server.

Il RARP Server risponde dicendo "Il tuo IP, da associare al MAC Address che mi hai fornito, è questo". Non è un protocollo di uso comune ed è molto raro che ci sia un server RARP. Se si usa, si fa spesso in tandem con BOOTP.

Non si usa perché appunto si dovrebbe mettere sù un server RARP nella rete, cosa che non serve.

BOOTP

Protocollo pensato per effettuare il bootstrap da rete in presenza di macchine senza sistema operativo. (Cioè vogliamo caricare, installare un sistema operativo da rete). Nella versione più banale la macchina non ha niente (disco fisso, memoria...), conosce solo il proprio MAC Address.

Deve esistere un server che permette di mandare il kernel dell'OS per effettuare il boot del sistema. Avendo un MAC Address può ricevere un IP Address da un server RARP. Questo IP è staticamente assegnato dall'admin di rete, e quindi potrà configurare la parte minima per realizzare una comunicazione di rete e quindi chiedere al server di mandare tutto il sistema operativo.

L'operazione di trasferimento file viene fatta con TFTP, dove la T sta per Trivial.

Essenzialmente è una versione di FTP senza autenticazione, viene bypassato il controllo dell'utente.

Utilizzato ad esempio per caricare il firmware nei router. È molto pericoloso e deve essere attivato esplicitamente.

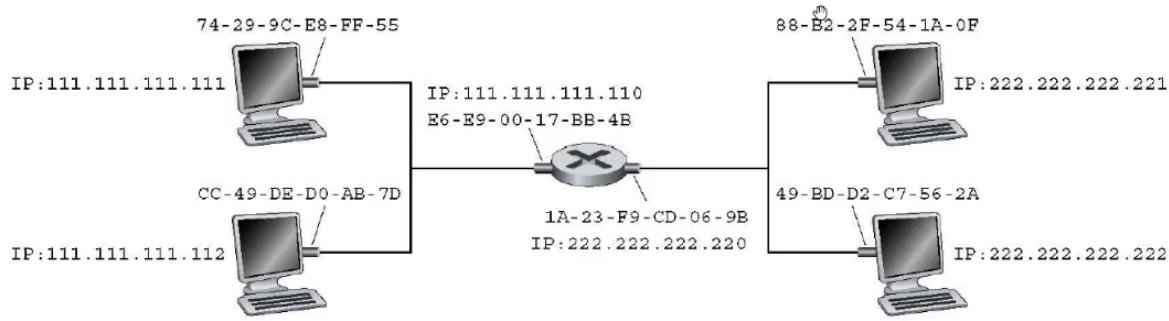
(Esempio UniCT: ExamBox funziona così [ndr: secondo Riccobene, ma ExamBox usa DHCP collegandosi a ExamBox-Server se si guardano i messaggi di sistema, quindi ??]).

BOOTP era utilissimo quando gli hard disk avevano un costo non indifferente ed era utile per creare delle macchine “terminale” senza un hardware dedicato. (Vedi MULTICS).

LAN Interconnesse

**DEI POVERI STUDENTI SONO STATI BOCCIATI PER NON AVER SAPUTO
RISPONDERE A QUESTA DOMANDA.**

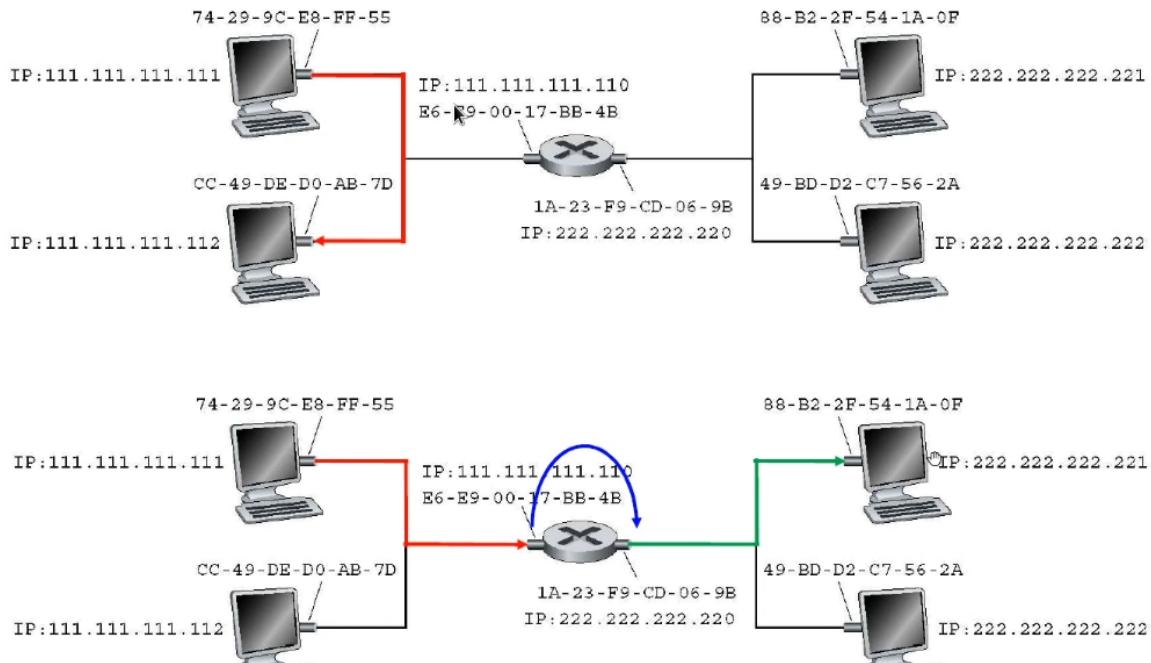
Andiamo al problema più importante: Le LAN interconnesse.



Due LAN separate da un Router (Livello 3 quindi) di mezzo.

Ha infatti sia indirizzi IP che MAC per ogni interfaccia.

Supponiamo che una macchina abbia necessità di parlare con un'altra macchina, che potrebbe trovarsi sia nella sua stessa LAN che in una LAN differente, anche attraversando più router.

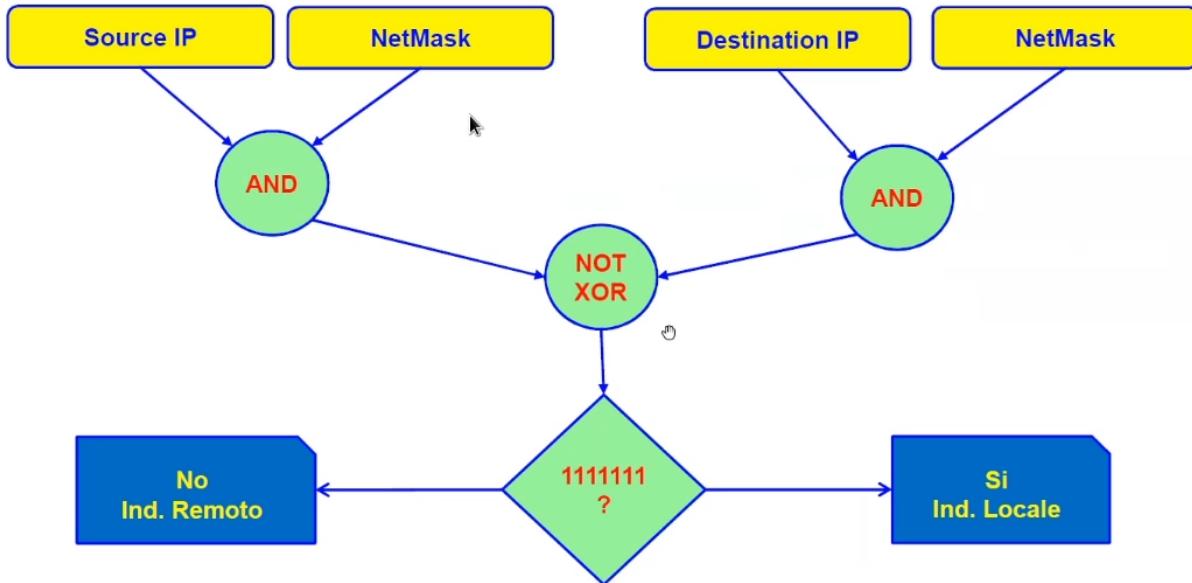


Le due operazioni sono nettamente differenti:

1. **Stessa LAN**: Il mittente deve solo scoprire che il destinatario appartiene alla stessa LAN, quindi può preparare una frame Ethernet (ricordiamo che la comunicazione avviene a livello DLL, non IP, perché il datagramma IP è encapsulato nella frame Ethernet). Deve solo scoprire il MAC Address del destinatario, cosa che può fare con ARP mandando in broadcast la richiesta. Questa macchina potrà rispondere tranquillamente alla richiesta broadcast in quanto si trova sulla stessa LAN
2. **LAN Differenti**: Se il destinatario è fuori dalla sua LAN non può usare ARP per scoprire il MAC address della destinazione, per due motivi:
 - a. Il router non fa passare nulla a livello DLL, perché per quanto riguarda il router ha due livelli DLL differenti, quello della LAN rossa e quello della LAN verde. Anche se il mittente riuscisse a scoprire il MAC address che gli interessa, la frame Ethernet verrebbe comunque spacchettata una volta arrivato al router, e ovviamente scartata.
 - b. Non esiste solo Ethernet: Ci sono reti DLL che non hanno il MAC Address, quindi semplicemente dire che il mittente vuole conoscere il MAC Address della destinazione è inutile, perché potrebbe anche non esistere.

Quindi se la macchina è fuori dalla LAN devo indirizzare il pacchetto al router, il quale spacchetterà la frame Ethernet, leggerà il pacchetto IP e creerà un'ulteriore frame Ethernet da spedire dentro l'altra LAN (verde).

Quindi il problema è innanzitutto scoprire dove si trova la destinazione, se su una stessa LAN o una LAN differente.



Il mittente conosce il suo IP e quello della destinazione.

Conosce anche la sua maschera (della propria LAN ovviamente).

Quindi in primis fa AND con il suo IP e la sua Mask. Così abbiamo l'indirizzo di tutta la sottorete di appartenenza, e tutto il resto viene mascherato con 0. Se faccio questa operazione con due indirizzi differenti della stessa sottorete ottengo lo stesso risultato. Poi faccio la stessa operazione con l'IP della destinazione e con la sottorete del mittente, non solo perché il mittente conosce solo la propria mask, ma anche perché se l'IP di destinazione appartiene alla mia sottorete, allora il risultato sarà identico, e se così non dovesse essere allora ottengo sicuramente qualcosa di diverso (almeno di 1 bit).

Facendo lo XNOR (!XOR, fa 1 solo se i due valori sono identici, quindi 0 XNOR 0 = 1, 1 XNOR 1 = 1, fa 0 altrimenti, ndr) tra i due valori, se ottengo tutti 1 allora la destinazione si trova nella mia LAN, altrimenti si trova al di fuori.

Quindi:

Source IP \wedge Source Netmask = Indirizzo Propria Sottorete

Dest. IP \wedge Source Netmask = Indirizzo sottorete destinatario.

Indirizzo Propria Sottorete XNOR Indirizzo sottorete destinatario => Tutti 1 solo se Source e Dest sono nella stessa sottorete, altrimenti sono in sottoreti diverse.

Se scopro che è nella mia LAN, eseguo una richiesta ARP, ottengo la risposta e genero una frame Ethernet direttamente.

Se scopro che è fuori dalla mia LAN, utilizzo ARP ma per scoprire il MAC Address dell'interfaccia di uscita. In questo caso preparo un pacchetto del genere:

Source MAC Address: Mittente

Destination MAC Address: Interfaccia di uscita

Source IP Address: Mittente

Destination IP Address: Host "remoto" (nell'altra LAN)

Per fare un esempio usando la figura sopra:

Source MAC Address: 74-29-9C-E8-FF-55

Destination MAC Address: E6-E9-00-17-BB-4B

Source IP Address: 111.111.111.111

Destination IP Address: 222.222.222.221

Questo vuol dire che quando configuro una macchina in una rete devo darle come informazioni (almeno):

- Il suo IP
- La sua maschera di sottorete
- IP del router di uscita, o **Gateway**

Avendo l'IP del gateway infatti risulta ovvio che ARP funzioni.

Poi una volta arrivata la frame Ethernet al gateway, allora sarà il router ad eseguire tutti i vari compiti di inoltro (ad esempio, scoprire qual è l'interfaccia corretta verso cui indirizzare il pacchetto, regole di routing da usare...).

Il router infatti poi manderà un pacchetto del genere (ovviamente facendo anche lui ARP prima in quest'altra LAN):

Source MAC Address: 1A-23-F9-CD-06-9B

Destination MAC Address: 88-B2-2F-54-1A-0F

Source IP Address: 111.111.111.111

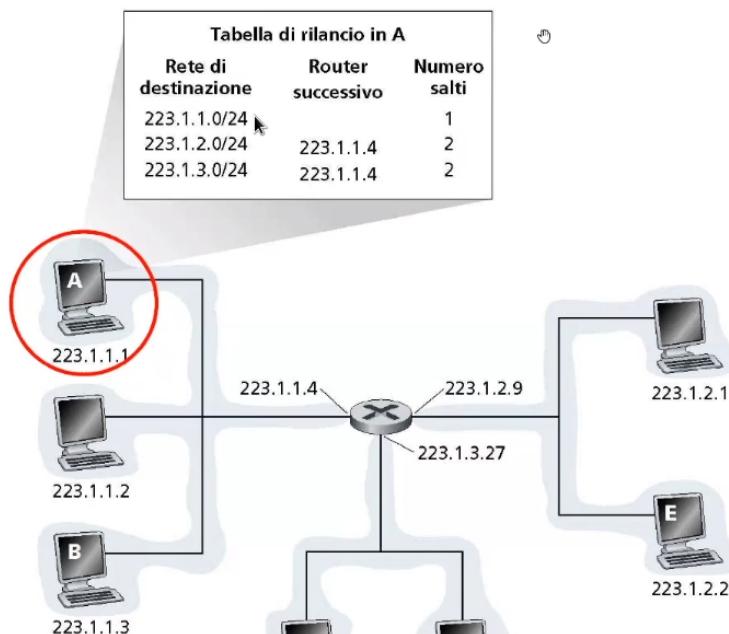
Destination IP Address: 222.222.222.221

[piccola_digressione_start]

Perché al mittente non serve conoscere la maschera della macchina di destinazione? Non gli interessa, né della maschera né di come è combinata la rete di destinazione. È impossibile da sapere. Non è neanche detto che esista una sola maschera.

[piccola_digressione_end]

Tabella di Routing



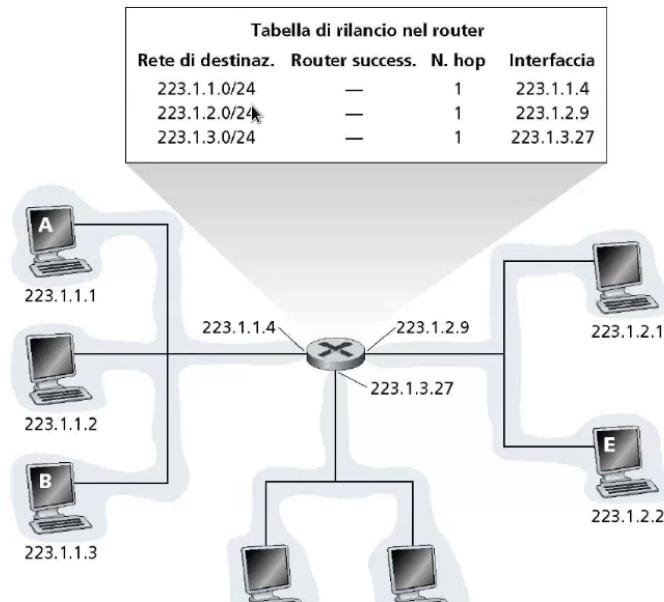
Ogni macchina (non solo il router) ha una tabella di routing come in figura. Contengono (varia a seconda dell'implementazione, manco a dirlo): una rete di destinazione, con una maschera, l'IP del router da contattare per uscire verso la destinazione, e una metrica per capire quanto convenga quell'uscita. Se non c'è router successivo è la sua stessa sottorete (si capisce anche dal numero di salti che è 1). Manca in questa tabella, ma di solito c'è anche l'indirizzo di loopback, "che dice di non uscire neanche nel mondo esterno (*sic*)"

Da notare che conosce la maschera dell'interfaccia che esce sulla LAN "subito dopo", l'host non sa se dopo quell'interfaccia ci saranno altre LAN con maschere differenti.

Questa è una configurazione completa, una semplificazione (che poi di fatto è quella usata nella maggior parte degli host su Internet), è che l'Host conosce due cose:

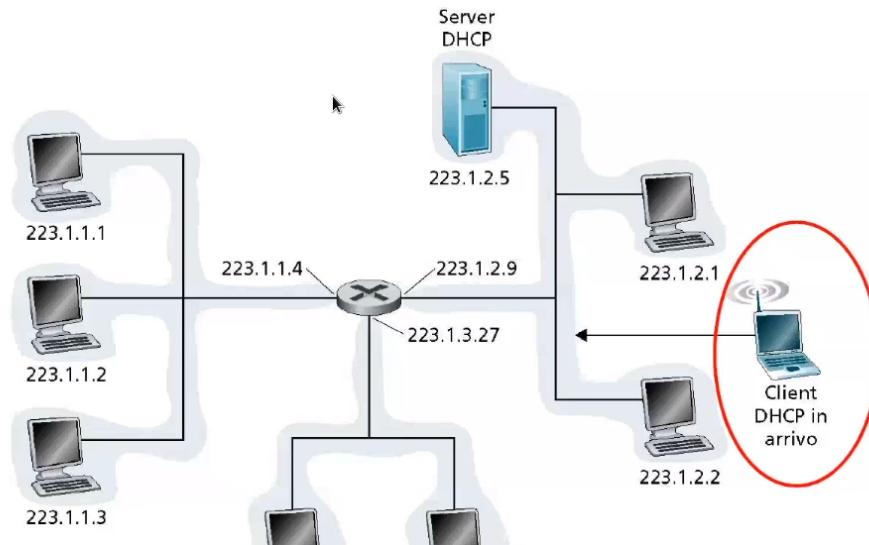
1. La propria sottorete
2. Il resto del mondo

Questo si ha quando c'è un solo gateway di uscita.



Nel router c'è una tabella identica, dove manca l'informazione "Router successivo" in questo caso (perché gli appartengono tutte e tre le interfacce).

DHCP



Dynamic Host Configuration Protocol.

È un protocollo Plug N Play (Non richiede configurazione) per l'assegnazione dinamica di indirizzi IP a host che si collegano a una rete LAN.

Da quello che abbiamo visto finora, l'IP deve essere compatibile con la LAN fisica a cui è collegata la macchina. Con una rete cablata e macchine fisse il problema non si pone. Se le macchine possono spostarsi (portatili, telefoni...) si pone il problema, perché un'eventuale macchina mobile può spostarsi da una LAN all'altra, quindi la macchina va riconfigurata.

La soluzione è di prevedere un server DHCP per configurare l'host al momento della connessione alla LAN.

Una volta che la macchina si aggancia fisicamente alla LAN deve chiedere al server di assegnargli un IP. Dato che si è agganciato fisicamente alla LAN può comunicare con le frame a livello DLL, ma non con IP ovviamente.

Di default tutti i sistemi operativi attivano un DHCP Client che invia una richiesta al momento della connessione.

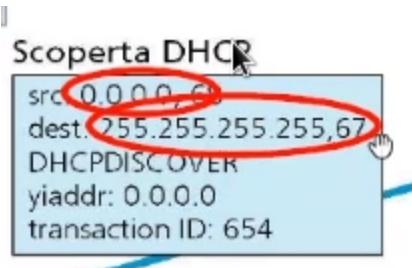
DHCP fornisce anche i server DNS.

Il server DHCP è dunque una macchina critica.

La configurazione con DHCP segue 4 fasi:

1. DHCP Discover (Scoperta)
2. DHCP Offer (Offerta)
3. DHCP Request (Richiesta)
4. DHCP ACK (Accettazione)

DHCP Discover



Mandata dal client.

SRC: 0.0.0.0:68

DEST: 255.255.255.255:67

DHCPDISCOVER

yiaddr: 0.0.0.0

transaction ID: 654

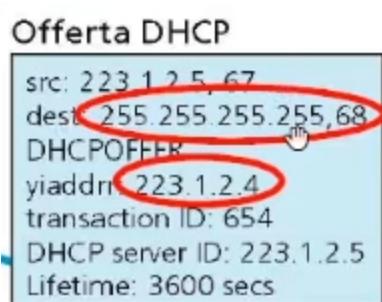
Mette come indirizzo di sorgente 0.0.0.0, cosa normalmente vietata (cosa per cui DHCP è stato criticato), ed utilizza una porta (68).

Come destinazione manda in broadcast, e quindi il router la blocca a livello DLL.

In pratica il nuovo client dice "Io sono nessuno, qualcuno mi dà un IP?", utilizza le porte perché il server deve essere un applicativo ben preciso.

Inoltre utilizza l'ID di transazione per identificare la richiesta, dato che potrebbero esserci più macchine che in contemporanea chiedono un IP.

DHCP Offer



Mandata dal server.

Il server, se esiste, prepara un'offerta DHCP, dove, nel source mette il proprio IP dato che ovviamente lo conosce, nella destinazione mette l'IP broadcast, entrambi con le porte, specifica che è un'Offerta, quale IP vuole offrire, replica l'ID di transazione, dice chi è, e infine per quanto tempo offre questo IP. In questo caso per un'ora.

DHCP Request

Richiesta DHCP

```
src: 0.0.0.0, 68
dest: 255.255.255.255, 67
DHCPREQUEST
yiaddr: 223.1.2.4
transaction ID: 655
DHCP server ID: 223.1.2.5
Lifetime: 3600 secs
```

Mandata dal Client.

La macchina accetta l'offerta, ancora usa tutti 0, perché ancora non è avvenuta la conferma, fa sempre riferimento al Transaction ID, ma aumentato di 1, specifica quale IP sta accettando (ne possono arrivare multipli), la destinazione è ancora broadcast nonostante abbia l'IP del server.

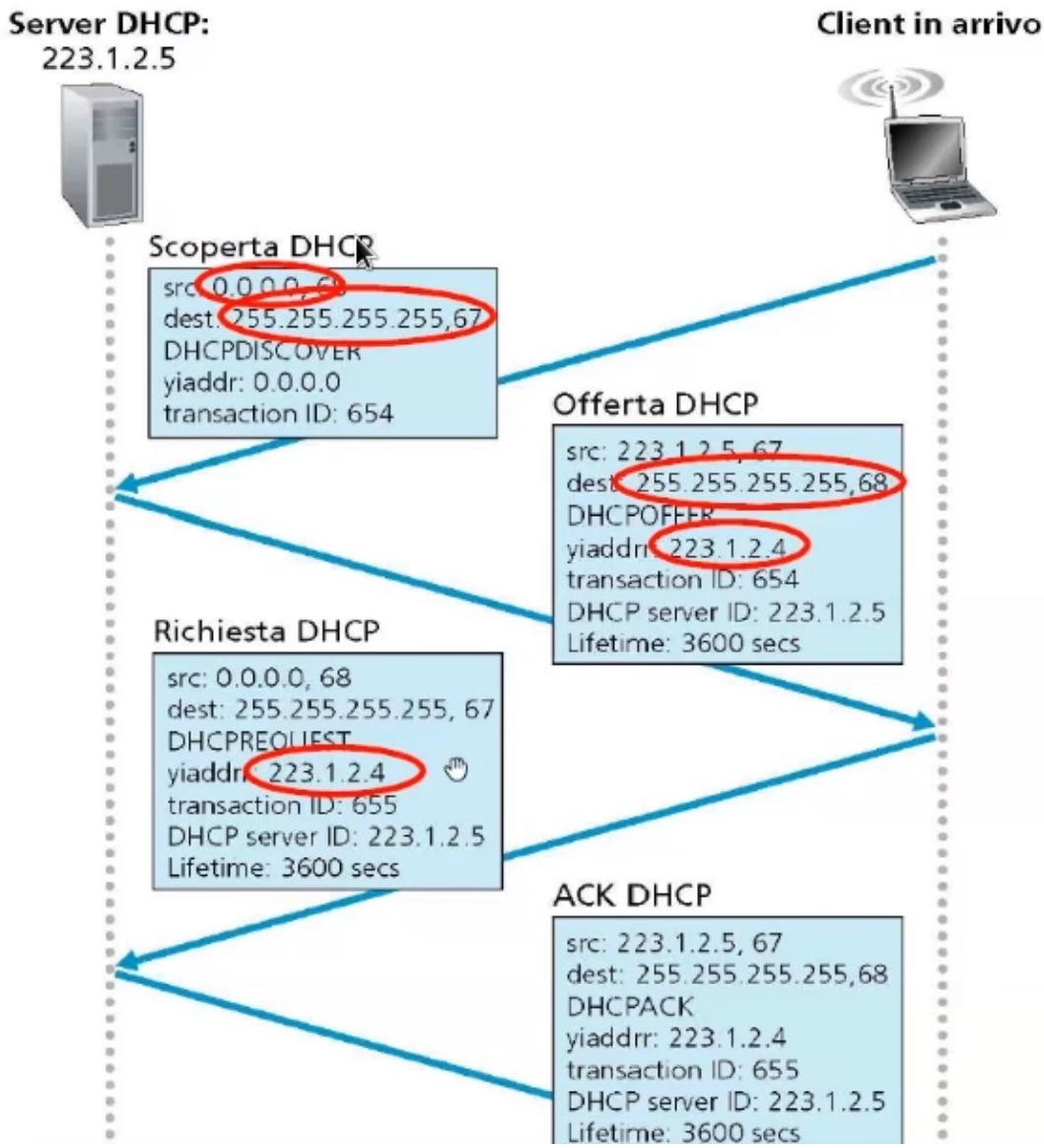
DHCP ACK

ACK DHCP

```
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
DHCPACK
yiaddr: 223.1.2.4
transaction ID: 655
DHCP server ID: 223.1.2.5
Lifetime: 3600 secs
```

Mandata dal Server.

Ultimo messaggio. La macchina è autorizzata ad utilizzare l'IP offerto.



Funzionamento di DHCPv4

Perché 4 operazioni?

La prima è una scoperta, la seconda è un'offerta, ma potrebbe essere rifiutata, perché ne potrebbero arrivare molteplici. Determinare l'offerta migliore dipende dalla macchina. Le ultime due fasi servono per assicurarsi che l'offerta sia stata accettata e quindi il server non darà l'IP ad altre macchine.

Problemi di DHCP

Che succede se passano 3600 secondi?

La regola dice: superati i secondi di Lifetime, o viene rinnovata la richiesta (Deve essere il client a voler rinnovare la richiesta), oppure se non avviene il rinnovo l'IP viene riciclato. Non è previsto un rilascio esplicito.

Se continua a usare l'indirizzo nonostante ciò?

Il server di norma non sà che la macchina sta continuando a usare l'IP, quello che succede dunque è che ci saranno due macchine distinte con lo stesso indirizzo IP.

"E qui succedono i casini più totali"

Quindi abbiamo una macchina che usa l'indirizzo abusivamente e una legittima.

Supponiamo che ci sia un'altra macchina che esegue una ARP Request per comunicare con la legittima. Allora entrambe, sia la legittima che l'abusiva rispondono con una Reply. Nella reply ovviamente ci saranno due MAC Address differenti.

La macchina che ha eseguito la richiesta quale deve prendere in considerazione quindi? Dipende dall'implementazione.

Essendo un problema di un altro protocollo questa situazione non è stata prevista

Empiricamente, dopo un po' di tempo si è osservato che la connessione, nonostante risponda ai ping e collegate via cavo, crolla inesPLICABILmente, in maniera randomica. Questo perché il traffico TCP viene deviato (cosa che non piace a TCP ovviamente).

Inoltre si incorre in un altro problema: Se in una rete è presente un server DHCP abusivo (malevolo o guasto) in più rispetto al DHCP legittimo. In questo caso la situazione è ancora più catastrofica in quanto, a seconda dell'implementazione (ad esempio, implementazione del client DHCP che riceve le offerte), alcune macchine riusciranno a connettersi alla rete e altre no, a causa del server DHCP abusivo. (Un esempio molto pratico è la differenza del DHCP Client tra Windows e Linux-based).

Lezione 12 - 04/05/2020

Recap

Tabella di Routing

Quando in un host viene inserito l'indirizzo di rete per una scheda di rete viene aggiunta una riga all'interno della tabella di routing dove c'è scritto l'indirizzo della sottorete (ad esempio, con maschera 24, 223.1.1.0/24) e la scheda di rete dove è stato inserito dell'indirizzo (in questo caso la scheda di rete ha indirizzo IP 223.1.1.4, viene detta interfaccia).

Questa riga serve per dire: “se al livello IP di quell'Host arriva un pacchetto con destinazione ***questa*** determinata **sottorete**, allora il pacchetto viene inoltrato con ***quella*** data **interfaccia**).

Chiamarla Tabella di Routing (indirizzamento) o di Forwarding (inoltro) è la stessa cosa all'atto pratico. Teoricamente però se il router è un “vero e proprio router” (cioè se il router non è connesso a macchine periferiche collegando più LAN, ma è connesso solamente alla propria LAN e ad altri router verso il mondo esterno), allora è più corretto parlare di tabella di Routing piuttosto che di Forwarding.

L'importante comunque è ricordarsi che questa tabella esiste in ogni host, a cosa serve, come viene creata, modificata e come viene utilizzata.

Le informazioni possono essere inserite nella tabella in vari modi.

Vengono create in automatico quando vengono inseriti gli indirizzi di rete nelle tre interfacce;

Si possono inserire / modificare / cancellare manualmente gli indirizzi;

Oppure queste info possono essere inserite automaticamente attraverso gli algoritmi di routing (vedi Lez.14).

Tipicamente viene aggiunto come ultimo riferimento il valore di default (default gateway), cioè se non sono soddisfatte tutte le regole di routing precedenti, mandala a una data macchina. (Che si spera sappia come indirizzare il pacchetto).

DHCP

Come detto precedentemente è una macchina critica, perché qualsiasi macchina potrebbe fingersi un server DHCP e fornire indirizzi fasulli. Un esempio sono le reti WiFi free dove è opportuno non connettersi o perlomeno utilizzare delle precauzioni.

Le macchine accettano (a meno di prendere precauzioni) ciecamente quello che gli arriva. Il vantaggio grosso è che non devo configurare la macchina per connettermi.

NAT

Tipicamente quello che fanno gli ISP è di fornire un solo indirizzo IP al modem / router quando si collega al suo *omologo(?) presente negli ISP(??)*.

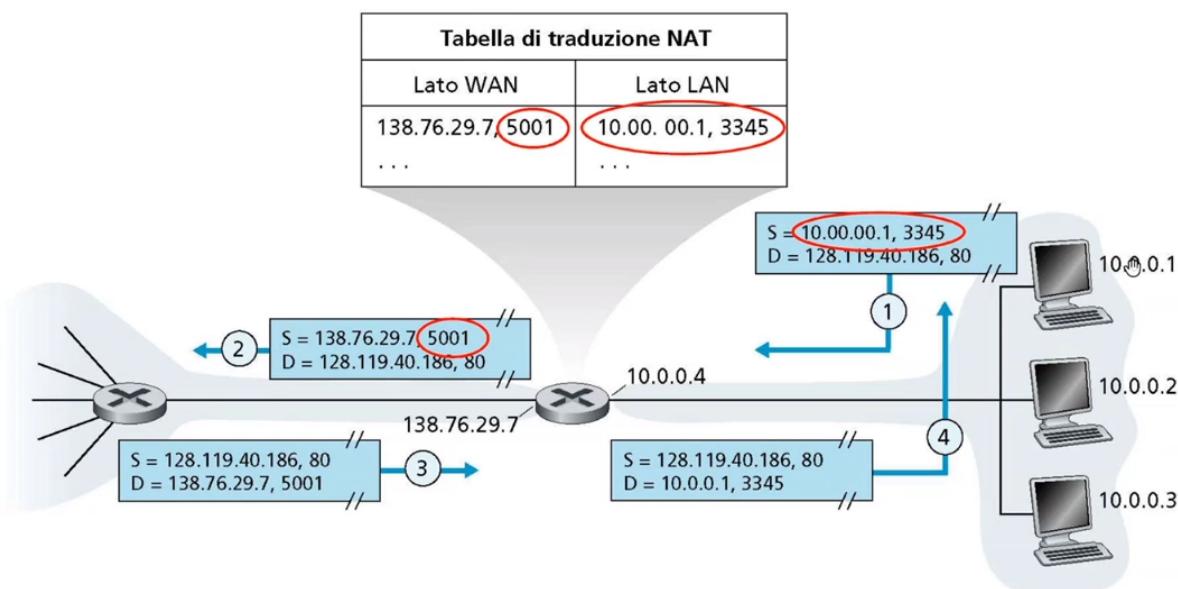
Questo perché gli IP scarseggiano, non è opportuno dare una serie di indirizzi a un abbonamento di tipo Consumer.

(ndr. Al momento della scrittura TIM teoricamente non fornisce neanche un indirizzo IP, in quanto si deve pagare un abbonamento di tipo business per avere un IP Statico. Ti prendi quello che ti affibbia il DHCP dell'ISP).

All'interno della LAN conviene avere indirizzi privati (es: 10.0.0.1, 192.168.1.1, etc..) perché non c'è nessun obbligo di avere indirizzi configurati correttamente secondo politiche a livello globale per una rete domestica, di piccole dimensioni (altrimenti dette reti SOHO: small office/home office).

Se il provider fornisce indirizzi pubblici dovrebbero essere messi in maniera corretta su ogni macchina, dato che sono pubblici. Se configurati incorrettamente potrebbero far danno a livello globale. Con indirizzi privati eventuali errori non si ripercuotono all'esterno della LAN perché il router di default li blocca.

Ovviamente però gli indirizzi privati sono inutili se non possono uscire all'esterno in qualche modo. Questa operazione di "uscita" viene fatta con il protocollo NAT: **Network Address Translation**.



Immaginiamo che ci sia la macchina 10.0.0.1, con porta locale (random) 3345, che vuole parlare con l'indirizzo pubblico, legale 128.119.40.186 su porta 80, utilizzando UDP o TCP.

Il pacchetto arriva al router, in particolare al processo che gestisce il NAT, e viene creata una entry sulla tabella di traduzione NAT all'interno del router dove c'è scritto:

Lato LAN: IP e Porta della macchina sorgente

Lato WAN: l'indirizzo pubblico con cui la macchina sorgente vuole comunicare e una **nuova** porta (*quasi random*) presa tra quelle libere nel router. Quindi la porta viene cambiata.

Nel pacchetto che viene mandato fuori al posto di indirizzo mittente 10.0.0.1 e porta mittente 3345 verranno messe queste altre informazioni: 138.76.29.7 e porta 5001. Il pacchetto alla fase (1) viene di fatto trasformato in quello che vediamo alla fase (2). La sorgente viene mutata.

La porta viene cambiata perché, banalmente, è come se fosse il router a spedire il pacchetto e non la macchina nella LAN, quindi va messa una porta libera. La porta che l'host ha scelto viene appunto scelta **dall'host e non dal router**, cioè non è detto che la porta che l'host ha scelto **sia libera sul router**. Quindi il router per prevenire errori la cambia a prescindere.

Dopodichè il pacchetto è libero di viaggiare su Internet perché sia source che destination sono validi.

Torna indietro con sorgente e destinazione invertiti. A questo punto il router ri-analizza la tabella, però stavolta la esamina al contrario per tradurre la destinazione WAN a destinazione LAN.

Con questo sistema è possibile far sì che tutte le macchine presenti nella LAN privata possano comunicare col mondo esterno.

Ovviamente ci sono tante limitazioni:

Problemi di NAT

1. Tutte le macchine private escono con lo stesso indirizzo IP.
In pratica per quanto concerne il mondo esterno (il destinatario, etc) è come se tutte le operazioni venissero eseguite dalla stessa macchina.
2. L'utilizzo della porta.

Un router può scegliere tra circa 65.000 porte, però, supponendo di avere 10 macchine in una LAN, allora le possibili combinazioni di porte sono 650.000, quindi (nonostante nella pratica è difficile che succeda) può capitare che una LAN abbastanza grande arrivi a saturare tutte le connessioni che possono essere effettuate dal singolo router, saturandolo.

Per limitare ciò si possono fornire più indirizzi IP (cioè più schede) al router. Di fatto il numero di porte disponibili così diventa $65.000 \times$ IP disponibili.

3. Una volta che viene generata una entry con una data porta, quella porta non potrà essere utilizzata per comunicare finché la entry non verrà cancellata dalla tabella NAT. Quindi non è possibile predeterminare la porta utilizzata per il messaggio di invio. (Anche se non è un requisito). Con lo schema NAT non si può fare. (ndr: ma molti lo fanno...?).
4. Se la comunicazione parte dal mondo esterno il NAT non sa che traduzione fare, in quanto non c'è una entry nella tabella, e quindi il pacchetto non può entrare dentro la LAN e viene scartato dal router. Con NAT la comunicazione deve sempre partire dalla LAN.

Soluzioni:

- a. Mettere una riga statica nella tabella a lato WAN indicando una porta di entrata (IP qualsiasi), che gira in automatico il pacchetto a una data macchina nel lato LAN a una data porta. (Di fatto il Port Forwarding / Mapping se si vuole hostare un server).
- b. UPnP (vedi sotto, di fatto la soluzione 'a' ma dinamica).

UPnP

Protocollo che permette alla macchina di mandare un messaggio al server NAT dicendo: "Creami una riga statica in modo da girare una comunicazione proveniente dall'esterno a me per default".

Questo protocollo fa sì che la riga non venga scritta dall'amministratore di rete, ma venga messa al momento, dalla macchina, quando ha bisogno di mettere sù il servizio esposto su Internet.

Ovviamente se un'altra macchina prova a usare UPnP sulla stessa porta verrà restituito un errore. Le entry UPnP nella tabella NAT hanno pure una scadenza, a differenza di quelle statiche settate dall'admin.

IPv6

IPv6 deriva dai limiti di IPv4:

- Esaurimento dello spazio degli indirizzi (4 miliardi: allocati male inizialmente, estrema diffusione di Internet negli ultimi anni, e Internet Of Things).
- Scalabilità del Routing (dovrebbe essere fatto in maniera geografica, per esempio puramente arbitrario: In America mettiamo gli indirizzi che iniziano per 1, in Canada per 11, in Europa per 2, etc..., Con IPv4 questo non succede e IP numericamente vicini potrebbero essere agli antipodi).
- Nuovi Servizi (Servizi che potevano essere implementati hanno dimostrato dei limiti su IPv4).

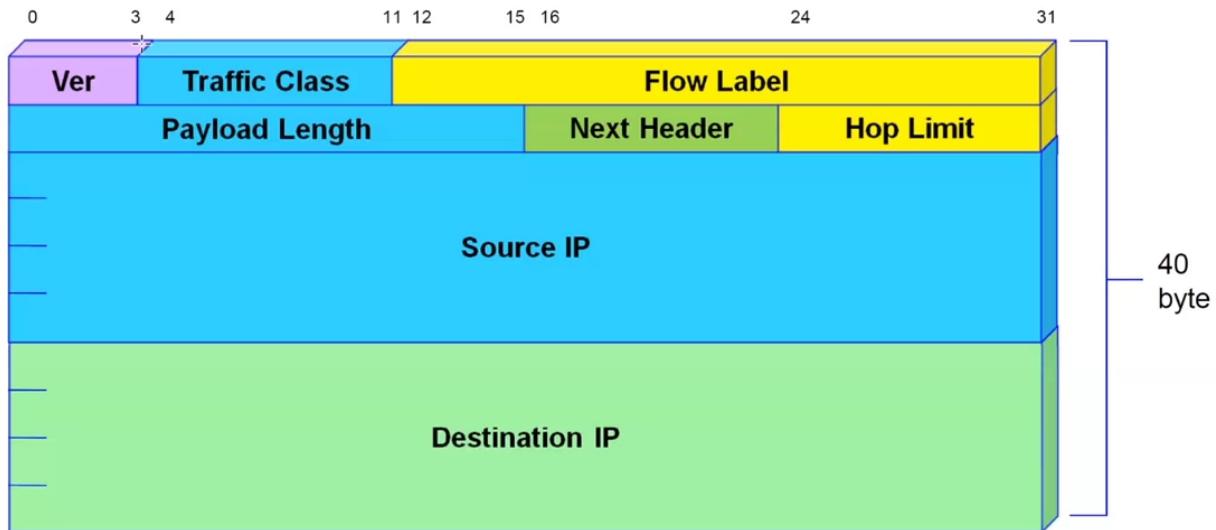
Oggi gli indirizzi IPv4 sono completamente finiti.

Inoltre, le entry nelle tabelle di routing iniziano a diventare “pesanti” (troppo complesse), un routing complesso è un routing lento.

Nuovi servizi di IPv6:

- Sicurezza
- Autoconfigurazione (Plug & Play)
- Gestione della Quality of Service (Prevista in IPv4 ma mai usato / ignorato)
- Indirizzamento Multicast (tanti host all'interno di una stessa rete)
- Indirizzamento host mobili

Formato Header IPv6



- Versione: Primi 4 bit.
- Classe di Traffico: 8 bit
- Etc... (*"li vediamo dopo"*)
- IP mittente / destinatario: da 2^{32} si passa a 2^{128} .

Siccome indirizzi da 2^{128} pesano un casino viene semplificata l'intestazione.

Sono stati **rimossi**:

- ID, Flags, Offset (si sconsiglia la frammentazione in IPv6)
- ToS (inutilizzato), Header length (Ci si sposta a un header a dimensione fissa)
- Header Checksum (considerato ridondante, non serve più)

HLen è stato rimosso perché con l'intestazione fissa il router è sicuro che deve guardare solo i primi 40 Byte senza sorprese particolari dopo.

Le opzioni vengono gestite in un altro modo.

Sono stati **cambiati**:

- Total Length → Payload Length
- Protocol → Next Header (gestito diversamente)
- TTL → Hop Limit (gli viene dato il nome corretto, 1 Byte)

Total Length diventa Payload Length dato che è diventata l'unica variabile nell'intestazione.

Sono stati **aggiunti**:

- Traffic Class
- Flow Label

Campo Flow Label

DEI POVERI STUDENTI SONO STATI BOCCIATI PER NON AVER SAPUTO RISONDERE A QUESTA DOMANDA.

Quando il campo Flow Label è settato (contiene un numero) i router di transito non vanno più a guardare l'indirizzo di destinazione, per quanto riguarda il routing, ma guardano nell'apposita tabella per capire come fare andare avanti il traffico.

È un modo per inserire il sistema di circuito virtuale in IP (che migliora molto il routing rispetto al servizio Datagram puro di IPv4).

Quindi i pacchetti di uno stesso "stream" seguiranno tutti la stessa strada in ordine.

Non ha preso molto piede e spesso viene ignorato.

Indirizzi IPv6

Avendo una quantità smodata di indirizzi, questi sono quelli di nota. Particolarmente interessanti sono quelli che iniziano con 100:

0000 0000	Riservati (includono IPv4)
0000 001	Indirizzi OSI
0000 010	Indirizzi Novell IPX
010	Indirizzi per service providers
100	Indirizzi geografici
1111 1110 10	Indirizzi locali di canale
1111 1110 11	Indirizzi locali di sito
1111 1111	Multicast

Struttura di un indirizzo IPv6:

Diviso in 8 blocchi da 16 bit l'uno, ogni cifra è capace di rappresentare 2^4 (16) valori (quindi rappresentabile in esadecimale, 0-F).

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

8000::123:4567:89AB:CDEF

I doppi due punti indicano tutti zeri fino al primo cambiamento.

::151.97.1.1 -- Indirizzo IPv4 mappato all'interno della gerarchia IPv6.

::1 -- loopback

:: -- unspecified

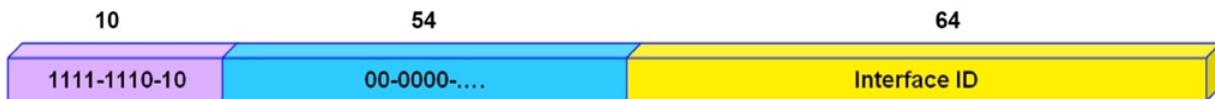
Global Unicast Addresses:



L'indirizzo viene suddiviso sempre in Rete, sottorete e interfaccia, spesso 64 di sottorete e 64 di interfaccia (ma non è un obbligo), la maschera funziona esattamente come in IPv4 (ovviamente con valori più alti).

Indirizzo UniCast significa che indica una singola macchina. I primi 48 bit indicano il "sito", cioè a una data regione è stato assegnato questo ID, detto Global Routing Prefix. I primi 3 bit dicono che tipo di indirizzo è, gli ulteriori 45 qual è il sito a cui si fa riferimento. Poi altri 16 bit di subnet per poter raggiungere velocemente la macchina di destinazione. Infine abbiamo 2^{64} indirizzi possibili.

Indirizzi Locali: FE80::/64



Utilizzabili solo tra nodi dello stesso link.

FE80 occupa 10 bit. (1111-1110-1000-0000-....)

Può essere un'azienda, un ente... che comunque deve specificare che vuole degli indirizzi di sito.

Indirizzi di sito: FEC0::/48



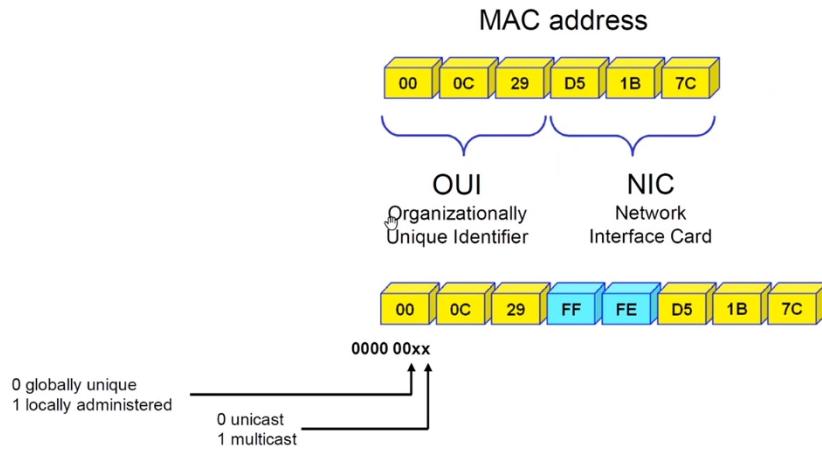
Si possono usare solo tra nodi di uno stesso sito (sconsigliati secondo Riccobene, RFC 3879 li depreca).

Simili a quelli di IPv4 ma che permettono di inserire una subnet.

Entrambi non possono essere usati su Internet.

Indirizzi IPv6 EUI-64

(RFC 2373)



Abbiamo detto che la scheda di rete ha un suo MAC address fatto di 6 Byte (2^{48} indirizzi), in cui i primi tre rappresentano la localizzazione (chi ha prodotto la scheda, etc...) e gli ultimi tre rappresentano la scheda prodotta.

L'RFC prevede questa operazione: Prendere i primi 3 Byte, scriverli all'inizio dell'indirizzo, aggiungere una coppia FF:FE ed infine copiare gli ultimi 3 Byte. Passiamo da 6 a 8 Byte.

Mettiamo però una ulteriore differenza: nel primo Byte ci sono due bit particolari: (0000 00**xx**)

Bit Rosso: Indica se è Globally Unique o Di Sito (cioè se c'è 1, allora non identifica un ente)

Bit Blu: Indica se è un indirizzo unicast o multicast.

Il protocollo forza a 1 il penultimo bit. Viene creato un indirizzo IPv6 valido che però **non può navigare su internet**.

Ogni macchina può crearsi il suo indirizzo IPv6 univoco, perché dato che gli indirizzi MAC validi dovrebbero essere univoci, allora la probabilità di creare due indirizzi IPv6 duplicati partendo da indirizzi MAC è prossima allo zero.

Ovviamente non segue un comportamento geografico perché parte dal MAC Address. Quindi, per fare un esempio, se dovessi usare il DHCP in questo caso non c'è più bisogno di mettere 0.0.0.0 (un indirizzo illegale) ma posso usare un IPv6 assolutamente valido.

Windows (+alcune distro linux) non usa questa struttura. È completamente random.

Lezione 13 - 06/05/2020

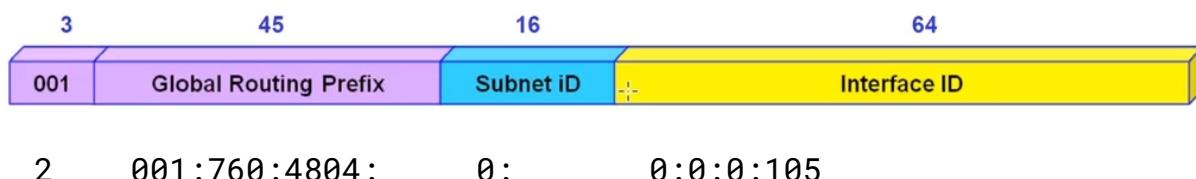
Recap

Avevamo visto il formato di indirizzo di IPv6, 8 blocchi da 16 bit ciascuno (4 bit per singola cifra), per un totale di 128 bit. La notazione può essere semplificata con i doppi due punti (::), dato che per i primi anni (decenni?) di adozione di IPv6 tutti gli indirizzi saranno caratterizzati da una lunga serie di zeri all'inizio.

l'Indirizzo IPv6 dell'Università di Catania:

2001:760:4804:0:0:0:105	Singola macchina
2001:760:4804::/48	Sottorete (con maschera)

Rivisto in Global Unicast Address:



I primi 3 bit (che poi in realtà sarebbero 4, ma il quarto viene raggruppato nel restante blocco da 45) sono fissi, il primo valore potrebbe essere o 2 o 3. (001 diventa 0010 o 0011).

Il blocco da 45 ha pure una sua suddivisione interna per migliorare ulteriormente il routing.

Poi abbiamo la subnet, e considerando che la maschera è /48, il resto dell'Indirizzo può essere gestito internamente all'università.

Approfondimenti IPv6

Gli indirizzi IPv6 sono divisi in 3 grossi gruppi: Unicast, Multicast e Anycast.

Unicast

Gli indirizzi Unicast iniziano con 2 o 3 come primo bit, cioè iniziano con 001.

Multicast

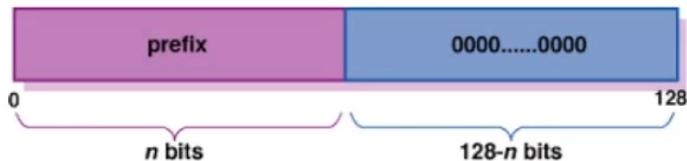
Gli Indirizzi Multicast iniziano con FFxx, i byte xx possono variare, e ci sono vari tipi (molti sono Link-Local, cioè non possono navigare su internet, altri Site-Local, cioè possono navigare dentro una rete di sito ma non su Internet globale).

Uno di quello che ci può interessare è FF02::2, "All-routers address".

Mandando un pacchetto a un indirizzo multicast, tutti i router che sono nel link locale risponderanno e riceveranno questo pacchetto come se fosse indirizzato a loro.

Non è un vero e proprio broadcast perché ovviamente il multicast è ristretto a un determinato sottogruppo dei possibili host indirizzabili (mentre broadcast non fa distinzione).

Anycast



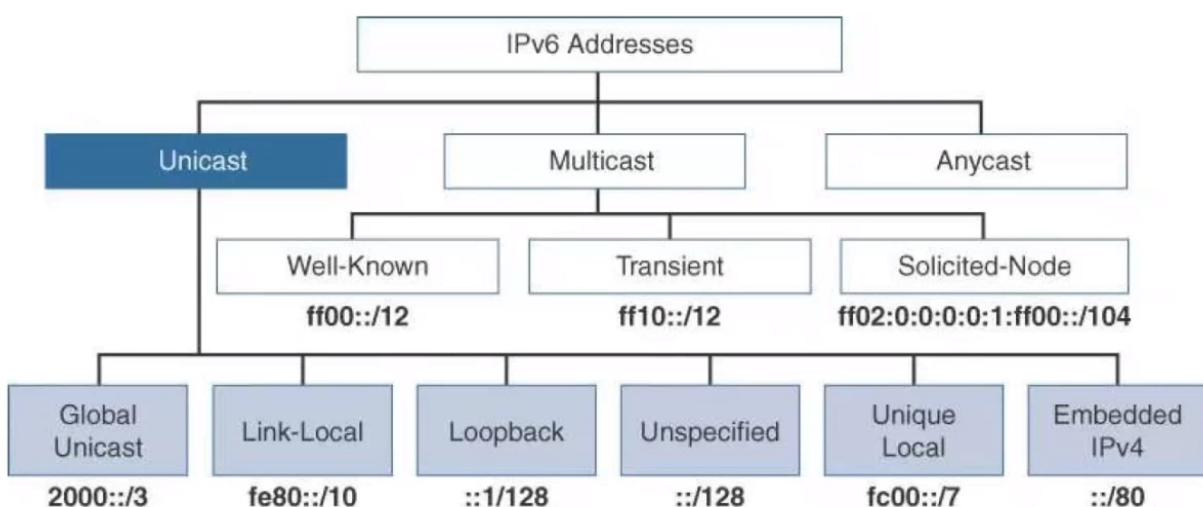
Hanno la parte finale tutti zero, potremmo confonderlo con un indirizzo unicast relativo a una sottorete. Non è una confusione erronea ma è stato fatto di proposito.

L'indirizzamento Anycast è del tipo "Io voglio rintracciare la prima macchina, che appartiene a questo schema di indirizzamento, che può realizzare un dato servizio". (Un po' come l'autocomplete dell'* in bash).

Ad esempio, un servizio distribuito di macchine tutte equivalenti tra di loro.

L'idea è: non devo indirizzare a una singola macchina, ma indirizzo qualcosa all'interno di una organizzazione, da cui il prefisso, che può essere anche quello di una rete vera e propria, dopodiché il primo router che riesce a gestire l'Anycast indirizzerà il pacchetto verso la macchina più conveniente.

Quindi Anycast sono macchine indirizzabili in Internet, semplicemente invece di indirizzare una singola macchina si indirizza a un Router che poi smisterà il pacchetto.



IPv6 over Ethernet

I pacchetti IPv6 vengono incapsulati in frame Ethernet esattamente come per IPv4, l'unico cambiamento è nel campo Ethertype: 86DD (invece di 0800).

Permette di capire velocemente se in una frame Ethernet è contenuto un pacchetto IPv4 o v6

Neighbour Discovery Protocol

RFC 4861 (evoluzione di 2461).

Protocollo che scopre quali sono i “vicini”:

Fornisce i seguenti servizi:

- Router discovery
- Prefix discovery
- Parameter discovery
- Address Autoconfiguration
- Address resolution
- Next-hop determination
- Neighbor unreachability detection
- Duplicate address detection
- Redirect messages

Possiamo scoprire chi sono i vicini presenti nella nostra rete e i router nella nostra rete proprio grazie agli indirizzi Multicast. Non si deve configurare dato che è già disponibile in IPv6.

Un'altra cosa importante è che va a implementare quello che era il protocollo ARP dentro il protocollo IPv6.

Infatti ARP non faceva parte di ICMPv4 (lo vediamo tra poco), mentre NDP fa parte di ICMPv6.

Prevede 5 tipi di messaggi:

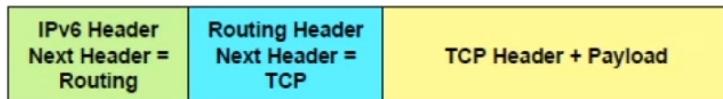
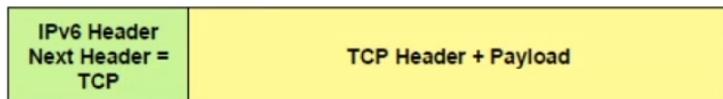
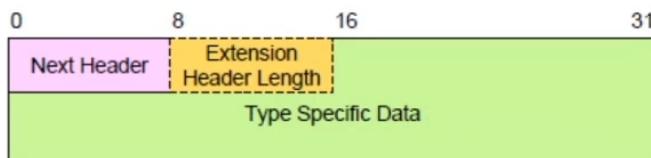
- Router Solicitation (ICMP type 133)
- Router Advertisement (ICMP type 134)
- Neighbour Solicitation (ICMP type 135)
- Neighbour Advertisement (ICMP type 136)
- Redirect (ICMP type 137)

Classi DI Traffico in IPv6

000-111 = **time insensitive** -- I primi tre bit indicano quanto è sensibile al tempo la comunicazione, essenzialmente è un campo per dire al router di dare una priorità ai pacchetti più sensibili al tempo.

1000-1111 = **priorità dei pacchetti**. Quanto fare andare avanti i pacchetti, cosa scartare e cosa non scartare assolutamente.

Header Opzionali



AKA Extension Header.

Nell'header di IPv6 c'è un campo "Next Header". Mentre in IPv4 l'idea era di mettere eventuali opzioni aggiuntive nel campo "Options" apposito, il che rendeva la dimensione dell'intestazione variabile (cioè 20 Byte era il minimo possibile), e questo era scomodo.

In IPv6 si ha invece un'intestazione fissa, più snella possibile. Quindi eventuali opzioni aggiuntive sono messi in quelli che vengono chiamati "Header aggiuntivi".

La prima figura è un pezzo di righe di 32 bit ciascuna che si aggancia subito dopo l'intestazione. Il campo Next Header dice "a seguire cosa c'è?", E subito dopo viene specificato quanto è grande il pezzo che segue, cioè quante righe ci sono.

Quindi è qualcosa di simile (ma non proprio) a una lista linkata. Dopo seguono vari esempi.

In generale la struttura sarà sempre:

Header IPv6 > Next Header = [qualcosa] > dentro [qualcosa]: Next Header=[quals'altro] > ...

Questo permette anche a IPv6 di frammentare.

Gli Header opzionali sono indicati da un numero. In particolare:

6: TCP

58: ICMP

59: NULL, nessun header a seguire. Se il primo valore Next Header fosse 59 avremmo solamente 40 Byte di header.

Frammentazione IPv6

In IPv4 qualunque nodo può frammentare.

In IPv6 ciò è scoraggiato, infatti l'MTU minima è 1280 Byte, con 1500 raccomandati (non è un caso).

Inoltre, solo il mittente può frammentare con un Fragment (Optional) Header.

In generale conviene capire qual è l'MTU massima che il canale può sopportare.

L'Header frammentazione dovrebbe essere messo prima del blocco TCP ovviamente. Questo perché il pacchetto può essere frammentato solo dopo il Fragment Header, tutto ciò che è prima viene considerato da non frammentare.

Sicurezza

Tra i vari header opzionali esiste un Authentication Header. Garantisce autenticità e correttezza del pacchetto.

Inoltre esiste anche un Encrypted Security Payload Header.

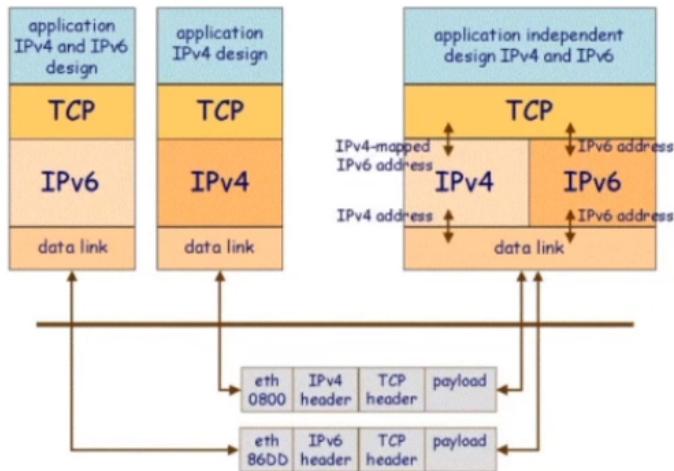
Essenzialmente sono state riciclate tutte le “pezze” di sicurezza poste a IPv4 in modo più organico e integrato con IPv6.

Migrazione da IPv4 a IPv6

Come affrontare il passaggio tra un protocollo all'altro?

Approccio Dual Stack

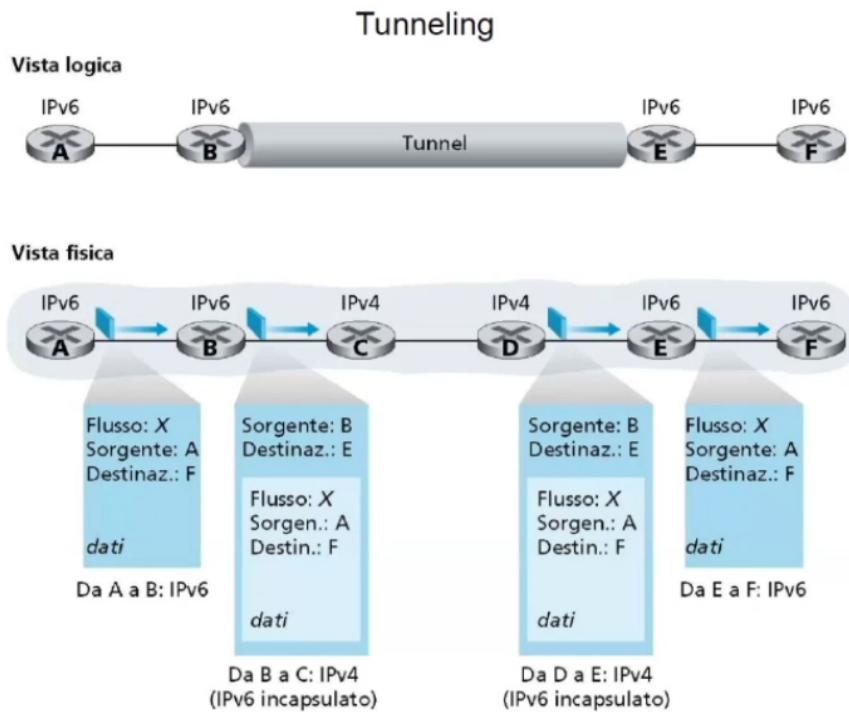
L'approccio più indolore è il dual-stack.



Nella nostra rete domestica possiamo mettere tutti e due gli stack, sia v4 che v6, se arriva una frame Ethernet faremo la distinzione tramite il campo Ethertype. A seconda di quello che indica il campo sapremo se la frame dovrà salire la pila protocollare con IPv4 o IPv6.

Per l'applicazione e per TCP non cambia tanto. Subito dopo TCP già si sa che strada seguire.

Tunneling



Il problema si pone quando noi siamo una macchina IPv6 e vogliamo parlare con un'altra macchina IPv6, passando attraverso una rete IPv4.

Il problema contrario non si pone perché gli indirizzi IPv4 sono mappabili dentro IPv6.

La soluzione è di creare un tunnel: Cioè quando avviene il passaggio da router IPv6 a router IPv4, i pacchetti devono essere v4, una volta usciti dalla rete v4 devono tornare ad essere v6. Nel caso della figura i router coinvolti sono B ed E.

Un tunnel si realizza facilmente: Il router di transito (B) prende il pacchetto v6 e lo incapsula in un altro pacchetto IPv4. Cambiano Sorgente e destinazione ovviamente. La sorgente ovviamente è il router di transito in entrata (B) e la destinazione è il router di transito in uscita (E). La macchina in uscita toglie il pacchetto IPv4 e lo fa viaggiare come IPv6.

Non c'è bisogno di dire al router in entrata che seguono dei router IPv4 perché già lo sa per forza (*"brutalmente, gliel'ha detto chi ha attaccato i cavi"*).

Essenzialmente quindi il pacchetto IPv6 è tutto incapsulato nel campo dati IPv4, non c'è cambio di intestazione o altro.

ICMPv4

Internet Control Message Protocol, permette di mandare messaggi di controllo che non richiedono porte di comunicazione.

Ad esempio il ping sono due righe: domanda e risposta. (echo request, 8, e replay, 0).

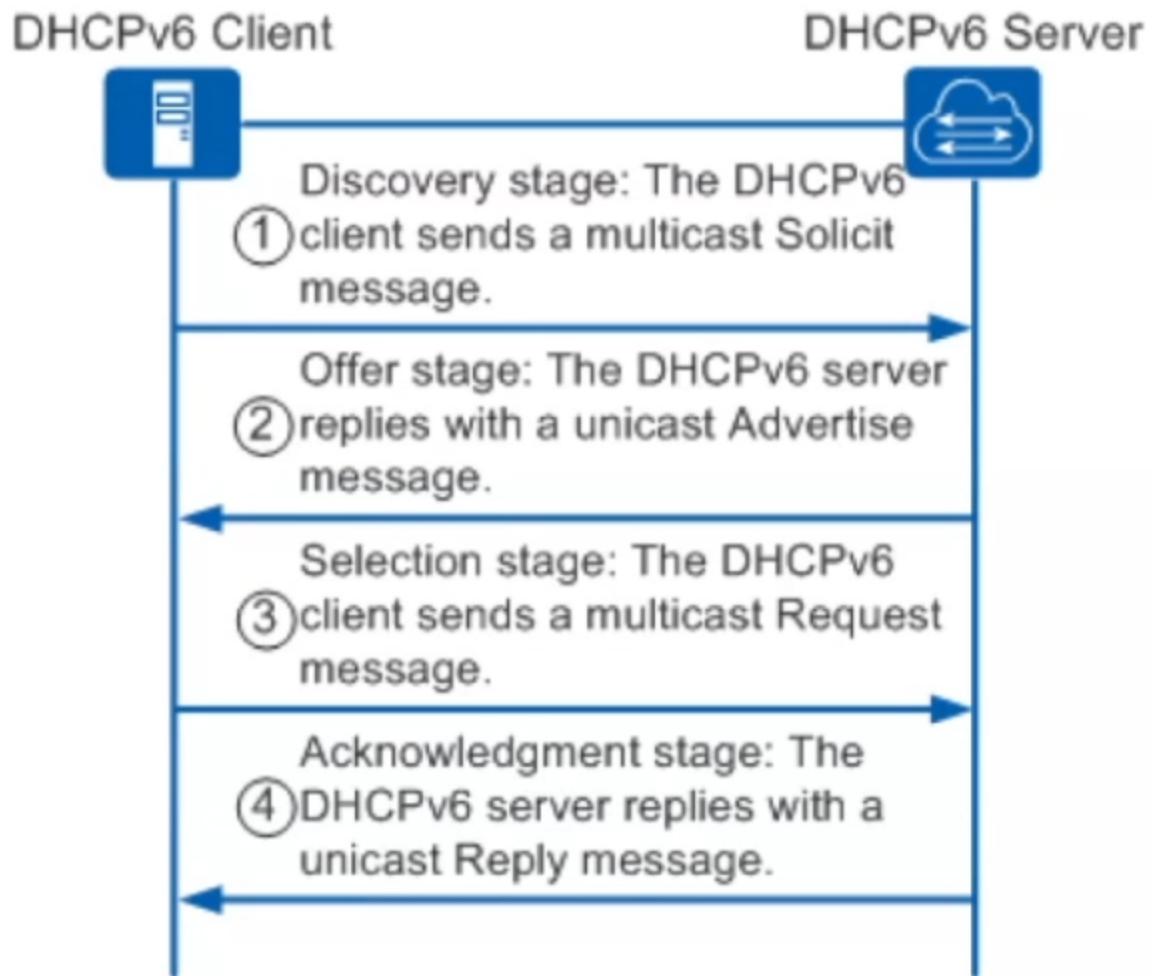
Essendo parte di questo protocollo può essere disattivato per far diventare la macchina invisibile alla rete. La macchina è on e risponde alle comunicazioni, semplicemente non risponde al ping.

Un altro comando interessante è il Traceroute: indica qual è la strada usata per arrivare alla destinazione. Funziona usando il messaggio ICMP di TTL scaduto, si incrementa di 1 ogni volta. Questo perché altrimenti i router di mezzo inoltrerebbero subito il pacchetto senza fare nessuna modifica e non saprei niente. Invece col TTL sono obbligati a rispondere obbligatoriamente.

ICMPv6

Versione IPv6 di ICMP. Conviene ping, NDP, è un po' differente.

DHCPv6



Il protocollo è tale e quale a IPv4, ma ci sono dei cambiamenti:

Discovery: Manda un messaggio Multicast per sollecitare i server DHCP a rispondere. Cioè non viene mandato un messaggio broadcast che va a intasare tutta la rete, ma viene mandato solo a quelle macchine che sanno di essere un server DHCP. Inoltre non ha un indirizzo sorgente non valido (tutti zeri), ma ha un indirizzo local link valido. Così il mittente è ben identificato.

Offer: La risposta è unicast infatti, grazie all'indirizzo local link.

Selection (Request): La scelta dell'indirizzo IP viene mandata nuovamente in multicast.

Acknowledgement: Viene mandata la conferma con una Unicast reply. Ovviamente tutto quanto è a blocchi di 128 bit, maschera compresa.

Lezione 14 - 11/05/2020

Algoritmi di Routing

Sono algoritmi che procedono in automatico, cioè non agiscono manualmente sulle tabelle di routing. Ma ovviamente il risultato di questi algoritmi va a modificare le tabelle viste precedentemente. Questi algoritmi automatizzano il processo di aggiornamento della tabella, anche tenendo conto dello stato di evoluzione della rete.

Distinguiamo gli algoritmi in due categorie: Statici e Dinamici, oppure a informazioni Locali e Globali.

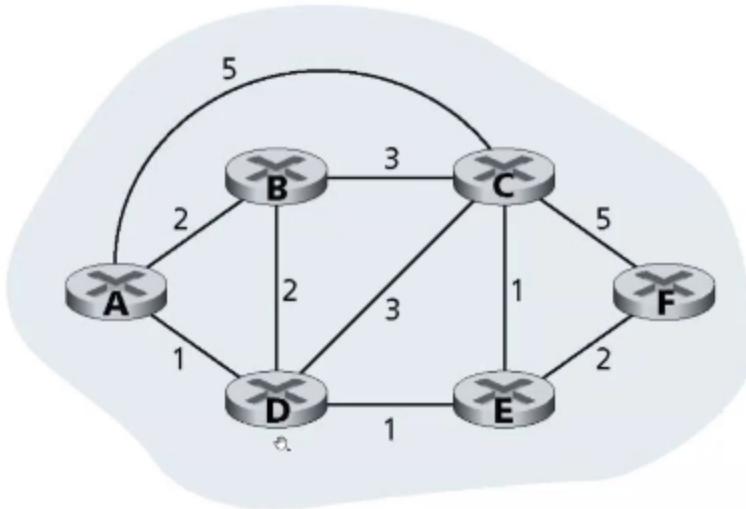
Premettiamo che la rete viene rappresentata sotto forma di grafo, i nodi sono i router e i collegamenti vengono rappresentati da archi, i quali archi hanno un peso, che rappresenta il costo di utilizzo di quel determinato arco da un nodo all'altro.

- Statici: Viene esaminata la struttura del grafo in un dato istante (“fotografia”, “snapshot”), e dopodichè computa le decisioni. L'informazione calcolata viene poi distribuita a tutti i nodi partecipanti, cosicché ogni nodo sappia come raggiungere le altre destinazioni. Tuttavia questi pesi potrebbero variare, per cui si potrebbe pensare che sarebbe più opportuno avere un approccio dinamico.
- Dinamici: Esamina di volta in volta la situazione dei pesi per poter prendere la decisione migliore a seconda dei mutamenti nella rete. Questo approccio però è molto pesante e costoso, perché avere le informazioni dei link non è semplice. Soprattutto perché non solo mi devo accorgere del cambiamento ma devo comunicarlo a tutti gli altri nodi della rete tramite messaggi. Chiaramente se questo scambio di informazioni è fatto troppo frequentemente appesantisce la rete. Causerei congestione.

Infatti teoricamente l'approccio dinamico è migliore per quanto riguarda il routing, ma se fatto ingenuamente diventa controproducente perché causa troppo traffico.

Non è una soluzione aggiornare la nostra “fotografia” del grafo poco frequentemente, perché avere informazioni vecchie non ci serve a niente (e anche questo diventa controproducente).

- Locali: Per esempio, al nodo E arriva un messaggio dal nodo D da indirizzare a B. E deve prendere una decisione, deve farlo passare per F, per C, o anche D stesso. Tornare indietro dovrebbe essere da evitare, ma dato che siamo in un approccio locale, il nodo prende la decisione secondo le informazioni scambiate con i nodi vicini, che potrebbero anche essere incongruenti con informazioni prese in altri punti della rete.
Ognuno prende le decisioni in maniera autonoma, e questo ovviamente potrebbe portare a dei problemi (approfonditi tra poco). Ad esempio, un pacchetto potrebbe rimbalzare all'infinito, non tanto tra due nodi, ma tra 3 o più nodi è interamente possibile.
- Globali: Vediamo tutta la situazione e decidiamo qual è il discorso migliore, a partire da qualunque nodo del grafo. Però per fare una cosa del genere servono troppe, anche tante informazioni



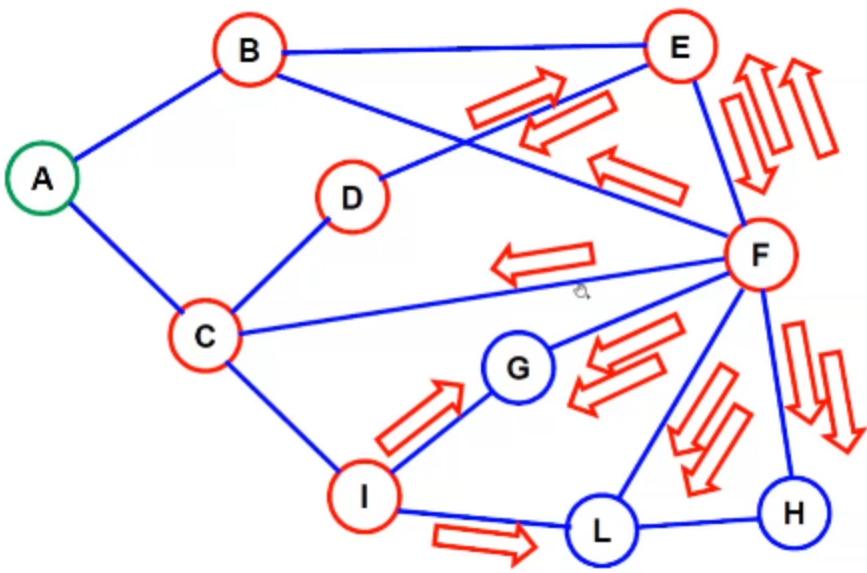
Flooding

“Soluzione quasi perfetta”.

È l'approccio migliore per certi aspetti, perché trova la strada migliore nel minor tempo possibile, senza aver bisogno di particolari informazioni.

Non sapendo che strada seguire, il nodo mittente fa una copia del pacchetto da spedire e lo manda a tutti i nodi a cui è connesso. Questi nodi a sua volta faranno una copia e lo manderanno a tutti i nodi a cui sono connessi, esclusione fatta ovviamente per il nodo dal quale hanno ricevuto questo pacchetto.

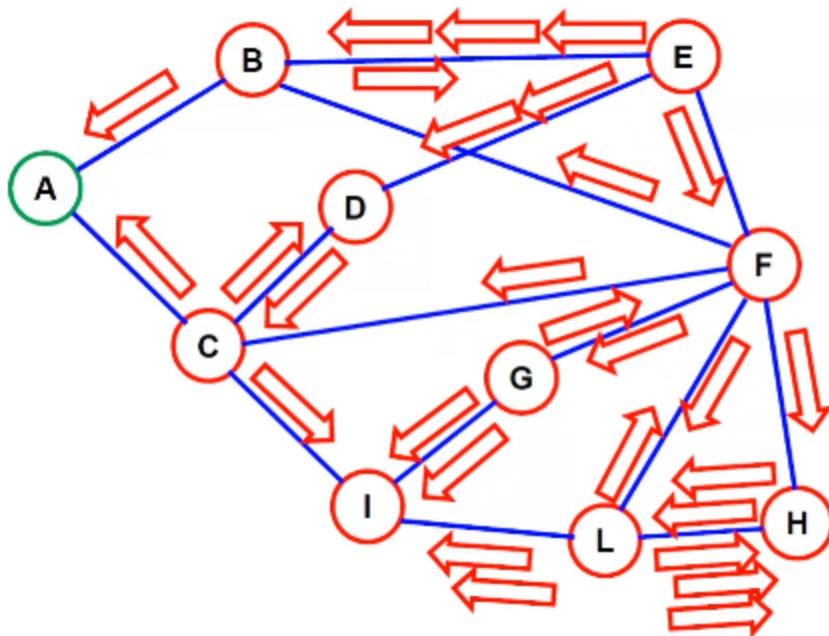
Il problema si ha quando si raggiunge la situazione che due nodi non sanno ancora che l'altro ha appena ricevuto il pacchetto di flooding, vedi D ed E in questa figura:



Altre situazioni interessanti:

F riceve la sua copia da B, dopodiché ricopia e rimanda nei suoi 5 link. Poi riceve la stessa copia da C, e non sapendo che sono gli stessi replicherà di nuovo sui link necessari. Se diamo la possibilità ad F che questi due pacchetti sono uno la copia dell'altro, allora F rispedirà indietro solo una di queste copie.

Se non sa riconoscere le copie, allora spedirà queste copie duplicate, ma riceverà anche altri pacchetti, ad esempio da E, e si avrà una situazione del genere:



Il numero di pacchetti cresce e la rete viene tempestata di copie dalla stessa origine. Addirittura tornano all'origine. In pochi passi comunque si ha il percorso migliore. In particolare, il numero di passi risulta essere il "diametro" del grafo. Nel caso della figura il numero di passi richiesti, che coincide con il diametro, è 3.

Il problema è bloccare il flooding prima che arrivi a inondare la rete di pacchetti duplicati, cioè di controllare il flooding.

Se ci riusciamo, avremo un protocollo che trova il percorso migliore con un numero di passi estremamente basso.

Inoltre riesce a trovare il percorso verso la destinazione, non importa quanto complicato possa essere, semplicemente perché prova tutti i percorsi possibili.

Primo modo di controllare il flooding:

Mettiamo un HOP limit (praticamente un TTL), magari pari al diametro del grafo.

Ad esempio con un diametro pari a 4 non si arriverebbe alla situazione vista nell'ultima figura.

Però dovremmo conoscere a priori il diametro del grafo, cosa non semplice.

Secondo modo di controllare il flooding:

Far capire alla destinazione che i plurimi pacchetti che possono arrivargli sono tutte copie. Oltre che alla destinazione, questa capacità potrebbe essere estesa anche ad altri nodi. Si potrebbe mettere un ID al pacchetto ad esempio, e il flooding verrebbe bloccati con un numero di salti superiore di 1 al diametro del grafo.

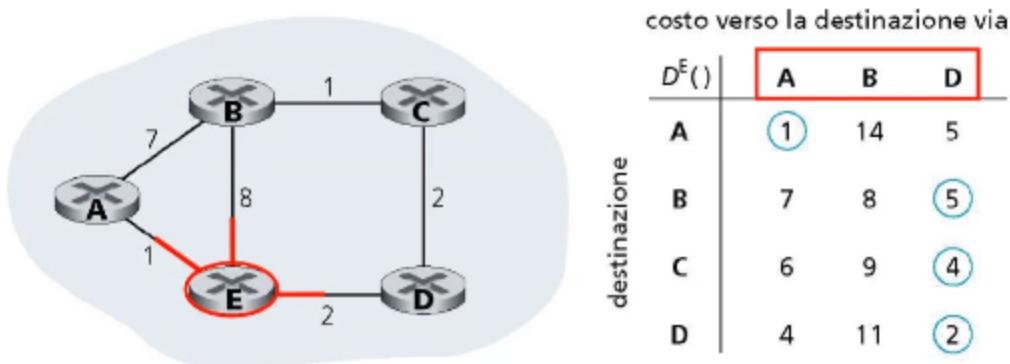
Così avremo un piccolo picco di congestione, ma niente di più.

Questa soluzione non si usa perché a questo punto F (ma anche gli altri nodi) dovrebbe tenere una tabella di tutti i pacchetti che ha visto, e confrontare ogni pacchetto in arrivo con questa tabella che comunque crescerebbe di dimensioni. Quindi questo approccio è intensivo dal punto di vista della memoria e del tempo di elaborazione. Funziona solo se il flooding viene utilizzato solo di rado.

Quindi in short: trova il percorso migliore, raggiunge in ogni caso la destinazione, qualunque sia la metrica utilizzata, può toccare tutti i nodi e quindi può essere visto come una forma di broadcast, solo che se non controllato satura la rete fin troppo velocemente.

Distance Vectors

Nelle prime implementazioni di Internet si pensò di utilizzare un approccio locale e statico; statico perché non teneva conto dell'evoluzione dei link, ma la filosofia principale è appunto l'approccio locale: ogni nodo decide in base in sue considerazioni personali ricavate da informazioni dirette o “*per sentito dire*” (si può anche dire che è distribuito e asincrono). Si basa sull'algoritmo di Bellman-Ford. I costi minimi per i percorsi sono correlati dalla formula: $d_x(y) = \min_v \{c(x,v) + d_v(y)\}$, dove $d_x(y)$ è il costo del percorso a costo minimo dal nodo x al nodo y , \min_v riguarda tutti i vicini di x (vd. Kurose per approfondimento)



Consideriamo il nodo E di questo grafo. E costruisce la sua tabella [infatti è indicata con $D^E()$], di norma è rettangolare, dove ci sono tante colonne quante sono le possibili uscite, e tante righe quanti sono i nodi presenti nella rete, esclusione fatta per il nodo stesso.

La tabella si legge così: Devo raggiungere il nodo A usando come prima uscita A, qual è il costo migliore? Questo è banale dato che c'è un collegamento diretto di costo minimo tra i due nodi.

Un altro esempio è arrivare ad A passando per D. In questo caso si ha una situazione dove il pacchetto esce da E e va verso D, e una volta arrivato lì il router lo rispedisce verso E, e finalmente verso A. Quindi di fatto torna indietro perché quello è il percorso migliore, dopo la “forzatura” imposta da noi di passare prima per D. Il costo infatti è $5=2+2+1$.

Un'altra situazione un po' strana è di usare come prima uscita B per andare da A. Andare direttamente ad A costerebbe $8+7=15$, tornare indietro per E sarebbe ancora peggio perché ci costerebbe 17, e infatti l'algoritmo, una volta arrivato a B, seguirà il percorso B->C->D->E->A, per un totale di 14.

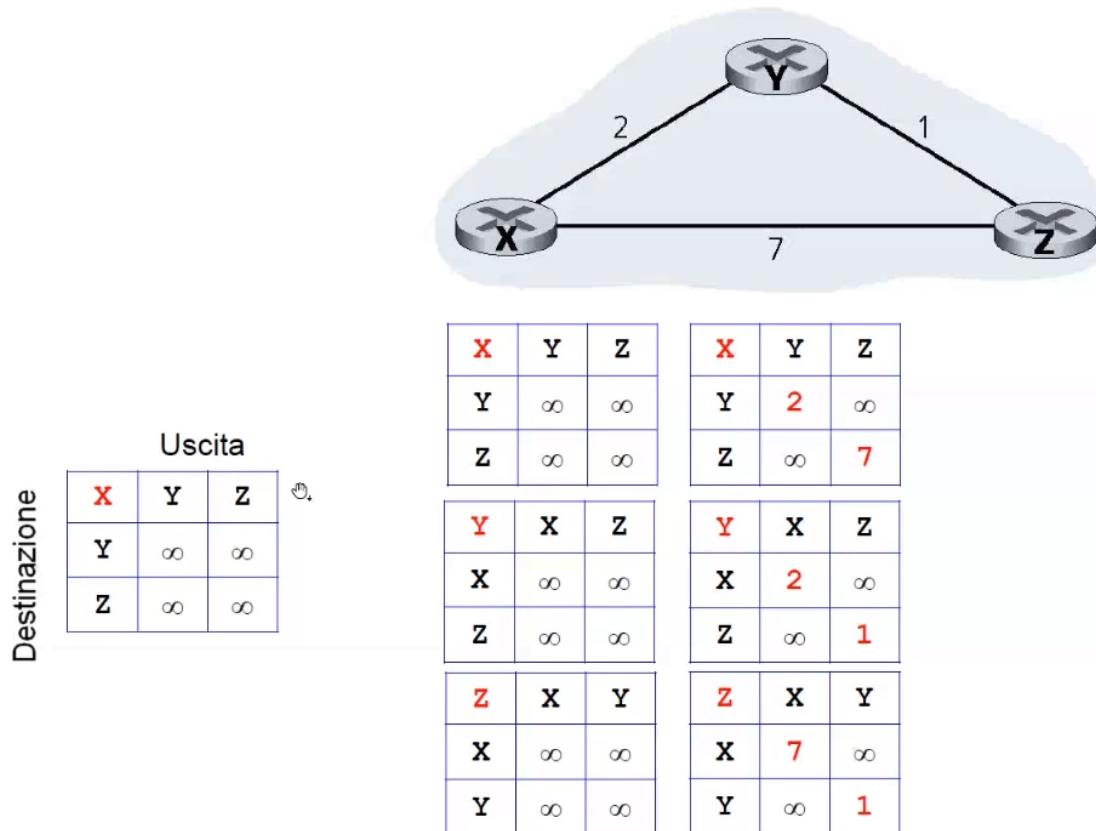
Quindi la tabella ci sta dicendo qual è il percorso minimo per quanto riguarda questo router, poi se il pacchetto ripassa per se stesso, questo il router non può saperlo.

Ovviamente noi vediamo il grafo nella sua interezza, se potesse vederlo anche l'algoritmo non avrebbe senso usare questa tabella perché saprebbe qual è il percorso migliore.

Costruzione Tabella DV

La cosa interessante della costruzione della tabella dei distance vector è che il nodo E inizia a costruirla essendo a conoscenza solo degli archi da lui uscenti. Ad esempio, non ha idea dell'esistenza dell'arco da A a B, da D a C e via dicendo. Ricaverà l'esistenza di questi archi da informazioni indirette dai router da lui vicini. Per "passaparola" per così dire.

Basta però che un router immetta informazioni errate per avere brutti problemi che vedremo dopo.



Utilizziamo il grafo sopra e in questo caso abbiamo una tabella quadrata, ma nella realtà si ha spesso una tabella rettangolare.

Inizialmente il nodo non conosce niente e viene messo tutto a infinito.

1. Primo Passaggio: X riesce a capire da solo che c'è un peso 2 verso Y e 7 verso Z. Gli altri due nodi Y e Z faranno lo stesso dal loro lato. Gli altri valori rimangono a infinito per il momento
2. Secondo Passaggio: i nodi adiacenti tra loro comunicano tra di loro una versione "sintetizzata" della propria tabella, ossia manda un vettore con i pesi migliori che lui conosce verso gli altri nodi (cioè distanze, da cui quindi il nome del protocollo). Quindi il peso minimo di ogni riga.

X	
Y	2
Z	7

Y	
X	2
Z	1

Z	
X	7
Y	1

3. Terzo Passaggio: Le informazioni raggiungono i nodi. Ad esempio, arriva l'informazione che Y riesce a raggiungere Z con un costo 1. A questo punto X, nel campo Uscita: Y verso Z deve sommare il costo per arrivare da X (se stesso) a Y, e da Y a Z.

Si fida ciecamente dell'informazione che arriva da Y. "Per sentito dire"

X	
Y	2
Z	7

X	Y	Z
Y	2	8
Z	3	7

Y	X	Z
X	2	8
Z	9	1

Z	X	Y
X	7	3
Y	9	1

4. Quarto Passaggio: A questo punto abbiamo un nuovo vettore, X spedirà 2,3; Y spedirà 2,1 e Z 3,1

Una volta che si arriva alla situazione migliore l'algoritmo si ferma.

Quindi l'algoritmo parte quando c'è una situazione in cui i nodi non sanno il costo di raggiungimento di altri nodi, o quando c'è una modifica, e si dice che **converge** rapidamente in pochi passaggi, e ha un punto **ben preciso** in cui si ferma, non ci sono

dubbi su quando l'algoritmo si ferma.

Si può infatti sapere qual è il valore massimo del numero di passi a priori (non è detto che raggiungerà il valore massimo [ndr: significa che può convergere prima?]).

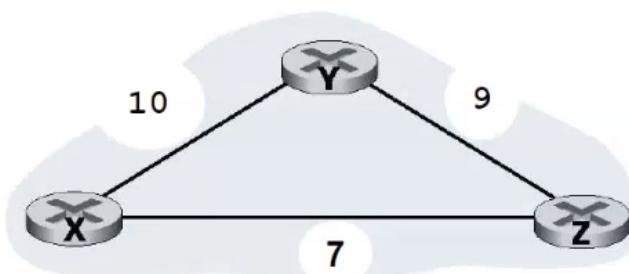
C'è però già un problema in questa situazione: Se fosse una situazione reale, il nodo Y si sovraccaricherebbe molto rapidamente, in quanto X e Z spediscono tutto il loro traffico attraverso di esso, reputando inutile il collegamento diretto di costo 7.

Cioè l'algoritmo non sta tenendo conto delle conseguenze future sullo stato della rete, ma solo di trovare il percorso migliore in un dato grafo con dati pesi.

La situazione muterà sicuramente in futuro.

Miglioramenti

Prendiamo un altro grafo:



Con le seguenti tabelle:

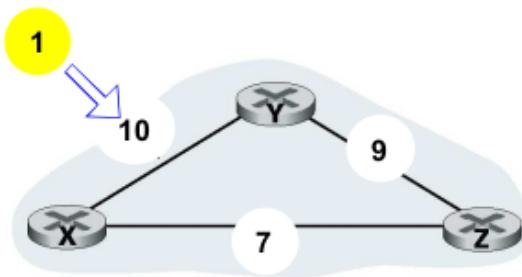
I Step:

X	Y	Z	Y	X	Z	Z	X	Y
Y	10	∞	X	10	∞	X	7	∞
Z	∞	7	Z	∞	9	Y	∞	9

II Step:

X	Y	Z	Y	X	Z	Z	X	Y
Y	10	16	X	10	16	X	7	19
Z	19	7	Z	17	9	Y	17	9

Al II step le tabelle sono di fatto definitive. Supponiamo ora che il peso tra il nodo X e il nodo Y cambi, da 10 a 1.



X	Y	Z
Y	10	16
Z	19	7

X	Y	Z
Y	1	16
Z	19	7

Y	X	Z
X	10	16
Z	17	9

Y	X	Z
X	1	16
Z	17	9

Z	X	Y
X	7	19
Y	17	9

Z	X	Y
X	7	19
Y	17	9

1

Per Z:
uscita X verso Y

I nodi che si accorgono di questo cambiamento sono ovviamente X e Y, che cambiano i loro rispettivi valori. Spediranno poi verso i propri vicini il proprio vettore delle distanze: X:{1,7}. Y:{1,9}

Il nodo Z riceve queste informazioni e cambierà i valori corrispondenti (se è effettivamente necessario).

A questo punto anche Z spedirà il suo vettore delle distanze dato che ha eseguito un cambiamento nella sua personale tabella.

Se non sono necessarie altre info l'algoritmo si ferma. Si può fermare in step differenti in vari nodi.

X	Y	Z
Y	1	16
Z	10	7

X	
Y	1
Z	7

X	Y	Z
Y	1	16
Z	9	7

X	Y	Z
Y	1	15
Z	8	9

X	Y	Z
X	1	16
Z	8	9

X	Y	Z
X	1	16
Z	8	9

Z	X	Y
X	7	10
Y	8	9

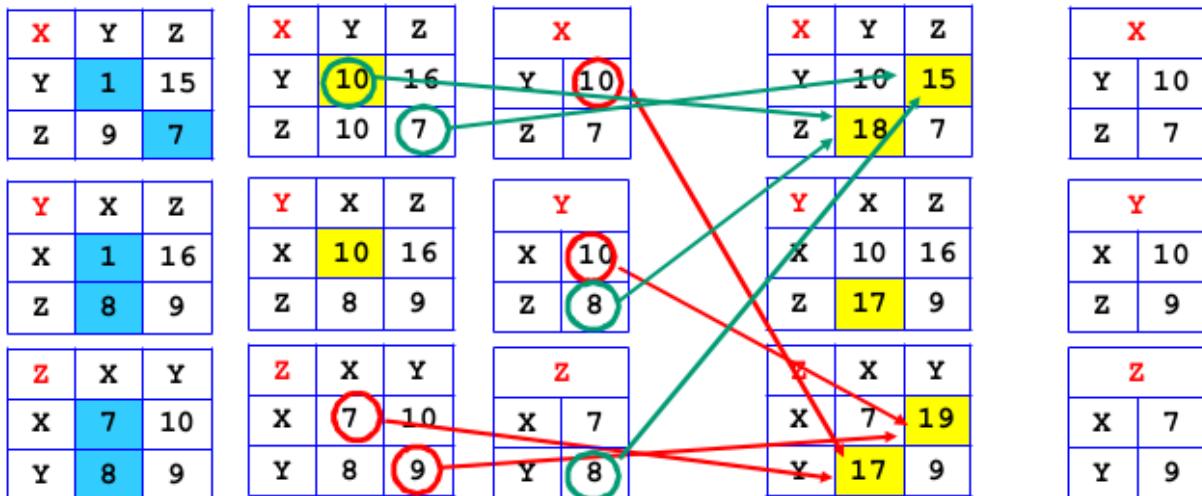
Z	X	Y
X	7	10
Y	8	9

Tabella migliorata completa

Peggioramenti

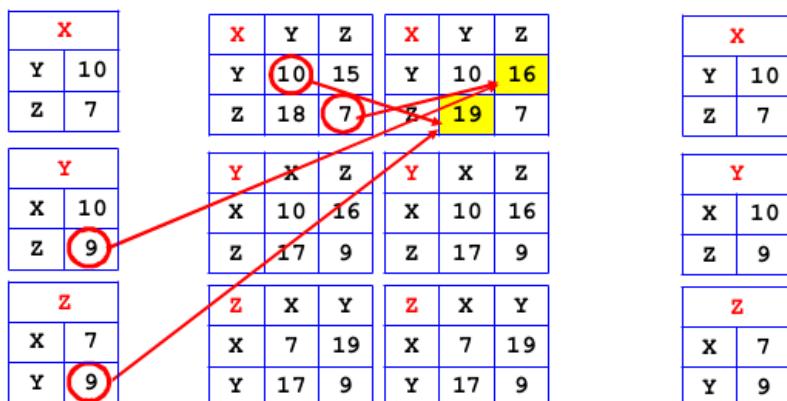
È un caso molto più brutto in cui ritrovarsi.

Il grafo è lo stesso di prima ma al contrario, tra X e Y, il peso varia da 1 a 10.



I primi nodi a cambiare i propri vettori sono sempre X e Y. Questi valori però ancora non sono "stabili". Ad esempio, nella tabella di X abbiamo un valore "18", ma componendo 10, 9 e 7 non possiamo ottenere 18.

Questo valore 18 deriva dall'8 presente nel vettore di Z, che si basava sul fatto di poter arrivare da X a Y con peso 1. Quindi è di fatto un valore intermedio falso. Anche 15, per fare un altro esempio. Non è un problema perché l'algoritmo non si ferma certo a questo step, perché gli altri nodi continueranno a cambiare.

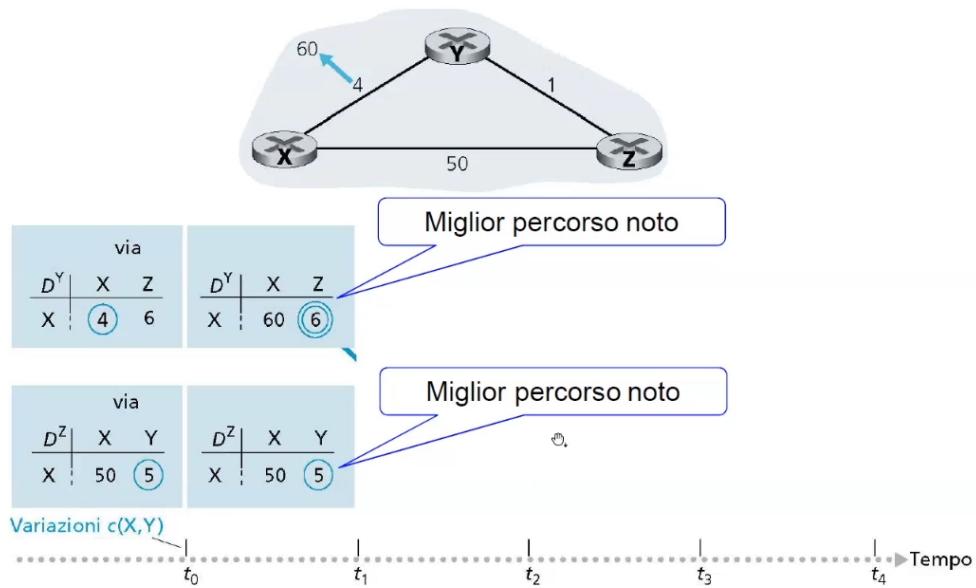


Step finale

Tutto sommato il sistema converge con pochi passi, in questo caso.

Il problema ovviamente è che durante la fase di convergenza ci sono dei dati falsi che potrebbero far girare in tondo i pacchetti, senza necessità.

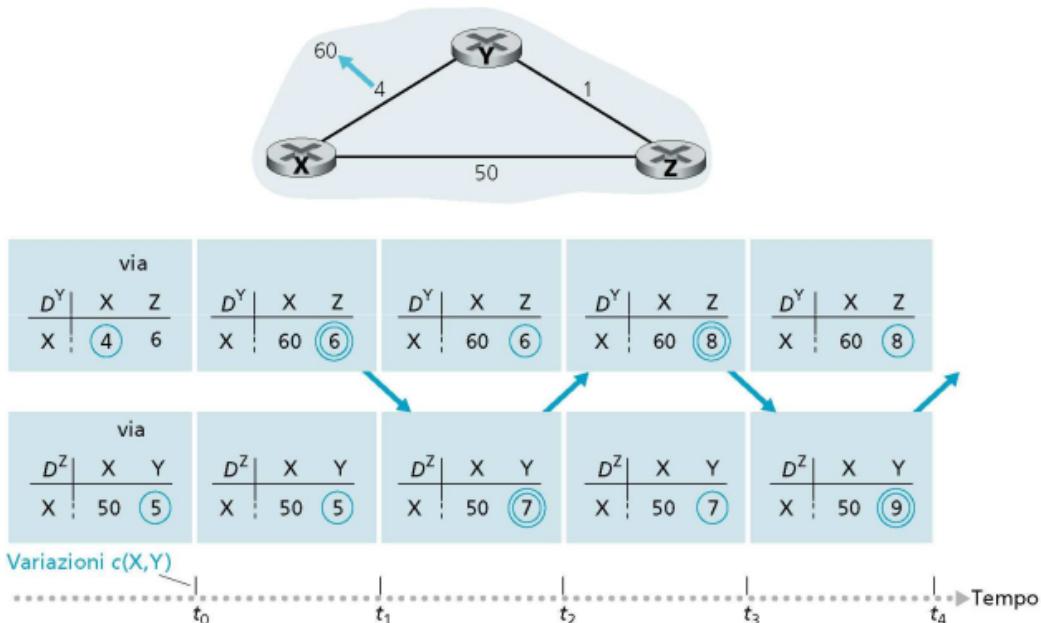
La situazione può diventare molto grave in certi casi particolari:



In questo grafo, per arrivare alla convergenza ci vorranno decine di passi, perché i rispettivi valori di D^Y e D^Z aumenteranno di 1 ad ogni aggiornamento (se l'arco tra Y e Z avesse avuto peso 2, gli incrementi sarebbero stati di 4).

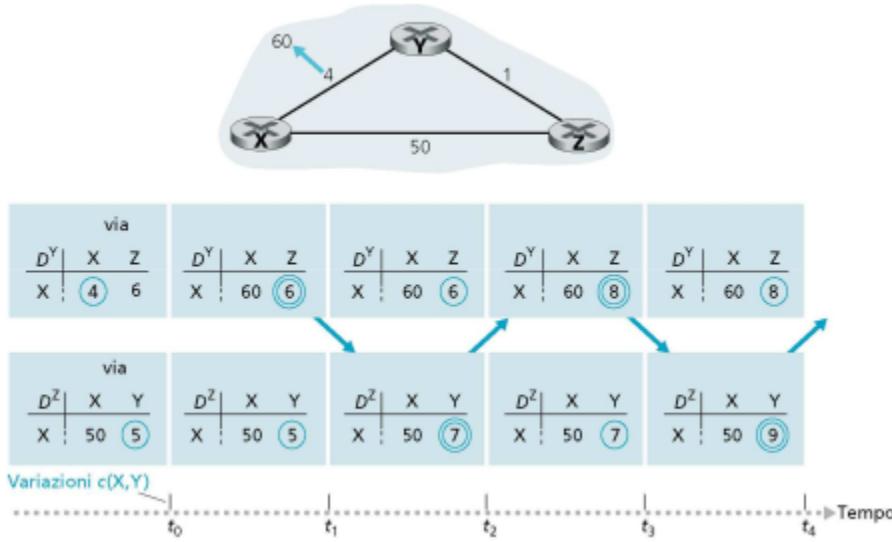
Converge sì, ma dopo molto tempo.

Per valori molto alti questa situazione raggiunge il punto in cui viene definita **Conteggio all'infinito**. (Anche se in realtà non è infinito).



In questo caso la convergenza non dipende dal diametro del grafo ma dalla particolare situazione dei pesi che sono stati mesi. Un modo per risolvere è il seguente:

Poisoned Reverse



Z dice a Y che la sua distanza verso X è infinita, se il suo percorso migliore verso X parte (passa) per Y.

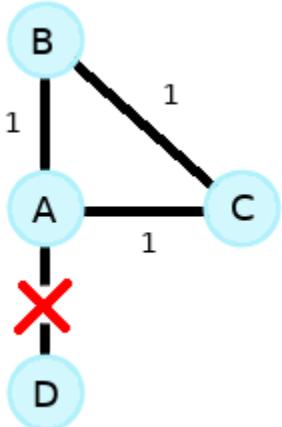
Questo perché non ha senso che un nodo dice a un suo adiacente di avere un percorso migliore verso un nodo remoto (non adiacente), se questo percorso passa per il suddetto nodo adiacente, perché il nodo adiacente dovrebbe già possederla.

Quindi la soluzione che viene utilizzata è che Z non manda ad Y il suo percorso migliore, ma "mente" e manda infinito, in modo tale che Y non fa l'errore contrario di raggiungere X passando da Z.

Z così sta bloccando un'eventuale informazione falsa.

ADDENDUM, DOMANDA ESAME:

-Esiste una configurazione in cui neanche la Poisoned reverse funziona? (sì)



Il collegamento tra A e D è rotto

A dice a B e C che D è irraggiungibile $DA(D) = \text{inf}$

B fa un nuovo percorso verso D attraverso C.

$$DB(D) = C(B,C) + DC(D) = 1 + 2 = 3.$$

B ora dice a C che D è irraggiungibile da $DB(D) = \text{inf}$ (a causa della poisoned)

B dice sempre ad A che ha un percorso verso D di costo 3.

A adesso calcola un nuovo percorso attraverso B.

A dice a C che D è raggiungibile (attraverso B)

Link-State

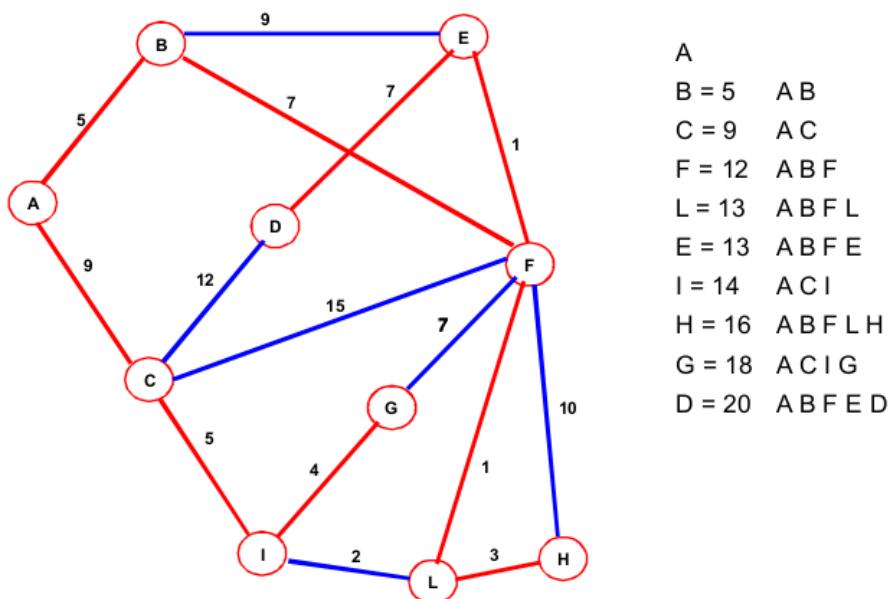
Il Distance Vector era stato pensato per gestire grafi dove i pesi non variavano, o al più variavano di rado, e soprattutto non teneva conto dello stato della rete.

L'algoritmo di Link State Routing mira a risolvere questi problemi.

Cerca di realizzare una fotografia completa del grafo, di tutti pesi, e avere un'informazione su come determinare i pesi. Anche lui non indica la metrica dei pesi. Si basa su Dijkstra. Un difetto dell'algoritmo è che non ci devono essere percorsi chiusi a peso negativo.

L'Algoritmo di Dijkstra, partendo da un nodo, deve trovare tutti i percorsi minimi verso tutte le destinazioni. Dopodiché si "blocca" il nodo a distanza minima raggiungibile e si aggiungono i nodi che quest'ultimo può raggiungere.

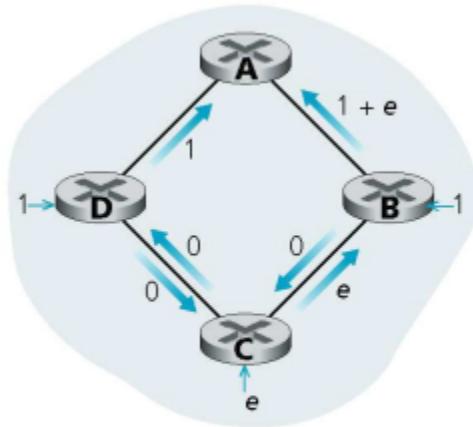
Una volta conclusa l'esecuzione avremo tutti i percorsi a distanza minima da A verso tutti gli altri nodi, ma contemporaneamente abbiamo trovato anche i percorsi minimi. Basta ricordarsi da dove "vengono fuori" i valori trovati. Cioè se scopriamo che il percorso a distanza minima da A per F passa prima per B, allora il percorso minimo sarà A B F.



Otterremo un albero di percorsi minimi (non ho cicli).

Il Link State routing funziona abbastanza bene ma ha anche lui dei problemi esistenziali.

Il peso è relativo allo stato dei link, varia dinamicamente nel tempo. Potrebbe succedere, avendo una struttura di questo tipo:



a. Routing iniziale

Abbiamo due percorsi quasi equivalenti, dato che Dijkstra sceglierà di passare, a partire da C, da D o da B per arrivare ad A, se al primo step viene scelto il percorso verso B, bypassando D, il traffico sul link ignorato potrebbe arrivare a 0. All'analisi successiva la prossima fotografia del grafo mostrerà un link completamente libero e uno invece molto carico. Allora l'algoritmo appesantirà D e ignorerà B, e via dicendo. In pratica c'è un'oscillazione. È un caso molto particolare, perché ci deve essere una struttura della rete che facilita questa situazione, e anche dei link con capacità simile. In ogni caso non sempre dà problemi

Confronto DV - LS

DV	LS
Decentralizzato	Globale
Messaggi solo per i vicini	Messaggi in Broadcast
Informazioni riguardanti tutte le destinazioni	Informazioni riguardanti solo i propri link
Conteggio all'infinito	Problemi di sincronia (oscillazioni)
Messaggi solo per variazioni sui link	Maggiore traffico di messaggi (invii periodici)
Poco robusto nei confronti di nodi maliziosi	Robusto agli attacchi

Distance Vector è decentralizzato, locale, ogni nodo esegue i conti per i fatti propri (asincrono). Link State è globale. Un'unica macchina prende la decisione su come trovare il percorso migliore in base a una fotografia dell'intero sistema. Poi magari l'implementazione manda la fotografia a tutti i nodi, in modo da far trovare a tutti la medesima soluzione.

DV si basa solo sui messaggi dei vicini, mentre LS, basandosi sullo stato dell'intera rete, ha bisogno di messaggi in broadcast.

La cosa paradossale è che DV ricevendo messaggi dai vicini ha informazioni su tutti, mentre LS ha informazioni principalmente per i propri link. Anche se qualcosa deve averla su tutti gli altri nodi, per poter lavorare. E non sapremo mai cosa.

DV ha il problema del conteggio all'infinito, LS le oscillazioni.

DV manda messaggi solo quando si hanno variazioni sui link, LS invece ha bisogno di invii periodici per capire qual è lo stato dei singoli link.

DV è poco robusto nei confronti di nodi maliziosi, mentre LS è robusto agli attacchi. Infatti un nodo potrebbe fingere di avere il percorso migliore (o peggiore) in assoluto a scopi maliziosi per dirottare il traffico. Nessuno può verificarne la correttezza. In LS non si può imbrogliare perché tutti i nodi hanno le stesse informazioni.

RIP

Routing Information Protocol, basato su Distance Vector, per i pesi del grafo usa come metrica gli HOP, il numero di salti effettuati da un nodo all'altro.
Non tiene conto della situazione del singolo link.

RIPv1

RFC 1058.

Il numero massimo di HOP è 15. Questo perché il numero massimo di HOP pone anche un limite al diametro del grafo, e un grafo con diametro troppo ampio risulterebbe lenta la convergenza dell'algoritmo.

Le tabelle di routing vengono scambiati ogni 30 secondi. Se un percorso non viene aggiornato per 180 secondi, la sua distanza viene posta ad infinito. Se trascorrono altri 120 secondi, ossia allo scadere del garbage-collection timer, il nodo irraggiungibile viene eliminato dalla tabella di routing.

Messaggi RIPv1

Utilizza due tipi di messaggi:

- REQUEST: Per chiedere info ai nodi adiacenti
- RESPONSE: Per inviare info di routing

Una tabella di routing RIP contiene:

- Indirizzo di destinazione
- Distanza dalla destinazione (in HOP)
- Next Hop: router adiacente a cui inviare i pacchetti
- Timeout
- GC-Timer

Datagramma RIP v1

Command	Version	Must Be Zero
Address Family Identifier		Must Be Zero
IP Address		
Must Be Zero		
Must Be Zero		
Metric		

A lunghezza variabile fino a 512 Byte (max 25 reti di destinazione).

Notiamo che non c'è informazione sulla maschera (cioè sono indirizzi Classful)

RIPv2

RFC 1723, 2453.

- Indirizzamento CIDR e VLSM
- Autenticazione dei messaggi
- Specifica del Next Hop
- Split Horizon, Poison Reverse.

Datagramma RIPv2

Command	Version	0000
0xFFFF		Autentication Type
Autentication		
Address Family Ident.	Route Tag	
IP Address		
Subnet Mask		
Next Hop		
Metric		

Viene mandato qual è il next hop per mandare conteggi all'infinito.

RIPng



Basata su RIPv2 esclusivamente per IPv6, per quanto non sia molto utilizzata.

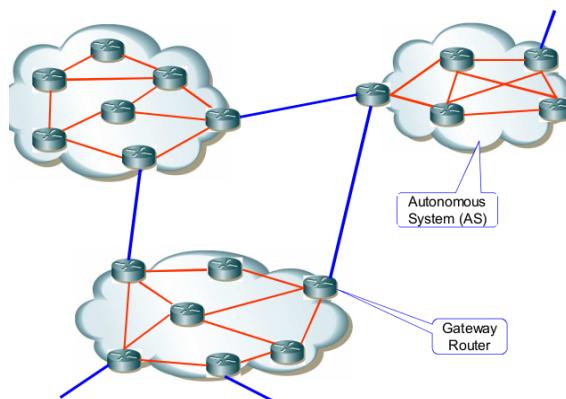
Non ha l'autenticazione

Autonomous System (AS)

Internet è composta da un numero enorme di router. Pensare di poter eseguire una fotografia per il LS, o eseguire DV su tutta la rete Internet è una cosa folle.

Quando c'è un passaggio da un provider all'altro, il routing su queste due reti differenti interessa ben poco l'un l'altro. Semmai quello che interessa è quali sono i punti di aggancio verso altre reti.

Ovvero, "noi" ci facciamo il routing nella nostra zona interna, poi sappiamo quali sono i punti di uscita, ci interessa al più sapere qual è il punto di uscita più corretto per arrivare più efficientemente alle zone esterne.



Permette di semplificare notevolmente la fotografia della rete da fare. Due AS non devono comunicare tra loro direttamente le informazioni, ma devo fare routing tra AS, cioè individuare i gateway router e fare routing tra essi con un protocollo di routing tra AS.

Dunque, in particolare, dentro un AS uso un **Interior Gateway Protocol (IGP)**, ad esempio RIP (impl. DV), OSPF (impl. LS); mentre all'esterno, tra AS diversi, uso un **Exterior Gateway Protocol (EGP)**, come BGP.

I protocolli EGP spesso non tengono conto del percorso migliore, ma di altre informazioni, anche di natura politica. Faccio transitare le informazioni in una tratta che passa per un paese amico e non per un paese che può spiarmi.

OSPF

(RFC 1131, v1)

(RFC 2178, 2328, v2)

Implementa Link State, pensato per sostituire RIP. Funziona anche su reti di dimensioni notevoli.

Consente l'uso di metriche differenti. Il calcolo viene eseguito così:

Ogni nodo manda un messaggio a un nodo vicino, in attesa della risposta. In base al tempo della risposta può essere terminata la bontà o meno del link. Questo tempo tiene conto anche di quanto traffico c'è nel link, non solo di quanto è buono il link (della sua capacità quindi), ma anche del traffico che c'è nel link in quel momento.

Tra l'altro è una misura che viene fatta contemporaneamente dai due nodi, cosa che permette di bloccare sul nascere eventuali imbrogli.

Ogni nodo prepara poi una sua tabella con i costi dei suoi link verso i suoi vicini dicendo anche quali sono i suoi vicini. Questa tabella viene mandata in Flooding, si blocca in questo caso mettendo in ID all'inizio di questa tabella, in modo tale che non torna mai indietro e non fa mai circoli viziosi.

- HELLO: Scoperta e verifica dei vicini
- EXCHANGE: Sincronizzazione iniziale del DB
- FLOODING: Aggiornamento del DB

Il Flooding funziona bene se bloccato per tempo prima di congestionare la rete. La soluzione infatti sta negli LSA:

Ogni router ha un database composta da “Link State Records”.

I vari DB vengono aggiornati e sincronizzati tramite i LSA: Link State Advertisement”.

Gli LSA sono emessi:

- Quando un router riscontra un nuovo router adiacente
- Quando un router - link si guasta
- Quando il costo di un link cambia
- Periodicamente (ogni 30 min)

Qualche nodo però potrebbe non avere le info aggiornate mentre calcola. Non è un grossissimo problema se non ci sono grossi errori, perché le imprecisioni potrebbero determinare qualche salto in più.

Solo in casi eccezionali può determinare una sorta di ping pong tra nodi, in cui il messaggio gira all'infinito. Ma in tal caso comunque il pacchetto ha il suo TTL e quindi eventualmente il messaggio verrà scartato, senza creare troppi problemi.

BGP

È molto complesso e dobbiamo solo sapere cosa fa, ma non in particolare, non è propriamente oggetto del corso.

Se arrivi a occuparti di BGP vuol dire che comunque sei arrivato a un buon livello nella gerarchia degli amministratori di sistema.

È un protocollo di routing tra AS, esterno agli AS dunque, il cui scopo principale è il realizzare le dorsali di comunicazioni. Non si basa solo su info sui percorsi minimi ma su altri tipi di informazioni, che dipende ovviamente dall'admin di rete.

Altre info tanto per:

- AS Border Router (ASBR): Router connesso ad altri sistemi autonomi
- BGP Speaker: Router che supporta BGP (uno speaker non necessariamente coincide con un ASBR).
- BGP Neighbours: Coppia di Speaker che si scambiano info di instradamento Inter-AS.
 - Interni: Se appartengono allo stesso AS.
 - Esterni: Se appartengono ad AS differenti.

Lezione 15 - 13/05/2020

(lezione molto moscia dal punto di vista teorico, laboratorio + esercizi pratici, si consiglia di recuperare la lezione. Se stai leggendo questo documento nel 2021, segui la lezione dal vivo o recupera quella del 2020 con metodi occultistico-satanici).

Firewall

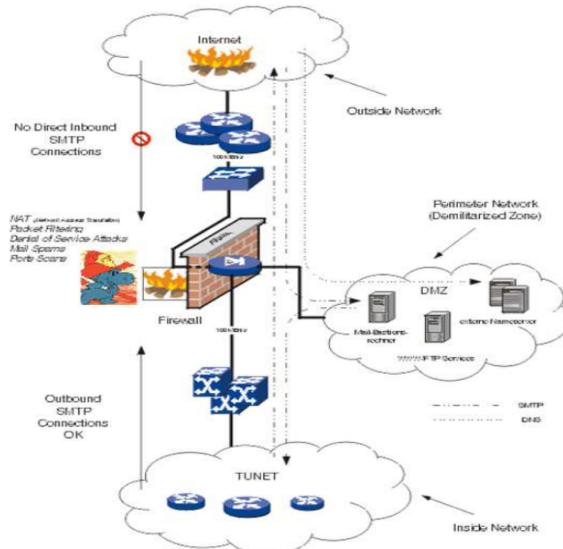
Esistono due tipi di firewall: software e hardware. A noi interessa principalmente quello hardware.

Un firewall software non è altro che la composizione di una serie di regole che vengono date ad un unico dispositivo per fare transitare o meno pacchetti dall'esterno verso l'interno e viceversa.

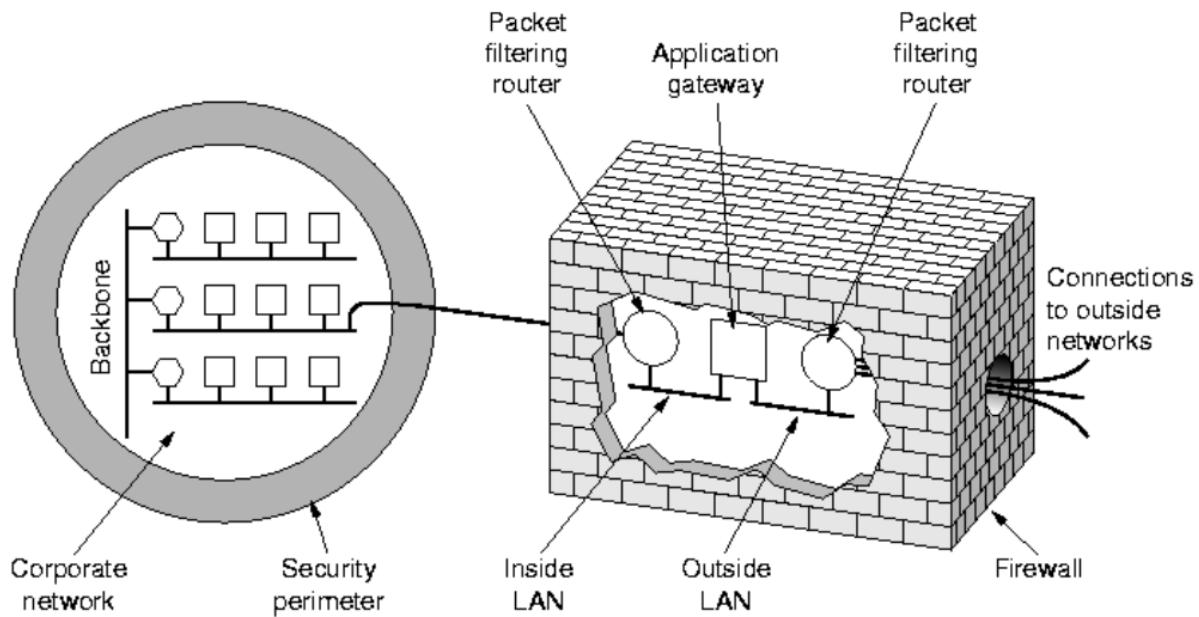
Siccome sono regole scritte in un dispositivo (che esegue un OS), se questo dispositivo è accessibile sia dal mondo esterno che dal mondo interno, quello che un attaccante potrebbe fare è trovare backdoor o bug presenti nel dispositivo, accedere al sistema, cambiare le regole di firewalling e ottenere l'accesso che veniva ritenuto bloccato.

Il router domestico ha una sorta di firewall integrato, che altro non è che delle banali regole che si possa navigare certo versi siti o possano provenire attacchi di qualche tipo dal mondo esterno. L'hardware però non è protetto dal mondo esterno: Accetta pacchetti e questi possono scalare tutta la pila protocolare fino ad arrivare al livello applicativo. Se c'è un bug o un errore o una backdoor in tutto il sistema il firewall non serve assolutamente a nulla.

Anche i vari OS hanno dei loro firewall software, che ovviamente seguono lo stesso principio. Teoricamente un applicativo malevolo potrebbe andare a cambiare le regole di firewalling (così come potrebbe essere capace di ingannare un antivirus, potrebbe anche ingannare / cambiare il firewall).



Schema Firewall Software



Firewall Hardware

Abbiamo un “qualcosa” che ha una connessione verso il mondo interno, una o più connessioni verso il mondo esterno, ma soprattutto all’interno ha un sistema hardware un po’ più complicato.

In particolare ci sono un primo router, un secondo router e un “Application Gateway”, che è un altro hardware separato dai due di ingresso e uscita.

Mentre un firewall software è un processo che gira all’interno del sistema operativo (e quindi se l’OS è compromesso anche il firewall potrebbe esserlo), in un router di questo tipo possiamo pensare di inserire un firmware limitato solo ai livelli 1,2,3.

Cioè potrebbe non avere processi che rispondono alle porte. Magari potrebbe anche essere a conoscenza dell’esistenza delle porte, poter identificare il formato dei pacchetti TCP/UDP, ma non avere nulla che risponde ai protocolli del livello di trasporto, e soprattutto al livello applicativo.

Se è semplice, è più difficile da bucare (meno codice c’è, meno errori del programmatore...).

Il compito dei router alle estremità dell’Application Gateway è molto semplice: filtrare pacchetti. Poi possiamo mettere delle regole banali in più. Separare i due dispositivi permette il fatto che se per qualche motivo si va a intaccare la sicurezza in uno dei due hardware, difficilmente si riesce a ottenere l’accesso anche all’altro dispositivo.

Quindi si vanno ad aumentare i fattori di sicurezza.

Non solo, poi vado a mettere due reti interne, le quali non sono raggiungibili né dal mondo esterno che dal mondo interno. Sono conosciute solo come reti di transito per i router di filtraggio e l'application gateway centrale.

Dato che non sono visibili da nessuna parte se non per il mondo interno al firewall, l'Application Gateway è come se non esistesse, ma lavora.

È l'equivalente di un router, però invece di lavorare a livello di pacchetti lavora al livello applicativo.

Quindi, nell'Application Gateway, possiamo pensare di inserire regole molto più complesse, anche un sistema operativo più complesso, che guarda il contenuto dei pacchetti.

Ad esempio, i router di filtraggio riescono a filtrare email di spam attingendo da blacklist, mentre una volta dentro l'Application Gateway potremmo vietare il passaggio a qualsiasi cosa contenga eseguibili: in qualche modo ho identificato che il pacchetto conteneva un eseguibile, oppure se contiene file zippati devono essere accessibili senza password. Quindi nell' AG possiamo mettere regole di filtraggio sui contenuti. È molto più efficiente dal punto di vista del firewalling: molto più resistente agli attacchi perché le LAN del firewall sono invisibili al mondo esterno.

Il problema del firewall hardware è che costa molto, molto di più di un firewall software.

DMZ

“Zona Demilitarizzata”. Nella realtà è una zona al confine tra due stati in rapporti tesi. Cioè una zona cuscinetto dove non ci sono né militari da una parte né dall'altra. Cioè una zona neutra. Nessuna delle due parti ha controllo politico.

Dal punto di vista informatico è una zona della rete non protetta dal firewall, ma appartiene ancora alla rete amministrata dall'admin interno.

Nei router domestici viene stabilita per poter mettere su un server web nel pc privato dell'utente senza aprire porte.

Nei sistemi più raffinati è possibile stabilire una porta a cui agganciare una rete DMZ dove attaccare fisicamente le macchine esposte ad internet. Non delegano più la sicurezza al firewall del router ma devono avere le loro regole.

Data Link Layer (4½ lezioni)

Lezione 16 - 15/05/2020

Si occupa di fornire al livello di Rete un servizio di trasmissione di flussi di bit. I suoi compiti principali sono:

- **Framing:** Raggruppare i bit provenienti dal livello fisico in modo da formare pacchetti;
- **MAC Sublayer:** gestire l'accesso al mezzo fisico, nel caso di un link broadcast;
- Fornisce un recapito affidabile, se richiesto;
- Gestire gli errori dovuti al canale di trasmissione;
- Regolare il flusso dei dati tra sorgente e destinazione.

È fondamentale perché ovviamente è quello che permette lo scambio reale di informazioni tra macchine adiacenti.

Il framing spreca tantissima larghezza di banda, non è un problema semplice da risolvere.

DLL in alcune tipologie di reti è diviso in due parti ben distinte, il **Medium Access Control Sublayer**, e lo strato software vero e proprio (Logical Link Control, ndr) che (uniti sono, ndr) è il DLL. Il MAC Sublayer si occupa principalmente di risolvere il problema di capire quale nodo sta parlando, quando inizia e quando finisce di parlare in un canale broadcast.

Nelle connessioni punto-punto è un elemento molto semplificato se non assente.

Poi c'è un problema di recapito affidabile. È molto richiesto in mezzi di comunicazione poco affidabile come wireless (che il corso non tratta).

Poi vi è la gestione degli errori, che avvengono molto spesso dato che il mezzo di comunicazione non è perfetto. Banalmente gli errori sono bit 1 che diventano 0 o viceversa.

Si possono rilevare (semplice), la mancanza di errori è probabile, ma non possiamo esserne certi perché potrebbero esserci dei falsi positivi, mentre la correzione è ancora più "inaffidabile" rispetto alla rilevazione.

Il Livello DLL si occupa principalmente della comunicazione tra apparati che sono contigui: collegati da uno stesso mezzo fisico di comunicazione.

Oltre al link fisico di comunicazione, serve anche una scheda che traduce i segnali elettrici in bit, esegue tutte le operazioni previste dal livello DLL e passa i dati al sistema software del PC. Molto spesso tutto quello che fa il DLL è implementato nel firmware della scheda di rete. Il firmware ovviamente può essere aggiornato o modificato, ma non sempre è necessario.

Framing

Facciamo un “saltino” in giù al livello fisico.

In una fibra ottica, in cui “in qualche modo” passa la luce, possiamo immaginare di avere due livelli: c’è la luce o non c’è la luce.



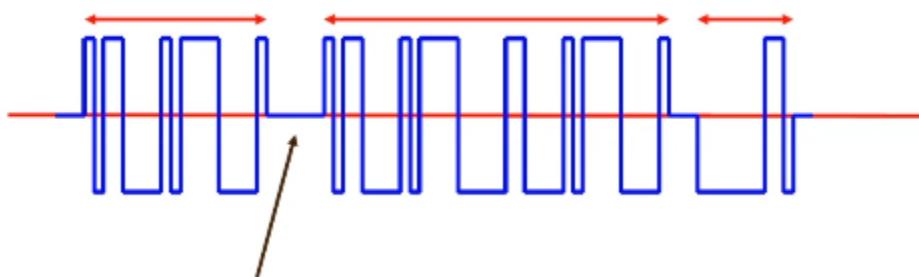
Immaginiamo di avere una frequenza di questo tipo, a occhio si potrebbe dire che il primo pezzo sembra una frame, perché c’è una sorta di comunicazione, la luce è presente in certi istanti di tempo e assente in altri. Poi c’è una lunga sequenza di luce assente, e quindi di 0.

Il problema è che nella zona evidenziata dalla freccia non sappiamo se c’è un’assenza di comunicazione o una sequenza molto lunga di bit 0. È qualcosa che non possiamo sapere, il livello fisico non ce lo dice. A livello DLL sappiamo che c’è un’assenza di segnale che il sistema ha interpretato come una sequenza di 0, che non è la stessa cosa.

Non è un problema da poco perché rischiamo di non capire nulla in fase di comunicazione.

I sistemi sincroni non sono una buona idea perché se si perde la sincronia si hanno conseguenze catastrofiche. Vedremo le cose che si usano realmente nei sistemi di comunicazioni.

Un sistema banale è di avere tre livelli:



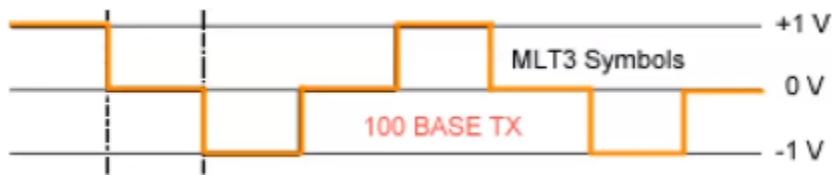
Usiamo uno dei tre livelli per segnalare un’assenza di comunicazione.

Inoltre, potremmo trasportare un bit “più qualcosa”, un “bit virgola qualcosa”.

In pratica noi stiamo sprecando un livello che potremmo invece usare per trasportare più comunicazione, quello che chiamiamo “ridondanza” nella comunicazione.

Ci sono altre tecniche che permettono di limitare la ridondanza.

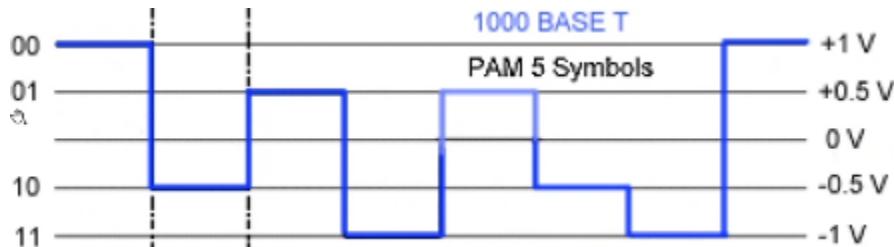
Ad esempio, usando tre livelli, non si associa un bit a un livello ben preciso.



In questo schema, se dopo un bit c’è un bit 1 cambio di livello, se ci sono bit 0 mantengo lo stesso livello. In questa maniera c’è un segnale che sale e scende in continuazione, ed è una cosa *simpatica* perché, come vedremo a livello fisico, limita le alte frequenze.

Non c’è quindi un livello che viene utilizzato esclusivamente per segnalare l’assenza di comunicazione.

Si può fare anche con 5 livelli:



Associo il livello 0V all’assenza di segnale, mentre ad ogni livello utilizzabile (4) associo non un bit, ma una coppia di bit. Ovviamente il sistema deve essere in grado di distinguere questi 5 livelli.

Codifica 4B5B

Mettiamo uno strato software per capire quando termina una sequenza di trasmissione. A livello fisico abbiamo sequenze di 5 bit, ma a livello logico abbiamo sequenze di 4 bit, codificate secondo una tabella prestabilita.

Ad esempio, se vogliamo trasmettere il valore 7, codificato con 0111, la tabella mi dice che lo devo trasmettere come 0111**1**, in pratica ogni 4 bit introduco una ridondanza di un bit.

Un altro esempio, per trasmettere 0, ossia 0000, la sequenza di zeri potrebbe ripetersi per parecchio tempo. Convertito in 5B, abbiamo 11110, cioè non ci sono mai sequenze dove ci sono bit tutte dello stesso tipo, cioè non avremo mai sequenze ininterrotte di 1 o 0 consecutivi.

In questa codifica, non avremo mai più di 3 “zero” consecutivi, o 8 “uno” consecutivi. Quindi, se abbiamo in caso di molti zeri consecutivi, vuol dire che non c’è comunicazione.

La mappatura non è casuale, ma secondo delle regole per evitare sequenze ripetute appunto. Rimangono però 6 simboli a disposizione, che possiamo utilizzare per mandare info di controllo, ad esempio start di frame o fine frame.

Dunque, con 4B5B usiamo un sistema di comunicazione a due livelli risolvendo il problema del framing. Sprechiamo però $\frac{1}{5}$ della banda.

Supponiamo di voler trasmettere a 100 Mbps, aggiungendo un bit sul cavo andremo a fare una comunicazione a 125 Mbps, solo che questa ridondanza in più è sprecata per risolvere il problema del framing, cioè il 25 % della banda. (*ndr: insomma, il 25% di 125 è 31.25, c’è un po’ di confusione col discorso dell’aggiungere banda e quanta è sprecata, per la percentuale si rifà ai 100 Mbps iniziali*).

Però in una codifica a 3 livelli avremmo sprecato il 50 % della banda, anche se sono sistemi su livelli diversi, 4B5B è DLL, codifica a 3 livelli è fisico.

Oltre a convertire però (che è tutto quello che fa questo sistema), potremmo voler raggiungere un obiettivo secondario, cioè dare una sincronia nel clock tra mittente e destinatario. Misurare una variazione alto-basso è facile dal punto di vista elettrico, misurare la “Lunghezza” di un tratto tra una variazione e l’altra è più difficile. Si sovrappone un segnale di clock, ma se non è perfettamente sincronizzato con la frequenza dati potremmo fare la misura esattamente durante la variazione, interpretando male i dati. Per tenerlo sincronizzato posso usare proprio le variazioni. Questa è proprio una delle politiche che ha determinato la tabella.

Codifica 8B10B

Codifica simile a 4B5B, elettricamente neutra (= il numero di bit 1 viene mantenuto il più possibile uguale al numero di bit 0 trasmessi). Usata in vari standard quali SATA, USB 3.0, HDMI, alcune versioni di Gigabit Ethernet, etc.

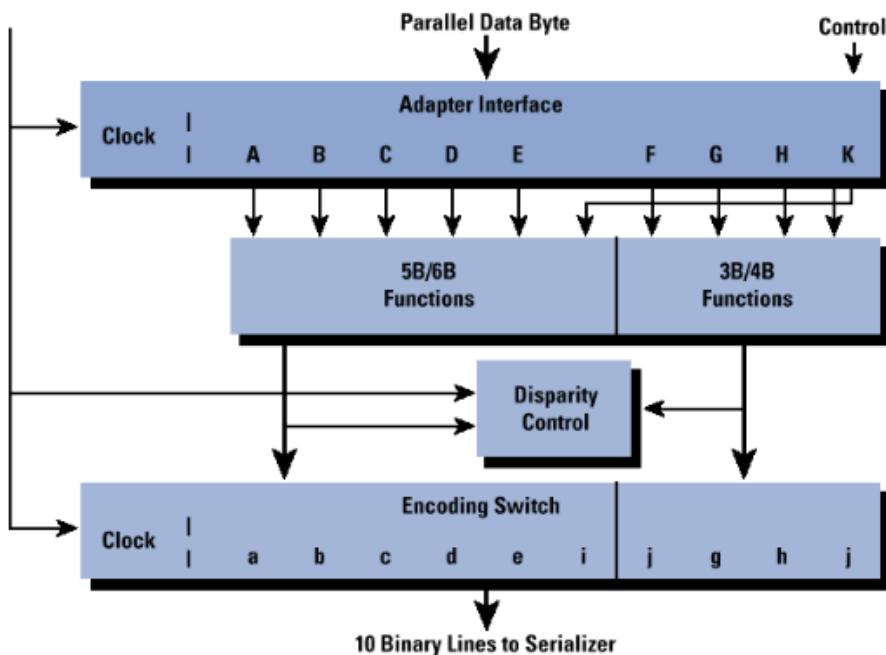
Usa 8 bit dati con 10 bit di segnale. Non cambia lo spreco di ridondanza.

Perché è importante che la codifica sia elettricamente neutra?

Se il sistema non è elettricamente neutro, ci sarà un trasferimento di energia tra un dispositivo all'altro. Possiamo dire che da un lato c'è un trasferimento di energia tra un dispositivo e l'altro, mentre dall'altro c'è, *"in un certo senso, l'energia che va in senso contrario"*.

Se riuscissimo ad equilibrare bit 1 e 0 il trasferimento sarebbe nullo. È importante per i dispositivi che usano 8B10B, infatti.

L'obiettivo di questa codifica è di avere, non solo nell'ambito di una sequenza di 8 bit, ma due sequenze, cioè 8+8 o 10+10 una volta trasformati, un sistema dove il numero di bit è sostanzialmente uguale, a meno di 2 bit.



Partiamo da 8 bit, suddivisi in un primo gruppo di 5b e un secondo gruppo di 3b, trasformati rispettivamente in 6 e 4 bit. Queste sequenze non sono fisse, in particolare la seconda, dipendono dal risultato dell'operazione precedente.

Seguono alcuni esempi di conversioni:

3b	3b	4b
Decimale	Binario (HGF)	Binario(fghi)
0	000	0100 oppure 1011
1	001	1001
2	010	0101
3	011	0011 oppure 1100
4	100	0010 oppure 1101
5	101	1010
6	110	0110
7	111	0001 o 1110 o 1000 o 0111

Notiamo che alcune codifiche sono differenti, in particolare si mette un eccesso di bit 1 o 0. L'idea è che se al passo precedente avevo un eccesso di bit 1, allora al successivo metterò un eccesso di bit 0, e viceversa. Se era già uniforme, creo uno squilibrio e cerco di riequilibrare dopo. Le sequenze che non hanno alternative ne sono sprovviste perché sono già equilibrate dentro la singola sequenza. (Ad esempio, 5 -> 101 -> 1010).

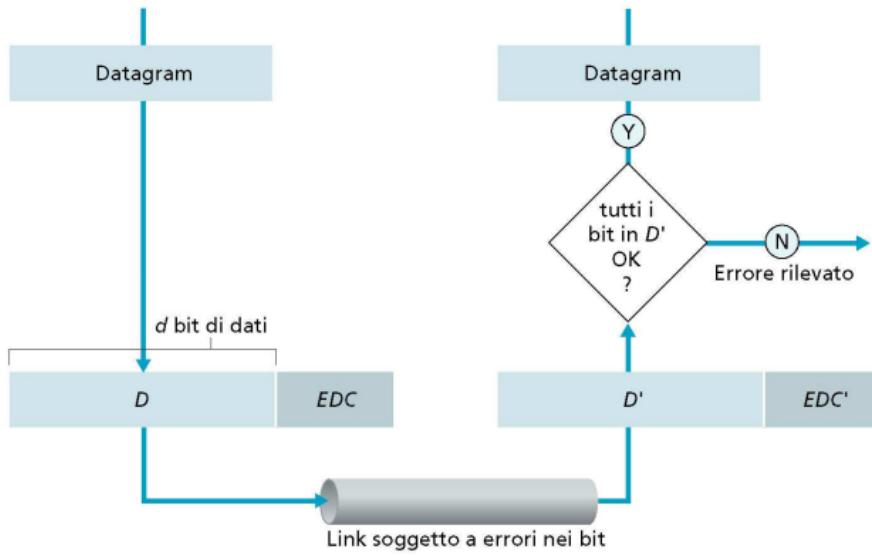
Ritornando alla figura precedente, Arrivano in input il byte, il valore di controllo che proviene dalla codifica precedente, che mi permette di stabilire quale coppia per 5B/6B e capire se serve aggiustare qualcosa nella parte rimanente 3B/4B. Se c'è uno squilibrio lo riporto alla successiva conversione.

Rilevazione e/o Correzione degli errori

La Rilevazione consente di individuare la presenza di errori nella trasmissione in una frame, ma non di correggerlo, cioè non sappiamo qual è, o dov'è l'errore, ma bensì indicare che è presente un errore dentro la frame. Il livello DLL quindi butta tutto a causa della presenza di uno o più errori.

La Correzione consente di correggere l'errore oltre che rilevarlo (anche se con forti limitazioni). Cioè individua esattamente dov'è l'errore e lo corregge.

Entrambi i metodi sono basati sulla presenza di ridondanza nella comunicazione. Hanno una base statistica, e come tale la probabilità di falsi positivi non è nulla. Individuare l'errore è facile e certo, la correttezza si basa su quanto è complicato andare ad avere una sequenza di cambiamenti nella frame, cambiamenti che possono essere casuali o mirati. Per avere maggiore fiducia sul controllo devo usare un algoritmo il più affidabile possibile.



Abbiamo quindi una sequenza di dati D formata da un numero d di bit. Aggiungiamo a questa sequenza un'altra sequenza di bit calcolata in funzione di questa sequenza di dati.

Il tutto viene trasmesso e al lato del destinatario abbiamo una sequenza D' e sequenza di controllo EDC' (potrebbe essere capitato un errore anche lì).

Si riutilizza la funzione per calcolare la ridondanza e facciamo il check con input D' . Se il risultato è uguale a EDC' , allora la prendiamo per buona. Se è sbagliato sicuro c'è un errore.

Controllo di Parità

Il modo più semplice è il controllo di parità, a 1 bit. Consiste nell'aggiungere un bit di parità a fine sequenza, in modo da far risultare il numero di bit “uno” pari (per esempio):

01101110 diventa
011011101

Più è lunga la sequenza di dati meno è probabile che riusciamo a beccare l'errore, perché se ci fossero due errori il risultato sarebbe ancora corretto. Questo sistema individua un numero di errori dispari. Più aumenta la lunghezza più aumenta la probabilità che ci sia un numero pari di errori.

Parità a due dimensioni

Parità di riga				
Parità di colonna	$d_{1,1}$	\dots	$d_{1,j}$	$d_{1,j+1}$
	$d_{2,1}$	\dots	$d_{2,j}$	$d_{2,j+1}$
	\dots	\dots	\dots	\dots
	$d_{i,1}$	\dots	$d_{i,j}$	$d_{i,j+1}$
	$d_{i+1,1}$	\dots	$d_{i+1,j}$	$d_{i+1,j+1}$

Nessun errore

1	0	1	0	1	1
1	1	1	1	0	0
0	1	1	1	0	1
<hr/>					0
0	0	1	0	1	0

Errore correggibile
del singolo bit

1	0	1	0	1	1
1	0	1	1	0	0
0	1	1	1	0	1
<hr/>					0
0	0	1	0	1	0

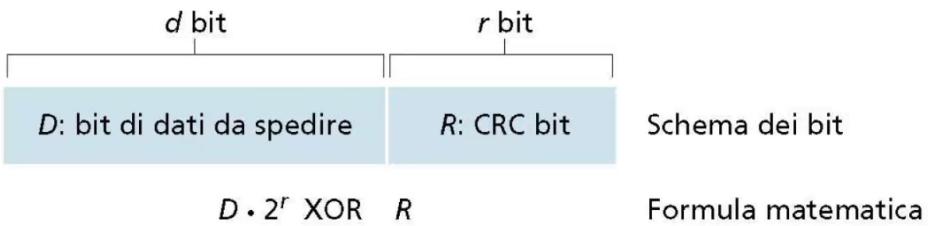
↓
Errore
di parità

→
Errore
di parità

Prendiamo una sequenza di bit e la scriviamo riga per riga, possiamo fare parità per riga e per colonna.

In questo modo, se c'è un errore contemporaneamente in una riga e in una colonna, allora ho centrato la posizione dell'errore. Però funziona solo se c'è un errore e uno soltanto nella sequenza di bit. Se ce ne dovesse essere più di uno il sistema fallirebbe.

CRC



$$\begin{array}{ccccccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ x^7+ & & x^4+ & & x^2+ & x^1+ & 1 \end{array}$$

Codice di Ridondanza Ciclica, una tecnica per la [rilevazione](#) degli errori molto più affidabile ed efficiente. Si usa dappertutto (anche hard-disk e carte magnetiche). È semplice da implementare e il circuito che realizza i calcoli costa poco.

Premessa Matematica

L'idea è cercare una sequenza di operazioni matematiche per cui il risultato R viene univocamente e correttamente generato dai dati D .

Trasformiamo il Byte 10010111 in un polinomio, dove la posizione del singolo bit viene usata come esponente del monomio, se c'è 1. Se c'è 0 non scriviamo il monomio. In altre parole il bit stesso diventa coefficiente del monomio, mentre la posizione diventa l'esponente a cui viene elevato. (x^0 o 1 è la stessa cosa ovviamente)

10010111 infatti diventa il polinomio $x^7 + x^4 + x^2 + x^1 + x^0$

Prendiamo un altro polinomio arbitrariamente:

01011011 -> $\quad \quad \quad +x^6 + x^4 + x^3 + x^1 + x^0$

Possiamo fare la somma e il prodotto, che risultano rispettivamente:

Somma: $x^7 + x^6 + x^3 + x^2$

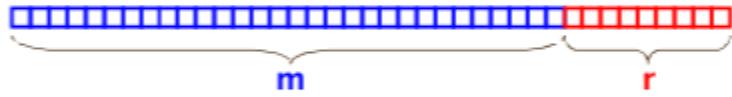
Prodotto: $x^{13} + x^{11} + x^8 + x^7 + x^5 + x^4 + x^0$

Possiamo farlo dal punto di vista matematico perché i coefficienti di questo polinomio sono nel campo \mathbb{Z}_2 , che è un campo dove esistono solo 0 e 1, la somma rispecchia la tabella di verità di XOR, e il prodotto è l'operazione AND.

Il fatto che sia un campo ci assicura che ci siano le proprietà commutativa, associativa, etc...

$M(x)$ polinomio con i dati

$G(x)$ polinomio generatore di grado r .



$$R(x) = x^r M(x) \bmod G(x)$$

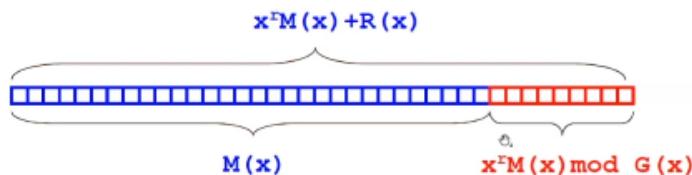
Parliamo ora di sequenze di bit: prendo m bit, e aggiungo la ridondanza di r bit.
Genero i polinomi $M(x)$ da m , e $G(x)$, un polinomio particolare di grado massimo r .
Ad esempio, in CRC-16, r è di grado 16 e scrivo $x^{16}+x^{15}+x^2+1$. (è uno standard)

Prendo i bit m di dati, aggiungo r bit di zeri, quindi ottengo una sequenza di $m+r$ bit. La divido per $G(x)$ prenendo il modulo di questa operazione.

Esempio: partiamo da 17, che vuol dire aggiungere degli 0 in un sistema decimale?
Moltiplicarlo per 10, cioè ottenere 170,1700,etc... Se fosse $r=2$, avremmo $10^2 \cdot 17$, cioè 1700.

Quindi, $x^r M(x)$ è il polinomio che viene fuori da m aggiungendo r zeri alla fine.
L'idea del CRC è di avere dei dati, seguita da una sequenza da calcolare, che viene fuori da $R(x)$, tale che tutta la sequenza $m+r$, dove su r c'è il modulo e non più zeri, risulta divisibile perfettamente per $G(x)$.

Quindi l'operazione che devo fare a destinazione è banalmente:



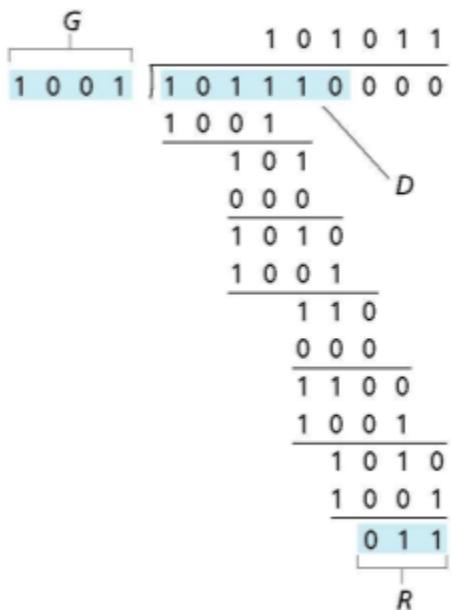
Prendere tutto il polinomio, dati e ridondanza, e fare la divisione per $G(x)$, se la divisione mi dà resto diverso da 0 c'è stato un errore, butto tutto via (rilevo e basta, non correggo!!).

Riusciamo a individuare una certa sequenza di errori (più lunga del controllo di parità ovviamente, e indipendentemente dal fatto che siano pari / dispari).

Dovremmo “sottrarre” i dati dalla sequenza in teoria, per far risultare il polinomio perfettamente divisibile per $G(x)$, ma siccome siamo nel campo \mathbb{Z}^2 , ci troviamo nel caso particolare in cui il risultato della somma e della sottrazione sono uguali, e ovviamente è più semplice aggiungere la ridondanza alla fine piuttosto che fare altro.

Calcolo del CRC

Come si fa la divisione? (CONVIENE DAVVERO FARLA SU CARTA):



Prendo i primi quattro, prendo il divisore e faccio la sottrazione, poi abbasso le ulteriori cifre.

A questo punto ho due possibilità, o metto 0 nel risultato della divisione (abbassare un'altra cifra), o metto 1 (ricopio il divisore e faccio un'altra sottrazione).

La sequenza R che ottengo la aggiungo a D e ottengo un qualcosa che è perfettamente divisibile da parte di $G(x)$.

Rifacciamo la stessa cosa ma in maniera più visibile, prendiamo un'altra sequenza e un altro polinomio generatore. Dato che il polinomio generatore è fatto da 5 bit, ne esamino 5 alla volta facendo XOR se il primo bit è 1, e poi shift, nessuna operazione e shift se il primo bit è 0.

```
11010110111110
10011
01001110111110
10011
00000010111110
00000
00000010111110
00000
0000000101011110
00000
0000000010111110
00000
0000000001011110
00000
0000000000101110
00000
0000000000010110
00000
0000000000001011
10011
```

Se alla fine il risultato è tutti 0, allora non ci sono errori. È un semplice registro a scorrimento, in pratica. Questa è l'implementazione di tutta la teoria matematica. Inoltre è interessante notare che ci interessa per la dimensione del registro a scorrimento è la dimensione del polinomio generatore, non la dimensione dell'intera frame.

Lezione 17 - 18/05/2020

Recap

Rilevazione e Correzione: c'è differenza tra le due, la rilevazione permette solo di rilevare qual è l'errore, ma non può capire dov'è o correggerlo, mentre la correzione è più complicata e richiede più ridondanza per sistemare l'errore. Permette (con forti limitazioni) di sistemare l'errore.

Inoltre, mentre la rilevazione è certa, la non-rilevazione dell'errore non significa che la frame sia corretta. Stessa cosa si può dire sulla correzione, perché applica un procedimento e non è detto che corregga l'errore.

Distanza di Hamming

Teoria alla base della correzione degli errori prima di procedere (**DEI POVERI STUDENTI SONO STATI BOCCIATI PER NON AVER SAPUTO RISPONDERE A QUESTA DOMANDA**):

Innanzitutto parliamo di distanze tra due codeword andando a capire di quanto differiscono due codeword, in questo caso due Byte.

10001100 XOR
11000100 =
01001000 distanza = 2

Poniamo la distanza a 2 perché per passare da una codeword all'altra dobbiamo cambiare due bit. Quindi la distanza è quante variazioni sono richieste per poter passare da una all'altra.

Possiamo parlare anche di vocabolario: un insieme di codeword ben definite.

Consideriamo codeword da 6 Bit:



Avremo 64 combinazioni ma ne consideriamo 4. Tutte le altre non fanno parte del nostro vocabolario. Se queste codeword stabiliamo tutte le possibili distanze, che saranno 3 e 6 tra loro, come si può vedere nel rettangolo. Su questo vocabolario diremo

che è composto da 4 codeword, e la distanza è il minimo di questi valori possibili (3 in questo caso). Quindi è un vocabolario di 4 codeword a distanza 3.

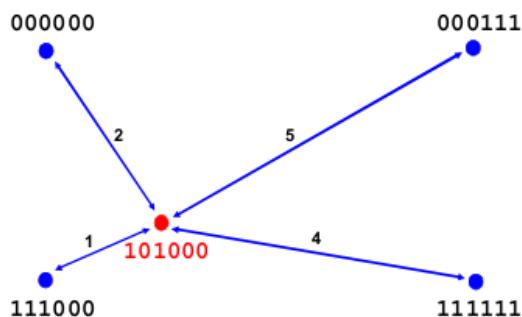
Perché tutto questo?

L'idea è che il trasmettente può generare solo una di queste quattro codeword, non ne può generare differenti. Il problema è che il canale potrebbe inserire degli errori.

Facciamo un'ipotesi molto forte: La probabilità che ci sia un singolo errore sia molto più grande della probabilità che ce ne siano due, contemporanee, sulla stessa codeword, e lo scriviamo:

$$P(e=1) \gg P(e=2) \gg P(e=3) \gg \dots \gg P(e=6).$$

Molto maggiore vuol dire due-tre ordini di grandezza di differenza. Se poi la probabilità di avere un solo errore è piccola di suo, allora quella di averne due è sostanzialmente trascurabile. Si tratta comunque di situazioni particolari, in altre situazioni si ha addirittura $P(e=1) = P(e=2)$, ad esempio nei canali wireless.



Supponiamo quindi che il canale abbia introdotto un errore, quindi è arrivata a destinazione una codeword errata. In questo caso la rilevazione è banale, non serve neanche il CRC, d'altro canto la ridondanza introdotta è fortissima, abbiamo introdotto 4 bit di ridondanza (per generare quelle 4 codeword bastano 2 bit).

Possiamo calcolare poi qual è la distanza dalle codeword legali. Nell'ipotesi che ci sia stato un solo errore e uno soltanto è ovvio (con forte probabilità) che la parola originale che ha generato questa codeword arrivata a destinazione con un errore è quella a distanza minore. Possiamo dire ciò su base probabilistica supponendo che la probabilità di avere due errori sulla stessa sequenza sia bassissima, ma nessuno ci garantisce che sia vero.

Correzione di errori

A questo punto cosa possiamo dire?

In un vocabolario del tipo 4 codeword distanza 3, presa una codeword errata, nel caso peggiore sarà a distanza 1 da una codeword lecita e a distanza 2 da un'altra lecita.

È il caso peggiore perché potrebbe invece capitare che è a distanza 5 da quasi tutte quante, e inoltre nel nostro vocabolario non può esistere una codeword errata che è contemporaneamente a distanza 1 tra due codeword lecite, proprio per come è stato impostato per ipotesi il vocabolario. Deve essere almeno a distanza 1 e distanza 2.

Se questa cosa è vera lo è perché il vocabolario è a distanza 3.

In sostanza, se abbiamo un vocabolario a **distanza d=3** possiamo correggere gli errori singoli (un bit errato a codeword).

Generalizzando questa affermazione possiamo dire che se vogliamo **correggere** un numero più grande di errori, allora abbiamo bisogno di un vocabolario di **distanza d=2e+1**.

Per rilevare invece una quantità **e** di errori è necessario un vocabolario con **distanza d=e+1**.

La correzione si effettua esclusivamente su base probabilistica.

A questo punto abbiamo scoperto alcune cose: Per poter procedere alla **correzione** degli errori **singoli** ci serve:

- L'ipotesi di base sulla probabilità: $P(e=1) >> P(e=2) >> \dots$
- Definire un vocabolario, diminuire il numero di simboli leciti
- Definire questo vocabolario con distanza minima $d=3$

Esempio: codewords da 10 bit:

(2 dati, 8 ridondanza)

0000000000

0000011111

1111100000

1111111111

Parole valide: 4

Distanza: 5

Correzione errori: 2 bit

Rilevazione errori: 4 bit

È possibile selezionare un altro vocabolario con parole da 10 bit ma con più parole, tale che la distanza rimane 5?

Sì:

A 0000000000
B 0000011111
C 0011100011
D 0011111100
E 1100100101
F 1100111010
G 1111000110
H 1111011001

	A	B	C	D	E	F	G	H
A	0	5	5	6	5	6	6	7
B	5	0	6	5	6	5	7	6
C	5	6	0	5	6	7	5	6
D	6	5	5	0	7	6	6	5
E	5	6	6	7	0	5	5	6
F	6	5	7	6	5	0	6	5
G	6	7	5	6	5	6	0	5
H	7	6	6	5	6	5	5	0

Parole valide: 8

Distanza: 5

Bit di dati: 3

Bit di controllo: 7

Combinazioni possibili: 2^{10}

Abbiamo 8 codeword, sulla matrice sono riportate le distanze tra una codeword e l'altra, sulla diagonale infatti abbiamo tutti 0 perché si confrontano le stesse codeword.

Possiamo fare ancora di meglio, e magari automaticamente?

Supponendo di avere 3 bit di dati, quanta ridondanza dobbiamo aggiungere per poter correggere su base probabilistica? E dopo che abbiamo trovato la formula, come costruiamo un codice per trovare un vocabolario che ci soddisfa?

RIDONDANZA

DEI POVERI STUDENTI SONO STATI BOCCIATI PER NON AVER SAPUTO RISPONDERE A QUESTA DOMANDA. SERIAMENTE, È IMPORTANISSIMA.

Ricapitolando dal paragrafo precedente, le domande a cui vogliamo rispondere sono:

Quanta ridondanza serve? Dati m bit di dati, quanti bit di ridondanza r servono?

Come costruire un codice per correggere errori singoli? Partendo da m bit di dati, sapendo come calcolare r , come costruire operativamente il vocabolario?

Supponiamo di avere un vocabolario con m bit dati e r bit controllo/ridondanza.

Quindi ci saranno $n=m+r$ bit per ogni codeword.

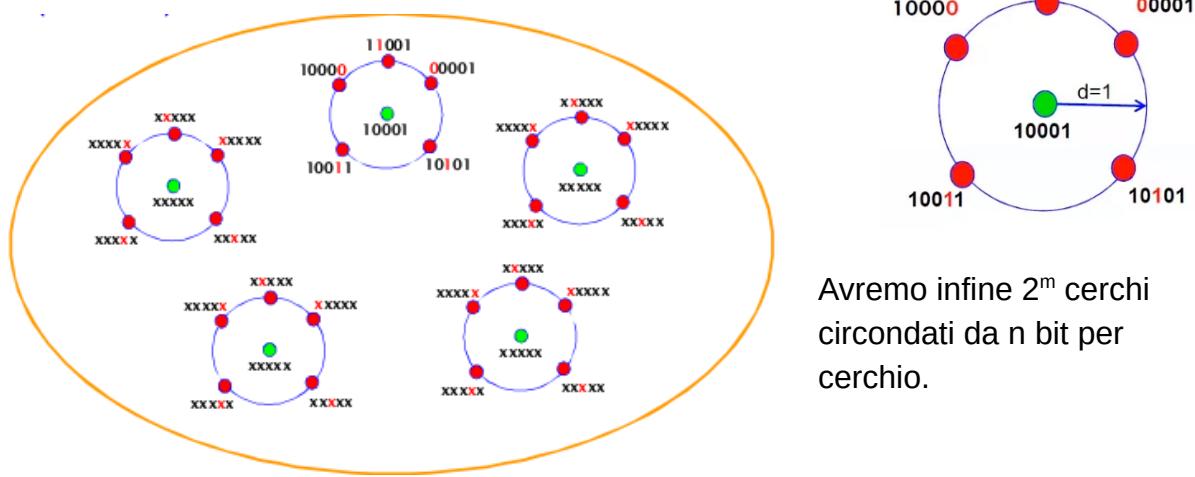
Le combinazioni possibili con n bit sono 2^n , di cui solo 2^m sono valide.

Consideriamo una codeword valida: 10001. Tutte le codeword a distanza 1 sono sicuramente errate.

Cambiando un bit alla volta possiamo scrivere altre n codeword, tutte sicuramente errate e a distanza uguale.

Possiamo metterle su un cerchio essendo a distanza uguale:

Estendiamo questo ragionamento a tutte le codeword valide:

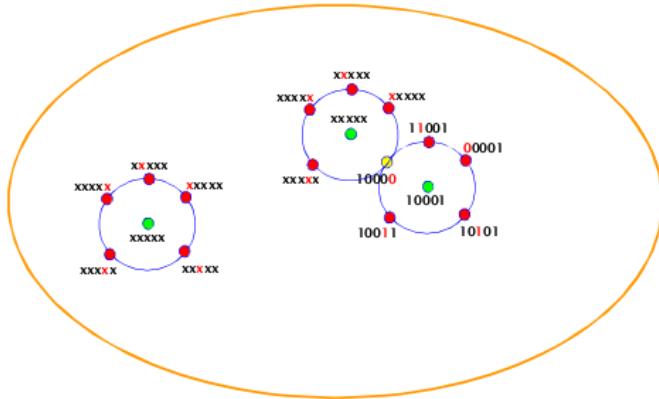


Avremo infine 2^m cerchi circondati da n bit per cerchio.

Quindi: 2^m sono tutti i possibili cerchi, $(n+1)$ sono le codeword presenti in un singolo cerchio, il totale deve essere minore o uguale a 2^n

Per $d=3$, risulta $(n+1)2^m \leq 2^n$

Però appunto non ci devono essere, come detto prima, codeword che sono a distanza 1 contemporaneamente tra due altre codeword, altrimenti non funzionerebbe questo discorso. Essere a distanza 3 preclude la possibilità che ciò accada



Quello che non vogliamo che succeda

In parole semplici, questi cerchietti sono tutti separati perché stiamo assumendo che la distanza minima del vocabolario è 3. Se abbiamo dimostrato ciò, allora l'unione di questi cerchietti è pari alla somma.

Allora, avendo $d=3$, possiamo scrivere con certezza

$$(n+1)2^m \leq 2^n$$

Semplificandoabbiamo:

$$(m+r+1)2^m \leq 2^{m+r}$$

$$(m+r+1) \leq 2^r$$

$$m+1 \leq 2^r - r$$

L'espressione risultante è ovviamente valida solo per $d=3$.

Esempi:

$$m=8 \rightarrow r=4,$$

$$(8+4+1) \leq 2^4$$

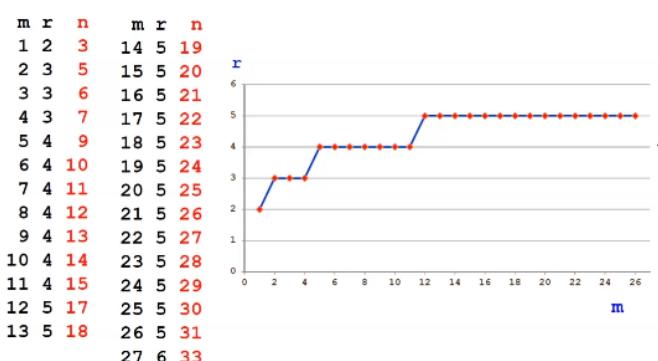
$$13 \leq 16$$

$$m=11 \rightarrow r=4,$$

$$(11+4+1) \leq 2^4$$

$$16 = 16$$

Possiamo diagrammare i valori di m ed r



Mancano in n tutte le potenze di 2, questo perché al passaggio tra 15 e 17 c'è l'incremento sia della m che della r di 1. Quindi i valori che sono potenze di 2 vengono saltati perché proprio in quei valori entrambi incrementano n di 2. Cioè quando arriviamo al valore in cui l'espressione soddisfa il risultato "uguale" invece del "minore stretto" allora incrementa anche il valore r . Ci interessa per il prossimo paragrafo.

Codici di Hamming

Abbiamo detto che nella n mancano le potenze del 2.

Prendiamo una codeword dove, numerando le posizioni dei bit in ordine crescente da sinistra verso destra, mettiamo in tutte le posizioni che sono potenze di 2 dei bit di controllo, nelle altre posizioni ci sono invece bit di dati. Con questo sistema mano a mano che aumentiamo i bit di dati, spunteranno automaticamente i bit di controllo.

Esempio:

00110010000

I bit in rosso sono di controllo.

Possiamo anche dire che tutti i bit di dati hanno di fatto una posizione data da una combinazione delle potenze del due.

Quello che facciamo è la seguente, prendiamo i dati originali, e li posizioniamo tutti in posizioni che non sono potenze del due, e nelle potenze del due mettiamo i codici di controllo, **che dobbiamo ancora trovare**. Li troviamo con le seguenti espressioni:

(Supponiamo una stringa originale 10010001100):

$\begin{array}{ccccccccc} \text{x} & \text{x} & \text{1} & \text{x} & \text{001} & \text{x} & \text{00001100} \\ \uparrow & \uparrow & \uparrow & & & \uparrow & \\ 1 & 2 & 4 & & & 8 & \end{array}$	$3 = 1+2$
$b_1 = 3 \otimes 5 \otimes 7 \otimes 9 \otimes 11 \otimes 13 \otimes 15$	$5 = 1 + 4$
$b_2 = 3 \otimes 6 \otimes 7 \otimes 10 \otimes 11 \otimes 14 \otimes 15$	$6 = 2+4$
$b_4 = 5 \otimes 6 \otimes 7 \otimes 12 \otimes 13 \otimes 14 \otimes 15$	$7 = 1+2+4$
$b_8 = 9 \otimes 10 \otimes 11 \otimes 12 \otimes 13 \otimes 14 \otimes 15$	$9 = 1 + 8$
 	$10 = 2 + 8$
$b_1 = 1 \otimes 0 \otimes 1 \otimes 0 \otimes 0 \otimes 1 \otimes 0 = 1$	$11 = 1+2 + 8$
$b_2 = 1 \otimes 0 \otimes 1 \otimes 0 \otimes 0 \otimes 0 \otimes 0 = 0$	$12 = 4+8$
$b_4 = 0 \otimes 0 \otimes 1 \otimes 1 \otimes 1 \otimes 0 \otimes 0 = 1$	$13 = 1 + 4+8$
$b_8 = 0 \otimes 0 \otimes 0 \otimes 1 \otimes 1 \otimes 0 \otimes 0 = 0$	$14 = 2+4+8$
 	$15 = 1+2+4+8$
 101100100001100	

Il bit in posizione uno quindi è pari al bit in posizione 3 XOR bit in posizione 5, etc....

Perché? Perché ogni bit in una data posizione si può scrivere come somma di due posizioni che sono potenze del due, come nella figura a destra. Infatti tutti i bit con cui facciamo lo XOR contengono 1. Da notare come, ad esempio, facciamo 3 XOR 5, e il 6 non è incluso perché non contiene 1 nella somma che indica la sua posizione.

Quindi, dalla figura a destra possiamo ricavare il modo per calcolare i bit di controllo, che sono semplicemente XOR in serie uno dopo l'altro.

La sequenza risultante sarà:

101100100001100

Inseriamo ora un errore in un bit dati qualsiasi:

101100100101100

Adesso il destinatario esegue lo stesso calcolo con gli XOR di prima, solo che risulterà in valore diverso rispetto ai bit di controllo (in questo caso sono tutti 1), quindi riesce a individuare che c'è un problema.

In questo caso, i bit in posizione 2 e 8 risultanti dagli XOR non coincidono con i bit nella sequenza che è arrivata. Nella sequenza valgono 0 e 0, il destinatario invece rileva che sono 1 e 1.

Qual è tra le righe che indicano la posizione di un bit come somma di potenze di due l'unica per cui vengono influenzati solo i bit 2 e 8? La riga che indica il 10, ossia $10 = 2+8$. Per cui il bit 10 è quello errato.

Questo di fatto è un'implementazione dell'osservazione ricavata prima sulla ridondanza.

--

Un piccolo trucchetto aggiuntivo: se gli errori fosse concentrati in una sequenza?

**xx1x110xx0x011xx0x110xx1x000xx0x011xx1x100xx0x001xx0x11
0xx1x011xx1x110xx1x001xx0x011**

Innanzitutto separiamole:

**xx1x110 xx0x011 xx0x110 xx1x000 xx0x011 xx1x100
xx0x001 xx0x110 xx1x011 xx1x110 xx1x001 xx0x011**

Adesso invece di trasmetterle in sequenza, trasmettiamo le codeword colonna per colonna, nel modo che vediamo sotto. Supponiamo che a un certo punto poi ci sia una raffica di errori. Organizzando in questo modo otteniamo qualcosa del genere (vedi prossima pagina):

**xxxxxxxxxxxxxxxxxxxxxx100101001110xxxxxxxxxx101001
010100111010011101010010101011**

Codeword trasmesse colonna per colonna

```
xxxxxxxxxxxxxxxxxxxxxx100101001110xxxxxxxxxxxx101001  
010100000101111101010010101011
```

```
xx1x100  
xx0x001  
xx0x100  
xx1x010  
xx0x001  
xx1x110  
xx0x011  
xx0x110  
xx1x011  
xx1x110  
xx1x001  
xx0x011
```

La raffica di errore si va a distribuire su codeword differenti, quindi sono tutti errori gestibili con i codici di hamming.

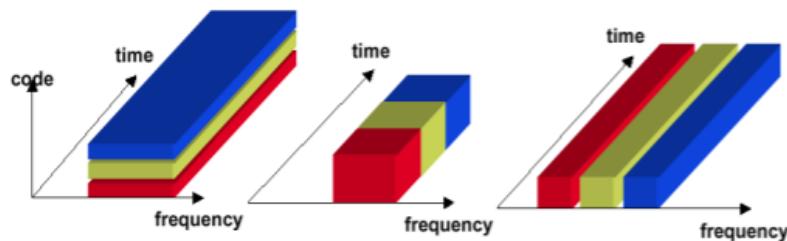
Lo svantaggio di questo sistema è che dobbiamo bufferizzare prima di trasmettere e poi trasmettere in colonna, e a destinazione fare l'operazione inversa, che comporta un certo ritardo di trasmissione.

Accesso al Link (MAC SubLayer)

Altro compito del livello DLL: Come gestire l'accesso al mezzo fisico (Medium Access Control Sublayer). Se il canale è punto-punto bidirezionale, non serve neanche un protocollo.

Il problema sta nei mezzi condivisi, l'accesso deve essere regolato da un protocollo, altrimenti si hanno collisioni nel canale.

Il protocollo può essere centralizzato o distribuito. Si preferisce distribuito perché avere una macchina unica che gestisce l'accesso è un punto di guasto che sarebbe meglio evitare.



Il volume è uguale in tutte e tre le tipologie di accesso multiplo

TDMA

“Time”, si parla a turno, si chiede in un qualche modo il permesso di parlare e si parla in uno slot di tempo predeterminato.

Come si identificano gli slot, quanto può ogni utente occupare uno slot di tempo, e chi li stabilisce?

Un “arbitro” stabilisce gli slot, una volta conclusa questa operazione non serve più ripeterla. In caso di guasto quindi le macchine potranno continuare a comunicare, semplicemente gli slot resteranno sempre ai rispettivi proprietari.

Uso tutto lo spettro delle frequenze ma in momenti differenti (vd figura)

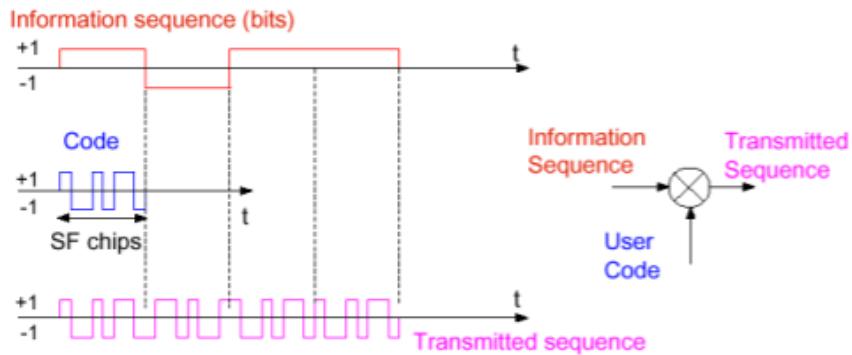
FDMA

“Frequency”. sullo stesso schema si può pensare a un sistema a frequenze. Un esempio banale è comunicare con dei laser colorati diversamente, e con un filtro ottico si fa passare solo una certa frequenza nei canali. Questa cosa si fa in maniera simile con le radio sequenze.

Utilizzo lo slot di frequenza assegnatomi per tutto il tempo, ma solamente il mio slot (vd figura)

CDMA

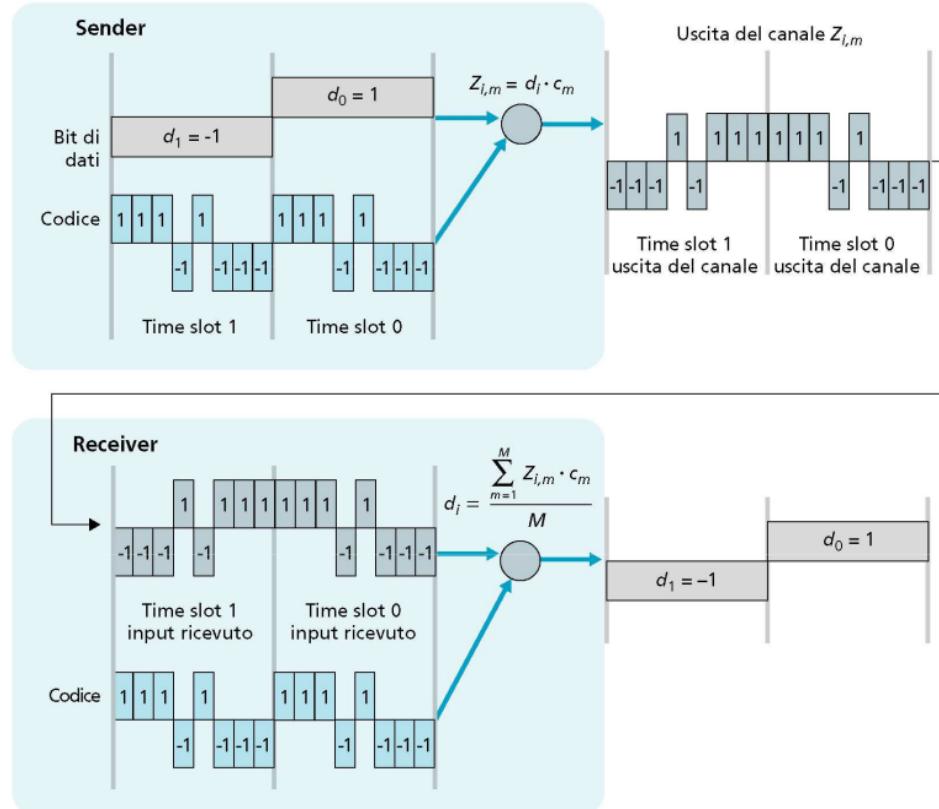
“Code”, risale alla WW2.



In realtà quando abbiamo una collisione, vi è una sovrapposizione di segnali, la chiamiamo una collisione perché non riusciamo più a capire i dati racchiusi in un segnale.

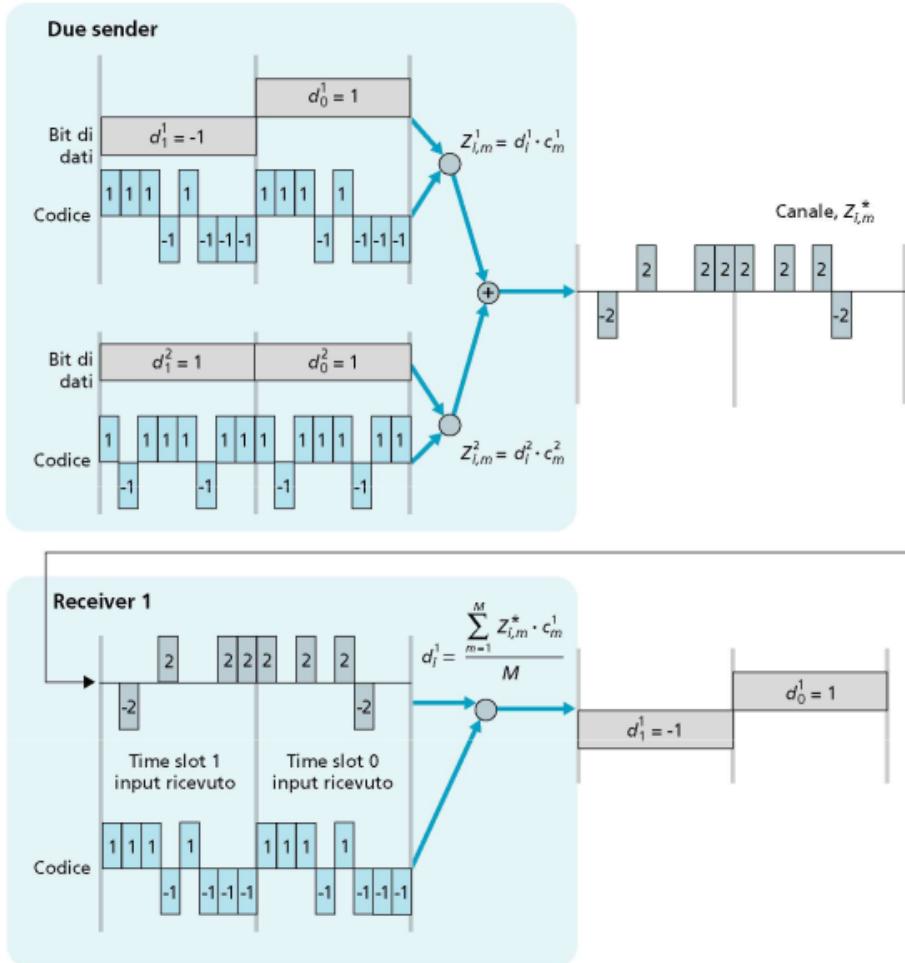
L'idea del CDMA è di, prima di mandare i bit (con +1 o -1, il valore del segnale) sul canale, andiamo a imprimere un codice facendo una banale moltiplicazione (Codice * Segnale).

Trasmetto sul canale la frequenza manipolata, ecco cosa succede:



Il ricevente poi riapplica lo stesso codice per ottenere la sequenza originale.

Facciamo ora qualcosa di più interessante: applichiamo due trasmettitori con due codifiche differenti, con una logica di separazione ben precisa.



Così riusciamo ad avere due trasmettitori che trasmettono contemporaneamente sulle stesse frequenze, nello stesso tempo, sovrapponendo i segnali, ma senza rovinare il risultato finale, nel senso che possiamo ricostruire il segnale originale.

I codici devono essere tra loro linearmente indipendenti tutti fra di loro.

Accesso Casuale

Questi protocolli sono distribuiti e non centralizzati.

ALOHA

Sviluppato nelle Hawaii per realizzare una rete di comunicazione wireless tra i vari isolotti dell'arcipelago.

L'idea di ALOHA è di avere una serie di nodi che non sono coordinati: metto una dimensione di frame fissa ben determinata. La regola è che parla quando vuole, se deve comunicare trasmette, altrimenti no. Si usava un'antenna per trasmettere e una ricevente nelle stazioni periferiche, con un'antenna centrale che fungeva da ripetitore. Quindi la stazione periferica che voleva trasmettere ad un'altra, trasmetteva la sua frame quando voleva, la frame arrivava all'antenna centrale che ripeteva la frame su un determinato canale di ripetizione. In caso di collisione veniva trasmesso il segnale rovinato.

Se invece nessun altro disturbava il canale allora arrivava correttamente a destinazione.

Qual è il periodo critico per il quale la frame poteva essere rovinata?

Solo se inizia a parlare mentre una frame sta per finire, o se qualcun altro inizia a parlare proprio mentre sta per finire la frame corrente.

Con queste considerazioni si può determinare un modello matematico: Siano G i tentativi al secondo e S il throughput per tempo di frame.

Il throughput massimo si ha per $G=0.5$, con $S=0.184$: In canale viene usato al 18.4 % delle sue potenzialità.

Una cosa interessante: una macchina poteva rilevare se una frame era stata rovinata prima ancora di ricevere riscontro dai livelli superiori, semplicemente ascoltando il canale di ritorno. Se sul canale di ritorno riusciva a sentire la sua frame vuol dire che non era stata rovinata, se sentiva rumore invece il trasmettitore doveva ritrasmettere il messaggio.

Quindi nonostante non fosse definito in maniera esplicita, di fatto c'era un controllo di collisione, proprio perché i canali di andata e ritorno erano separati.

Slotted Aloha

Ha prestazioni all'incirca doppie rispetto ad ALOHA puro. Però ad aumentare della richiesta di occupazione del canale aumentano anche le connessioni. Cioè entrambi i sistemi ALOHA funzionano bene con basso traffico.

È stato riciclato nel sistema GSM per i telefoni.

CSMA

Se uso lo stesso canale però raddoppio la banda, e poi perché disturbare una macchina che già sta trasmettendo? Posso introdurre il sistema **Carrier Sense**.

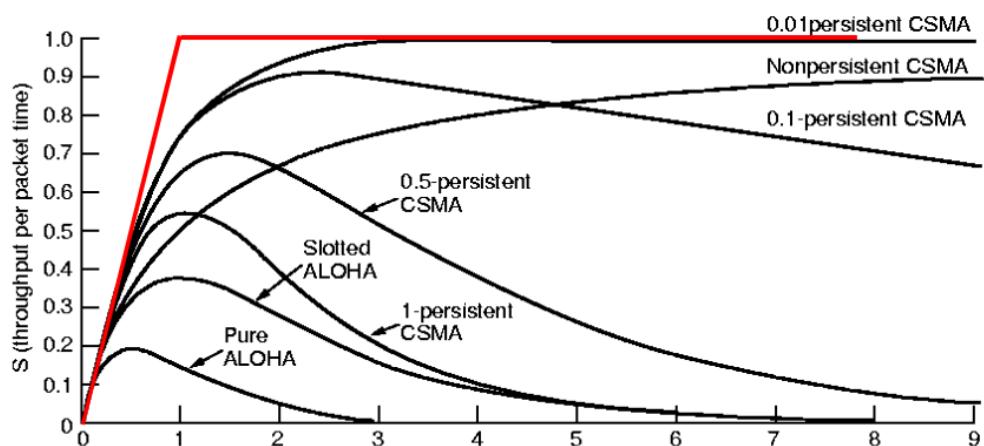
Prima di cominciare a trasmettere ascolta il canale (AKA se c'è una portante di modulazione), e se c'è qualcuno che sta già parlando sta zitto.

Posso trasmettere con sicurezza appena scopre la portante, che comporta il seguente possibile errore: due macchine bloccate sulla stessa portante iniziano a trasmettere non appena rilevano la fine della trasmissione, disturbandosi tra loro (**1-persistent**).

Oppure, trasmettere con una certa probabilità p , appena sento che la trasmissione precedente è terminata, se sono fortunato trasmetto, altrimenti sto ancora zitto e aspetto un altro po' prima di ritentare la trasmissione (**p-persistent**)

Oppure ancora, posso fare lo stesso discorso ma aspetto un tempo random prima, cioè alla fine del Carrier Sense aspetto un tot di slot prima di tentare la trasmissione, e se non c'è nessuno dopo la fine di questi slot, trasmetto. (**non-persistent**)

Anche qui possiamo ottenere un modello matematico:



Nelle ordinate il throughput che arriva a destinazione indisturbato, nell'ascissa quello che viene richiesto dalle singole stazioni, che in questo caso arriva a 9 volte la capacità del canale.

La “curva” rossa (che sarebbe una bisettrice se il grafico fosse fatto in scala) rappresenta il throughput ideale.

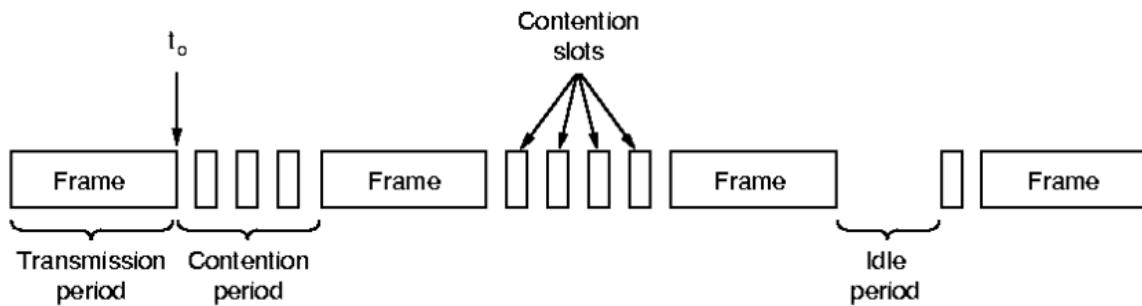
Da questo schema la soluzione migliore *sembrerebbe* 0.01.

Però attenzione: 0.01 significa che con probabilità 99% sto zitto, e trasmetto solo con probabilità 1%. Quindi non è la politica migliore (è il peggiore in realtà)

Lezione 18 - 20/05/2020

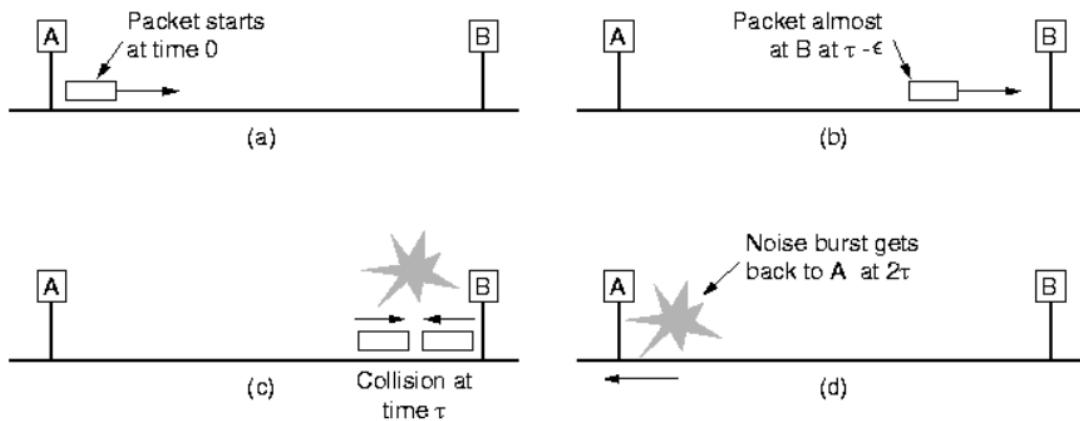
CSMA e /CD

CSMA/CD, il CD sta per Collision Detection:



CSMA/CD, periodi di trasmissione e periodi di collisione in cui i trasmittenti stanno fermi

Finita la trasmissione della frame, in tempi non proprio coincidenti tutti rilevano che non c'è più segnale sul mezzo di trasmissione e quindi si può cominciare a trasmettere. Qualcuno lo sente prima e qualcuno dopo per differenze di ritardo di propagazione.



A e B hanno rilevato che il canale è libero, e possono comunicare tra loro se non tramite il canale.

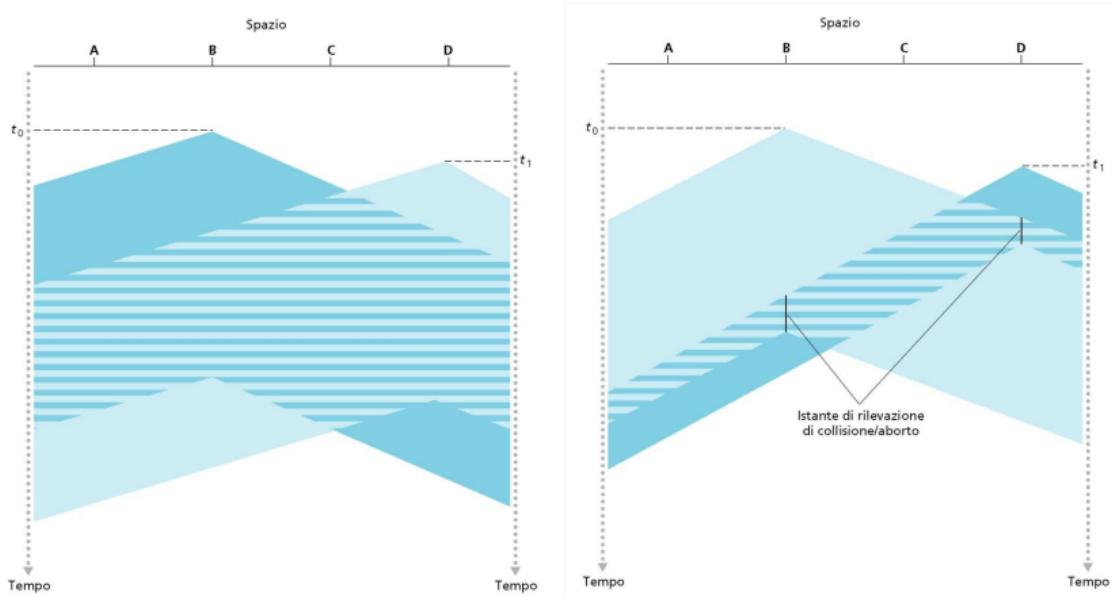
La politica CSMA/CD ci dice che appena il canale è libero i nodi devono trasmettere, se devono trasmettere.

A comincia a trasmettere il pacchetto, e il suo segnale si propaga lungo il canale. Arriva in prossimità di B ma ancora B non ha sentito il suo segnale. Se continua a fare il

monitoraggio del canale con CSMA (senza CD) rileva che ancora non c'è nessuno che sta utilizzando il canale, ma è un falso, semplicemente ancora il segnale di A non è arrivato a destinazione.

Un ϵ prima che il pacchetto di A arrivi a destinazione, B inizia a trasmettere a sua volta, e i segnali collidono (si sovrapppongono, diventa collisione se non si riesce più a distinguere i rispettivi pacchetti di A e B). Il segnale di A continua verso B, e rileva che c'è una collisione quasi subito. Dopo un po' di tempo anche A rileva che c'è stata questa collisione. Sia τ il tempo di propagazione del segnale da A verso B, allora A individuerà la collisione in un tempo 2τ , nel caso peggiore.

Andiamo a vedere in un diagramma temporale il comportamento delle macchine (in una situazione che non è la peggiore in assoluto). Le macchine che trasmettono sono B e D:



A sinistra CSMA, a destra CSMA/CD

B comincia a trasmettere, il "triangolo" rappresenta la propagazione del segnale a B verso le altre macchine. D comincia a trasmettere poco prima di ricevere il segnale di B. A sinistra la frame viene quasi del tutto rovinata. A destra tentiamo di stringere la parte di collisione (interrompendo la trasmissione) riusciamo a liberare il canale il prima possibile. B e D sono in grado di rilevare la collisione non appena i due segnali iniziano a sovrapporsi. Ciononostante, senza /CD, le due macchine continuano a trasmettere imperterriti.

Rilevando la collisione invece, posso far bloccare i trasmittenti il prima possibile dopo la rilevazione della collisione. Se la frame è molto più grande del periodo di collisione ha senso fare CSMA/CD.

In un sistema p-persistent potevano capitare situazioni del tipo: due trasmittenti provano a trasmettere nello stesso periodi più volte, fino a quando una delle due non rinuncia.

CSMA/CD invece non tenta in base a una probabilità, ma in base a un tempo. Dice al trasmittente: (vedi prima figura del paragrafo) Nel *contention period*, al primo slot disponibile fai una prova. Se ti va bene, tutto ok, se rilevi una collisione al prossimo tentativo lancia un dato che ti determina un valore reale, non probabilistico, 0-1. Se ti esce 0 provi subito, se ti esce 1 provi al secondo slot.

L'idea è che se al primo tentativo ci sono due macchine che collidono, al secondo tentativo tutte e due le macchine genereranno un numero tra 0 e 1. Il caso migliore è che generino numeri diversi, ovviamente. La macchina che trasmette è tranquilla, l'altra rileva la portante e non trasmette.

Se tutte e due generano 1, entrambe le macchine rimangono in idle.

Al secondo tentativo i numeri si raddoppiano (0,1,2,3), poi si raddoppiano ulteriormente fino a un massimo di 10 volte (quindi si possono avere 1024 slot).

All'ultimo tentativo possibile le due macchine devono essere talmente sfortunate da beccare lo stesso slot su 1024 possibili.

Quando le due macchine sono in idle danno comunque la possibilità ad altre di trasmettere.

La dimensione degli slot è pre-determinata, quindi la possiamo porre pari a τ .

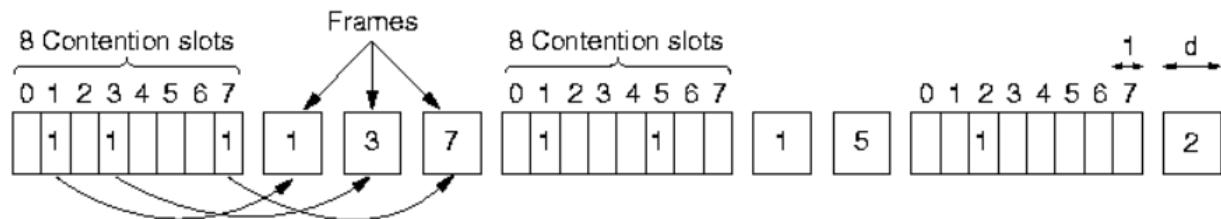
(vedi sotto per altri dettagli, nella sezione Fast Ethernet)

Protocolli senza collisioni

È possibile in uno schema di accesso multiplo che garantisce l'assenza di collisioni?

Sì ma sono complicati con forti svantaggi:

Bitmap



Si ha un numero di slot pari al numero di stazioni (numerate) che possono parlare contemporaneamente nel mezzo.

Se dico che ci sono 50 macchine devo avere 50 slot separati.

Le macchine che vogliono trasmettere mettono 1 bit (o anche rumore, volendo) nel loro slot.

Terminato il periodo di contesa, tutte le macchine sanno chi vuole parlare, avendo ascoltato il canale, e allora trasmetteranno in ordine.

Non garantisce la totale assenza di collisione, ci può ancora essere perché qualcuno potrebbe non rispettare il protocollo.

Tutti sanno quanti slot ci sono e quanto è grande un singolo slot.

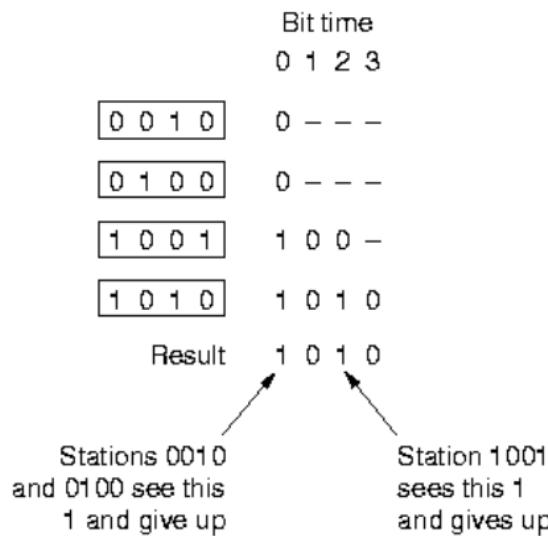
Problemi:

1): Se ci sono tante macchine il periodo di contesa è inutilmente grande, e considerando che le frame hanno la loro ridondanza, quello è un ulteriore ritardo inutile. Inoltre, è uno schema che ha senso se tutte le macchine presenti tendono a parlare. Altrimenti, se abbiamo tante macchine che parlano poco il canale rimane inutilizzato per gran parte del tempo.

2): A priori devo sapere quante macchine ci sono nel mio sistema, non posso aggiungere o togliere macchine a piacimento. Non basta "aggiungere" uno slot, perché devo informare tutte le macchine, se qualcuno non dovesse capire questa informazione (dato che non c'è sincronizzazione) qualche macchina potrebbe interpretare l'ultimo slot come momento dove cominciare a trasmettere.

CSMA/BA CanBus

Conteggio binario a ritroso.



Invece di distribuire gli slot con una bitmap, lo schema di comunicazione sul canale è tale che se voglio trasmettere un bit 0 sto zitto, altrimenti metto un segnale alto.

Se due macchine dicono “1” contemporaneamente il segnale non va in collisione, il risultante è 1. Se una dice 0 e una 1 vince il bit 1 e lo 0 scompare, neanche qui c’è collisione.

Vengono assegnati dei numeri a ogni stazione (per esempio, in questo caso, 0010, 0100, ...). Ogni macchina mette il proprio bit più significativo sul canale, e tutte ascoltano.

Al passaggio successivo chi trova il suo bit differente sta zitto. Le macchine che avevano 0 e che sentono 1 sul canale non continuano ma si fermano. Al passaggio successivo tutte e due dicono 0 e continuano.

Al terzo passaggio una dice 0 e l’altra 1, vince quella che ha messo 1, l’altra si è fermata al quarto passaggio.

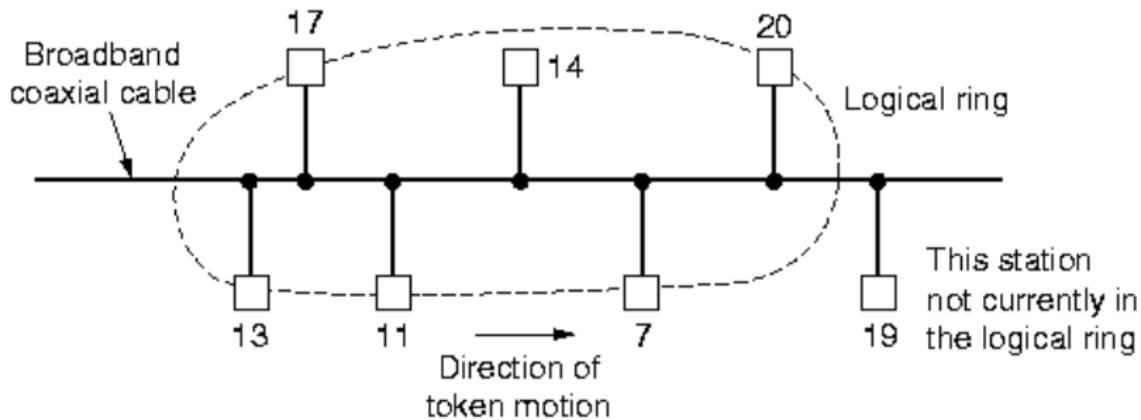
Di fatto questo non è uno spreco di banda, perché magari il “numero” della stazione può essere il suo indirizzo mittente, quindi ha già iniziato a scrivere il suo indirizzo.

Vantaggi: Non ho gli slot e sto anche iniziando a scrivere il mio indirizzo (e quindi la mia frame). Non c’è ridondanza dagli slot.

Non c’è bisogno di dire a priori quante macchine possono trasmettere. Usando n bit dirò semplicemente quante macchine possono esserci **al più**.

Però se ho un indirizzo alto ho proprietà sulle altre macchine. Non è per forza uno svantaggio.

Token Bus



L'idea è di realizzare un anello logico che si passano il token (permesso) per parlare. Nella figura le macchine presenti nell'anello sono 13, 11, 7, 20 e 17.

Chi ha il token occupa il canale, al termine passa il token alla macchina successiva. Il token è una frame e potrebbe collidere con altre frame o anche perdersi a causa di interferenze. Se si perde il token non finisce tutto.

Tutte le macchine sorvegliano il canale per individuare il passaggio di token. Se dopo un certo tempo (più grande del tempo precedente) tutte notano che il token non sta girando allora tutte sono autorizzate a generare un nuovo token, nella speranza che venga generato un solo token alla volta.

Se ci sono due token si deve prevedere una politica per la rimozione del superfluo. Poi una macchina deve poter entrare/uscire dall'anello. Uscire è facile, ma come si fa per entrare?

Deve essere previsto un momento di contesa dove tutte le macchine che vogliono entrare provano ad utilizzare il canale, con la possibilità di collisione.

Richiede software molto complesso.

IEEE 802

"I triple e", Insieme di protocolli per l'accesso al mezzo fisico

802.1 **High Level Interface** (HILI)

802.2 **Logical Link Control** (LLC) [in 'hibernation']

802.3 **CSMA/CD**

802.4 **Token Bus** [in 'hibernation']

802.5 **Token Ring** [in 'hibernation']

802.6 **Metropolitan Area Network** (MAN) [in 'hibernation']

802.7 **BroadBand Technical Adv. Group** (BBTAG) [in 'hibernation']

802.8 **Fiber Optics Technical Adv. Group** (FOTAG) [disbanded]

802.9 **Integrated Services LAN** (ISLAN) [in 'hibernation']

Le espansioni di CSMA/CD sono indicate con lettere (802.3a, .802.3b, ...).

802.10 **Standard for Interoperable LAN Security** (SILS) [in 'hibernation']

802.11 **Wireless LAN** (WLAN)

802.12 **Demand Priority** [in 'hibernation']

802.14 **Cable-TV Based Broadband Comm. Network** [disbanded]

802.15 **Wireless Personal Area Network** (WPAN)

802.16 **Broadband Wireless Access** (BBWA)

802.17 **Resilient Packet Ring** (RPR)

802.18 **Radio Regulatory Technical Advisory Group**

802.19 **Coexistence Technical Advisory Group**

Cablaggi

Name	Cable	Max. seg.	Nodes/seg.	Advantages
10Base5	Thick coax	500 m	100	Original cable; now obsolete
10Base2	Thin coax	185 m	30	No hub needed
10Base-T	Twisted pair	100 m	1024	Cheapest system
10Base-F	Fiber optics	2000 m	1024	Best between buildings

La prima definizione standard di 802.3 definiva una velocità da 1 Mbit (e una da 2 Mbit) ma non hanno mai avuto un'implementazione.

Quello più comunque è stato il 10Base[X]. 10 vuol dire 10 Mbps. Base è modulazione in banda base e poi il tipo di cavo.

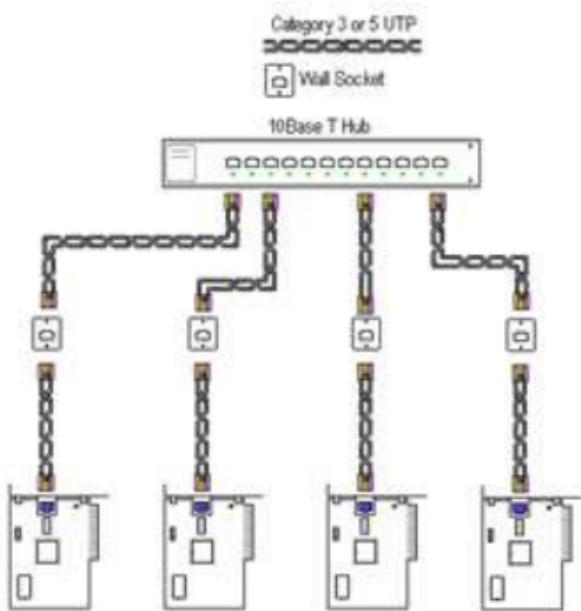
Il primo è il cavo coassiale grosso (thick), **10Base5**, molto rigido perché era un pezzo di rame unico non intrecciato, una guaina isolante e una calza metallica di protezione.

Ovviamente poi c'è la guaina esterna di protezione totale. La lunghezza massima per un segmento unico era di 500 metri, e potevano essere collegati tra di loro con dei ripetitori / amplificatori di segnale fino a un massimo di 5 segmenti consecutivi, quindi 2.5km.

Come intercettare il cavo interno senza rovinare il cavo esterno e senza fare cortocircuito? Si usava la presa a vampiro. Forava la parte isolante e andava a fare contatto nella parte interna. Per non fare cortocircuito si va a tagliare la guaina in dei punti specifici segnati da dei puntini verdi (ogni 2.5 metri, al massimo però si potevano mettere solo 100 nodi, perché altrimenti si va a "perdere" il segnale, dopo 100 nodi si metteva un amplificatore).

Per quanto riguarda **10Base2**, thin coax, è molto più flessibile e per estendere il cavo semplicemente si tagliava (cosa che rovina il segnale) e si inseriva il connettore. Non avere più il cavo unico è un problema e quindi ha segmenti massimi di 185-200 metri, sempre 5 segmenti massimo fra di loro, quindi arriva a stento a 1km,e 30 nodi a segmento.

10Base-T, Twisted Pair, cavo a coppie intrecciate, ha 4 coppie, ogni coppia ha un colore pieno e un colore bordato di bianco, questi cavi permettono comunicazione massimo 100m con più nodi, questo perché lo schema di collegamento prevedeva un hub centrale, un concentratore, e la connessione era diretta tra scheda di rete e concentratore. Possiamo vederlo anche come un centro a stella.



Non previene le collisioni.

L'ultimo schema è **10Base-F**, la fibra ottica (la velocità è sempre la stessa). Può andare fino a due km e potremmo avere fino a 1024 nodi, ma in realtà la comunicazione con fibra ottica è punto-punto, come quello intrecciato. È solo un valore teorico.

Ci sono altre categorie di cavi intrecciati, ad esempio:

Category	Data Rate	Signal Frequency	Standard
Cat5	100 Mbps	100 MHz	TIA/EIA
Cat5e	100 Mbps / 1 Gbps	100 MHz	TIA/EIA-568-B
Cat6	1Gbps / 10 Gbps	250 MHz	TIA/EIA-568-B
Cat6a	1Gbps / 10 Gbps	500 MHz	ANSI/TIA/EIA-568-B.2-10

Per 10Base-[X] basta un cavo Cat3 o Cat5.

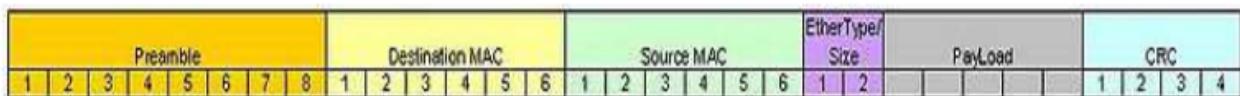
Ethernet 10BaseT:

Color	Pin (T568B)	Usage
White/Orange	1	Transmission (Tx+)
Orange	2	Transmission (Tx-)
White/Green	3	Receive (Rx+)
Blue	4	--
White/Blue	5	--
Green	6	Receive (Rx-)
White/Brown	7	--
Brown	8	--

Prevede 8 poli (da 4 coppie). Questo sopra è uno schema ma a volte ven

Ethernet

Frame Ethernet



Abbiamo in primis il MAC address di destinazione e sorgente. (6 e 6 Byte)

Il campo Ethernet-Type / size (2 Byte) indica la lunghezza del campo dati per valori inferiori a 1500, mentre indica il tipo di frame per valori superiori. In origine specificava la dimensione del payload. Siccome però la dimensione si può ricavare a seconda del tipo di framing utilizzato (come visto prima).

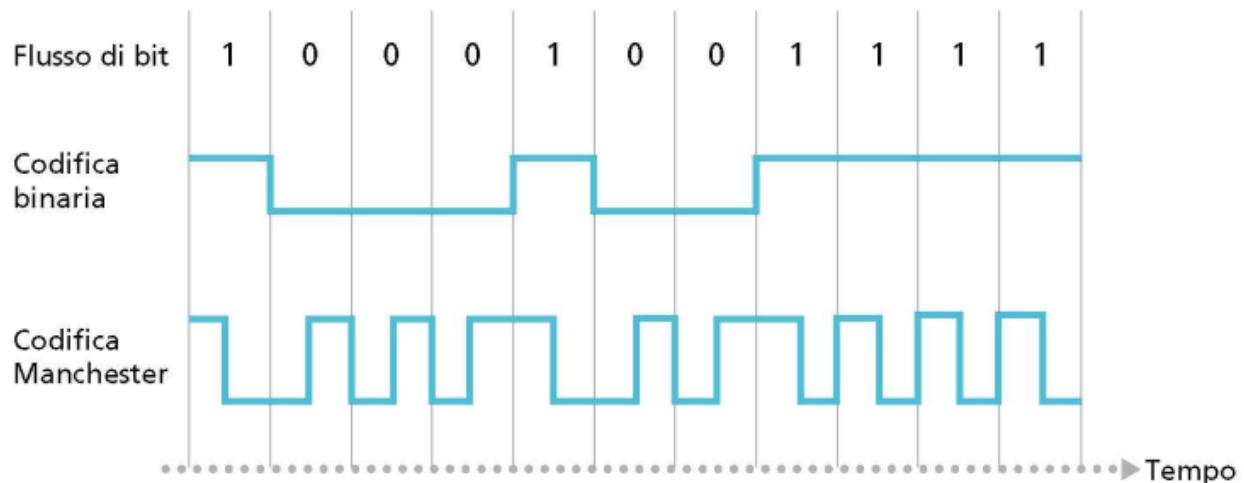
Il campo tipo viene utilizzato per il mux / demux sul livello network. (Ad esempio 0x0800 significa IPv4, 0x86DD IPv6, etc).

Se il valore è superiore a 1500 allora il campo è Ethertype, altrimenti è size.

Il campo del CRC è 4 Byte.

Codifica Manchester

Tipo di codifica utilizzato nel passaggio livello fisico-DLL.



Ci sono due livelli, alto basso, il bit 1 è identificato da un passaggio alto-basso, mentre il bit 0 da un passaggio basso-alto. Devo però identificare i “tratti verticali” da un bit e l’altro, per non confonderli con dei bit. Faccio questo perché individuare una variazione è più semplice che capire che c’è un segnale stabile.

Per scandire bene i passaggi tra un bit e l’altro dobbiamo sincronizzare mittente e destinatario, operazione che è svolta dal preambolo.

Il preambolo è costituita da una sequenza nota a priori a entrambi. Il destinatario non deve fare altro che sincronizzare il suo clock con quello del mittente. La cosa è semplice perché entrambe le macchine conoscono la sequenza da usare ed è composta da successioni di **10101011**

La sequenza 101010 determina una onda quadra dove ci sono solo variazioni nel mezzo del bit (tra un bit e l’altro).

La sequenza 11 è una forma d’onda dove la frequenza è il doppio di quella precedente (onda quadra a 20 MHz), che dice “il preambolo è finito”. Quindi anche se perdi l’inizio appena arriva la coppia 11 il destinatario sa che è finito il preambolo e può comunque iniziare la comunicazione. Può iniziare la comunicazione ignota (dal Dest MAC Address).

In Wireshark non si trova perché non sono dati.

Esiste anche la codifica di manchester differenziale, dove 1 rappresenta una variazione, lo 0 invece è una ripetizione di qualsiasi cosa c’era prima(???). È meglio dal punto di vista statistico perché diminuisce il tasso di errore ma non ci interessa più di tanto.

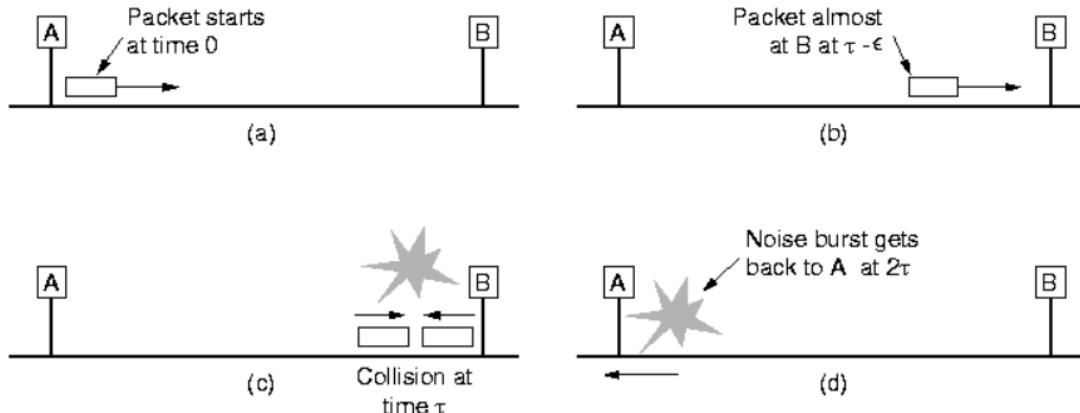
Fast Ethernet

Name	Cable	Max. segment	Advantages
100Base-T4	Twisted pair	100 m	Uses category 3 UTP
100Base-TX	Twisted pair	100 m	Full duplex at 100 Mbps
100Base-FX	Fiber optics	2000 m	Full duplex at 100 Mbps; long runs

Aumento della velocità di Ethernet. I tre schemi standardizzati sono 100Base T4, TX e FX. Sono scomparsi i cavi coassiali.

---INTERMISSION---

Cosa dimenticata CSMA/CD



Il tempo critico è pari al tempo di propagazione massimo moltiplicato per due, con Ethernet abbiamo la dim massima che è 2.5km.

Facendo due conti, scopriamo che alla velocità di 10Mbit in questo schema qui la macchina A nel tempo di propagazione del segnale (51.2ms) riesce a spedire 512 bit, cioè 64 Byte.

Secondo questo schema nei primi 64 Byte che trasmette la macchina A non può avere la certezza che la sua frame non ha avuto problemi di collisione. Passati 64 Byte ormai eventuali collisioni nella situazione peggiore sono già tornati indietro. Per cui se continui a trasmettere la frame è sicura, non ci sono stati problemi.

Immaginiamo che A trasmetta meno di 64 Byte: allora non può avere la certezza che la sua frame eventualmente è stata rovinata da una collisione, perché potrebbe anche riguardare qualche altra frame.

L'idea di Ethernet è di realizzare frame di dimensione minima di 64 Byte, allora se sento una collisione ho la certezza che la mia frame è stata rovinata e allora la devo ritrasmettere subito.

Quindi collision detect mi serve non solo per sapere che devo fermarmi a causa della collisione, ma anche che la mia frame è stata rovinata e la devo ritrasmettere subito dopo.

Se la faccio più piccola la collisione la sento, ma non posso avere la certezza che la mia frame è arrivata a destinazione senza problemi.

Come si realizza la dimensione minima della frame?

$64 - 18 = 46$. Dimensione minima del payload è 46 Byte, se non ho nulla da trasmettere metto un campo PAD per fare arrivare a 64 Byte la dimensione minima di tutta la frame.

---FINE INTERMISSION---

Quando passo a fast Ethernet, e prevedo le collisioni, la lunghezza massima del cavo è quella che mi determina la dimensione minima della frame.

Uso tutte e 4 le coppie presenti nel cavo, una in andata, una in ritorno, alla velocità di 33 Mbps ciascuna, le ultime due coppie le utilizzo alternativamente o in una direzione o nell'altra, in modo tale che ho 33x3 (più un epsilon) in una direzione e 33 nell'altra.

Quindi 100 in una direzione e 33 nell'altra. Quindi non ho 100 Mbps Full duplex, ma solo dove mi serve.

Questo permetteva di usare i vecchi cavi telefonici aumentando la velocità di comunicazione. Ovviamente per la fibra non ci sono problemi

Viene abbandonata la codifica Manchester, nel cavo TX uso 4B5B (cavo CAT5), mentre per il T4 (CAT3) uso 8B6T. (visto prima, a 3 livelli, ne metto 6 consecutivi per trasmettere 600 combinazioni differenti, di cui ne usiamo solo 256).

Lezione 19 - 22/05/2020

Gigabit Ethernet

Dato il successo di Fast Ethernet, si è pensato di aumentare ancora di più la velocità. Il problema è il cablaggio. Con Fast Ethernet si è già dovuto rinunciare al cavo coassiale a causa dei problemi di affidabilità che per problemi dovuti alla dimensione minima della frame (che per Ethernet era 64 Byte, mentre con Fast Ethernet con cavo coassiale avrebbe portato il tutto a 640 Byte, considerato troppo eccessivo).

Name	Cable	Max. segment	Advantages
1000Base-SX	Fiber optics	550 m	Multimode fiber (50, 62.5 microns)
1000Base-LX	Fiber optics	5000 m	Single (10 μ) or multimode (50, 62.5 μ)
1000Base-CX	2 Pairs of STP	25 m	Shielded twisted pair
1000Base-T	4 Pairs of UTP	100 m	Standard category 5 UTP

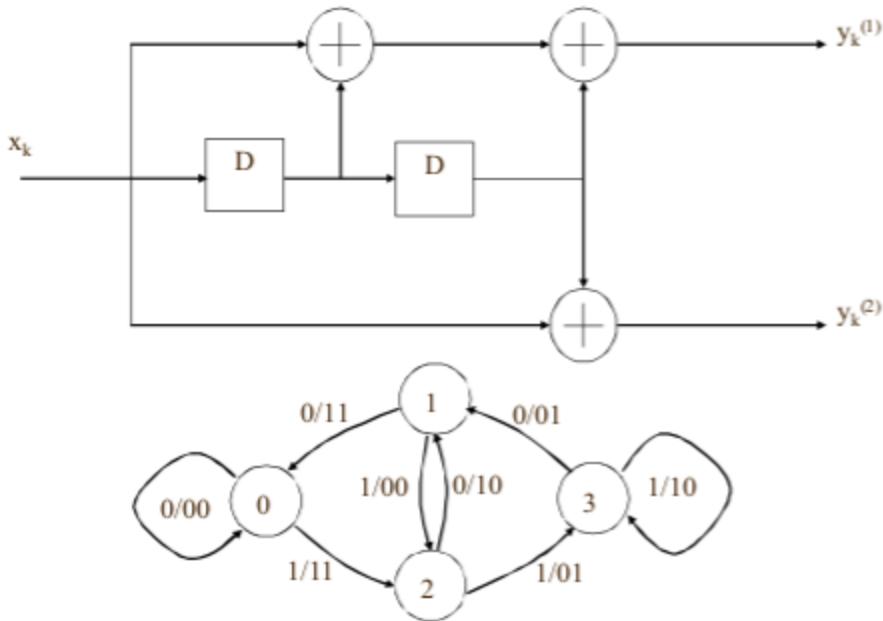
Questo è il cablaggio standard gigabit Ethernet, i primi due in fibra ottica, monomodale o multimodale, con distanze piuttosto notevoli.

Per quanto riguarda il cablaggio in rame vi sono due standard, uno con coppie schermate (STP), l'altro con quattro coppie non schermate (UTP), con distanze molto limitate.

L'innovazione è stata nel cercare di utilizzare i cavi già esistenti (CAT5e) utilizzando tutte e 4 le coppie, evitando di ripassare i cavi dove già erano presenti.

Cinque passi verso 1000Base-T (CAT5):

- Rimuovere codifica 4B5B (100 \rightarrow 125 Mbps)
 - Già solo rimuovendo questa codifica aumentiamo il throughput, ma ritorna il problema del framing.
- Usare le 4 coppie simultaneamente (125 \rightarrow 500 Mbps)
 - Fast Ethernet ne usa solo 2, una di andata e una di ritorno. Usando tutte e quattro le coppie simultaneamente possiamo quadruplicarlo.
- Trasmissione full duplex (500 Mbps full duplex)
 - Conseguenza dell'uso delle 4 coppie simultanee.
- Usare 5 livelli per baud invece che 3 (MLT-3) (500 Mbps \rightarrow 1Gbps full duplex)
 - Un livello viene usato per fare framing, gli altri 4 permettono di trasportare 2 bit al livello. Da 500 passiamo a 1 GBps. L'unico problema è che il tasso di errore è così alto che ogni frame sarebbe soggetto a errore.
- Usare Forward Error Correction per recuperare 6 dB
 - Correzione errori a destinazione per sistemare il problema.



Trasmettiamo a 2 Gbps invece di 1 Gbps, cioè inseriamo una ridondanza del 100 %. Però questo eccesso ci permette di recuperare gli errori. Il funzionamento base è dato dal “circuito” sopra (D= Delay Box, ritardano la propagazione del bit). Il segnale entra da x_k , per ogni bit entrante vengono generati due bit uscenti.

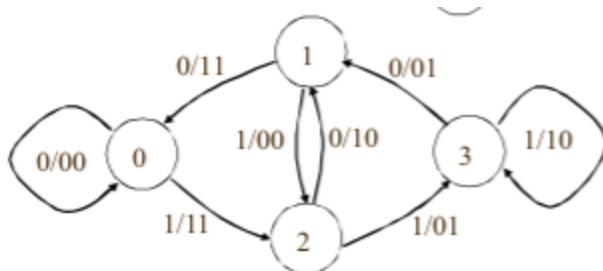
Possiamo vederlo più chiaramente con una FSM. Ci sono 4 stati: {0, 1, 2, 3}.

Baud

Un baud è la variazione di segnale dallo stato precedente. (La possibile variazione rispetto allo stato precedente).

Il baud al secondo rappresenta quante variazioni ci possono essere nell'arco del tempo. Ad ogni baud si può associare un solo bit come viene fatto in MLT3, oppure 2 bit nella codifica a 5 livelli (PAM5).

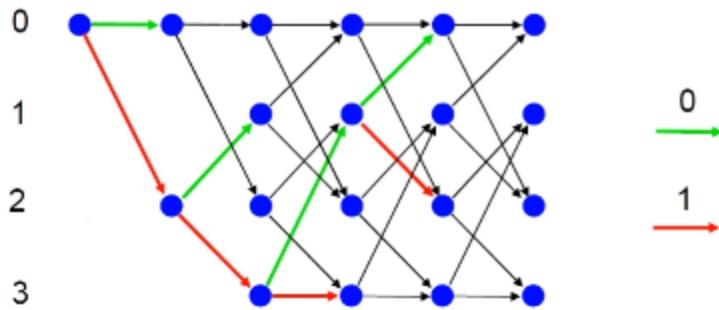
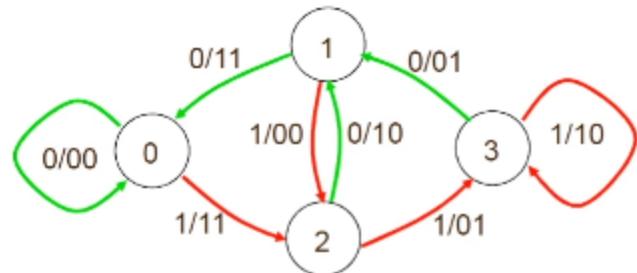
Back to Gigabit



Ci sono 4 stati: {0, 1, 2, 3}, ogni stato ha un suo comportamento e due frecce in uscita.

Ad esempio, lo stato 0 può restare nello stesso stato, se in input riceve il bit 0 e in uscita emetterà la coppia 00, o passare allo stato 2 se riceve il bit 1 e in output emetterà i bit 11.

Evidenziamo i bit 0 in **verde**, e i bit 1 in **rosso**:



Possiamo riprodurla con questo “traliccio” (sic) qui: Lo stato 0 può o restare in se stesso o passare allo stato 2. Gli stati 2 e 1 non possono mai restare in loro stessi e infatti variano sempre stato.

Data una sequenza di input genero un percorso all'interno di questo traliccio, tra quelli permessi dalla FSM.

Alcuni elementi notevoli:

0 e 3, 1 e 2 sono simmetrici tra loro.

Le uscite di 0 e 3 sono antisimmetriche. (0/00 - 1/10, 1/11, 0/01)

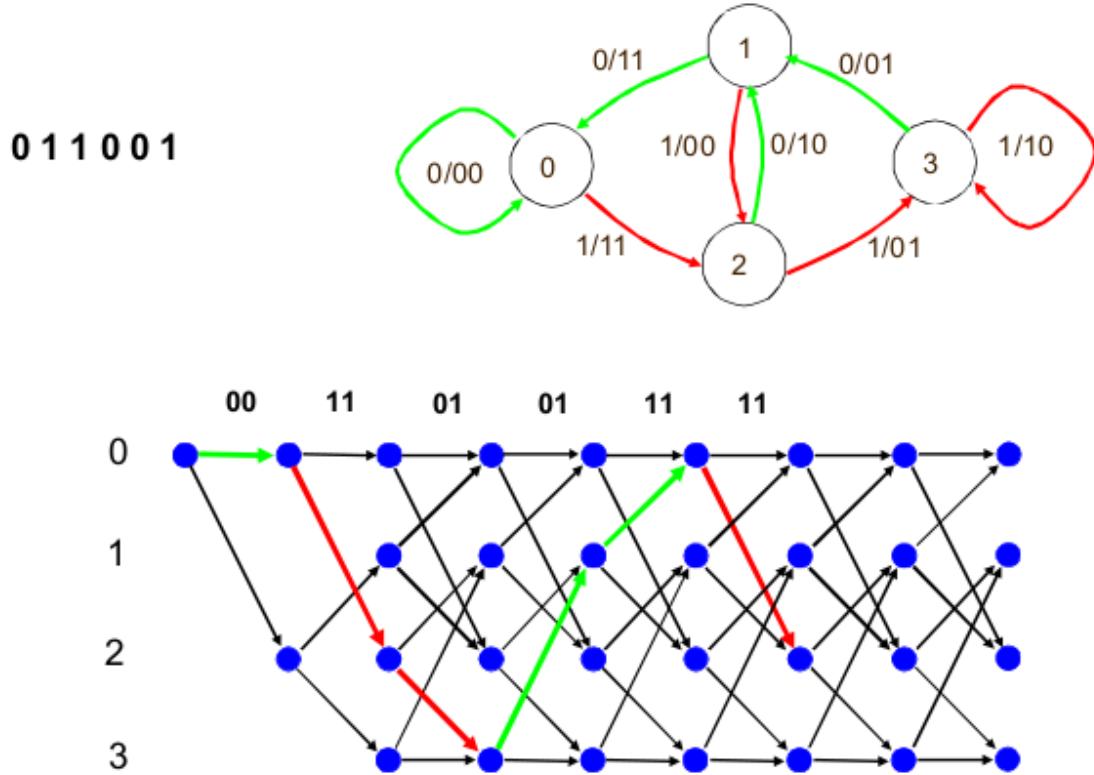
Le uscite di 1 e 2 verso gli stati 0 e 3 sono antisimmetriche (0/11 - 1/01).

Infine, le frecce entranti di 0 e 1 hanno come input 0, mentre le frecce entranti di 2 e 3 hanno input 1.

Considerando una sequenza abbastanza lunga, a distanza di 2 salti posso arrivare a qualunque altro stato.

Gli stati finali risultano “praticamente” equiprobabili. Sono tutti equiprobabili una volta passata la fase iniziale. Questo fatto ci permette di fare la correzione degli errori.

Esempio:



In input riceviamo la sequenza 011001, lo stato iniziale viene settato a 0, di fatto non cambia nulla se non il percorso.

La sequenza generata sarà 00 11 01 01 11 11.

A distanza di 1 salto posso avere tutte le possibili coppie di 2 bit emesse. Quindi o quelle dove i bit sono uguali o dove le coppie sono diverse. Questa regola è presente a partire da qualunque stato.

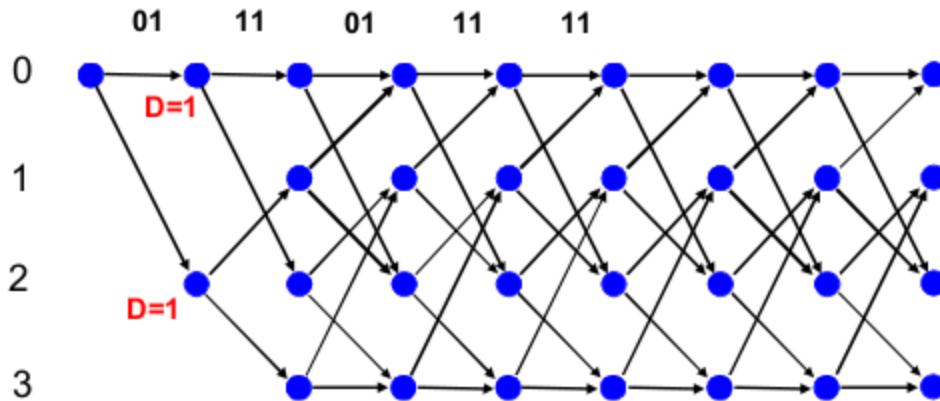
In particolare, 0 e 1 emettono coppie di bit uguali, 2 e 3 coppie diverse.

Inseriamo ora degli errori nella sequenza ottenuta, e vediamo se riusciamo a riconoscerli:

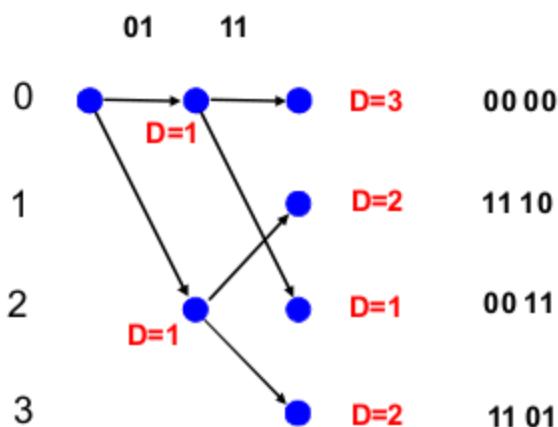
00 11 01 01 11 11
01 11 01 11 11 11

Usando sempre lo stesso traliccio e sempre lo stesso stato iniziale, cioè 0. Già da questa informazione vediamo che è presente un errore, perché lo stato 0 genera solo coppie di bit uguali, cioè 00 o 11.

Non sappiamo però se il bit sbagliato è il primo o il secondo. Tenendo conto di questa incertezza, prendiamo entrambi i possibili cammini, segnando però le distanze di Hamming rispetto a quello che poteva essere generato.



A questo punto andiamo a esaminare la coppia successiva, 11, semplificando lo schema:



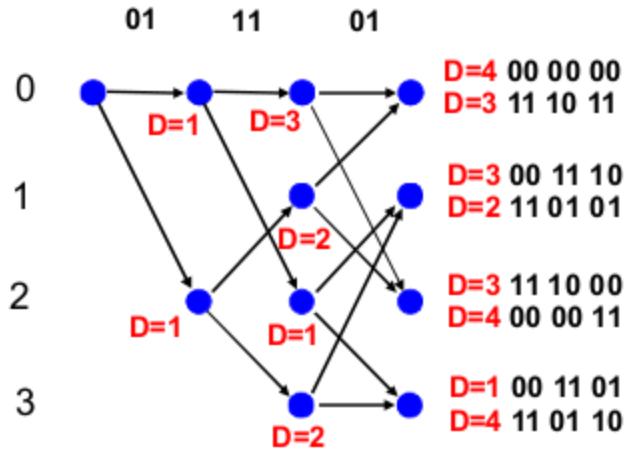
Abbiamo questi possibili percorsi, però:

Se ciò che doveva arrivare a destinazione era dovuto alla strada a partire dallo stato 0 (cioè 00 00), allora la differenza in bit è di 3, e porto D=3.

La strada 00 11 ha differenza D=1.

Le altre due strade hanno D=2.

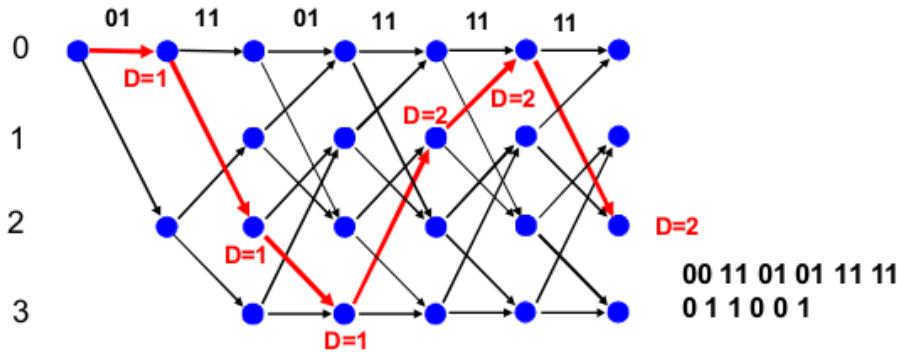
Se tutte le destinazioni sono equiprobabili, la probabilità che ci sia un solo errore è maggiore a quella di averne due o tre, quindi la strada da seguire era probabilmente quella a distanza D=1. La considerazione continua, nel senso che posso andare a vedere cosa succedeva nella coppia successiva, cioè 01:



Le strade diventano otto.

Nella sequenza di prima c'era un solo errore, se ce ne fossero stati due il sistema non sarebbe stato in grado di correggerli correttamente. Fallirebbe brutalmente, perché inizieremmo ad avere sequenze valide per la FSM.

Alla fine di tutti i possibili percorsi ne avremmo uno solo con D=2 (cioè i due errori sono stati trovati), e quindi il sistema avrà funzionato. Da questo deduciamo che il percorso corretto è quello che fa restare la distanza al minimo possibile tra tutti questi valori, e riesco a correggere.



Questo sistema ci permette di effettuare la correzione a destinazione inserendo un elevato numero di bit errati (1 bit errato ogni 6), a patto che i bit errati non siano troppo vicini tra loro (nella realtà sono spesso distanti).

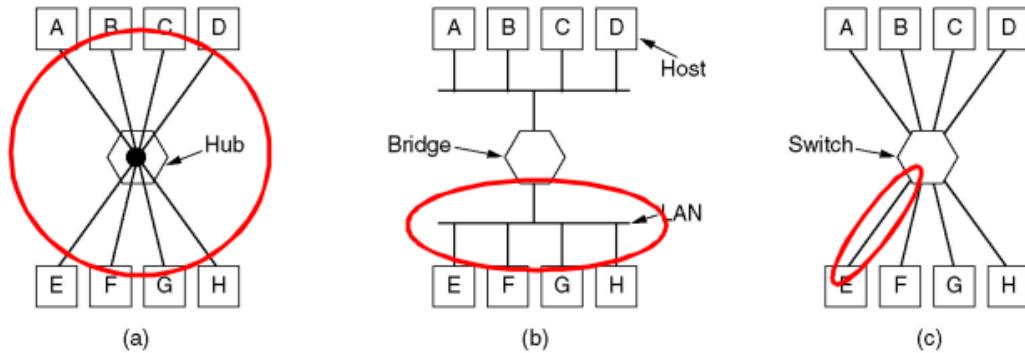
Nel caso di fallimento della codifica / decodifica e di questo sistema di controllo, controllo sempre il CRC (anche se pure questo non è una certezza).

Dunque, questo schema è importante perché è quello effettivamente usato in Gigabit Ethernet, e inoltre la FEC ha molte altre applicazioni all'infuori di Gigabit.

Collegamenti tra LAN differenti

Iniziamo con dei problemi di nomenclatura: Cos'è realmente una LAN?

La definizione originale di LAN riguardava tutto il sistema di bus condiviso dove potevano verificarsi le collisioni (aka il dominio di collisione):



In Ethernet 10Base-2 o -5 c'era un unico cavo di connessioni, tante derivazioni più o meno lunghe, ma di fatto c'era un unico mezzo fisico ben identificabile, per cui quello costituiva la LAN.

L'evoluzione è stata quella di passare da 10Base-2 a 10Base-T, cioè il cavo Twisted Pair. A questo punto si parla di Hub (fig. a), un concentratore che non faceva altro che riprodurre elettricamente i segnali provenienti da una coppia senza fare alcuna interpretazione del segnale. In questa maniera il dominio di collisione però comprende tutti i cavi che arrivano all'hub.

Le cose si complicano passando allo Switch: è un dispositivo intelligente che ha una scheda di rete per ogni interfaccia in ingresso, cioè non è più un amplificatore di segnale, ma legge le frame, le trasforma in bit e poi le ripropone per l'uscita corretta. Il dominio di collisione è ristretto fino alla singola tratta di collisione tra i due dispositivi, questo perché è presente una scheda ethernet che trasforma i bit in segnale nella macchina host, e lo switch ha una scheda di rete che trasforma il segnale in bit. Quindi eventuali collisioni fisiche su una tratta punto-punto non possono propagarsi sul resto della rete.

Quindi, restando con la definizione di LAN pari al dominio di collisione, lo switch realizza 8 LAN separate, cosa che sta un po' stretta.

C'è inoltre un altro dispositivo che si chiama Bridge (diverso dallo switch).

È un dispositivo intelligente (cioè provvisto di livello DLL a differenza dell'hub) con due interfacce distinte e separate. Per cui le frame della LAN E-H vengono interpretate e se necessario fatte girare verso l'altra uscita, verso la LAN A-D.

Dato che sono due interfacce separate totalmente, queste LAN potrebbero anche essere considerate differenti.

Se il bridge è connesso a due LAN Ethernet allora ha un compito facilissimo: Prendere la frame da un lato e se necessario riproporla all'altro.

Lo switch è praticamente un bridge specifico per Ethernet. Quindi, nel caso dello switch conviene estendere il concetto di LAN a tutta la struttura presente nella figura (fig. c), dato che è una struttura **uniforme** (molto importante), utilizza lo stesso tipo di frame (quasi identiche) e soprattutto sono operazioni che vengono fatte dallo switch senza aver bisogno di sapere cosa c'è sopra il livello DLL (cioè si guardano solo i MAC Address, lo switch potrebbe anche non riuscire a interpretare il livello IP, perché non c'è bisogno). All'interno sono presenti un processore, una memoria e un firmware operativo che permettono di far lavorare il dispositivo interpretando il contenuto delle frame Ethernet.

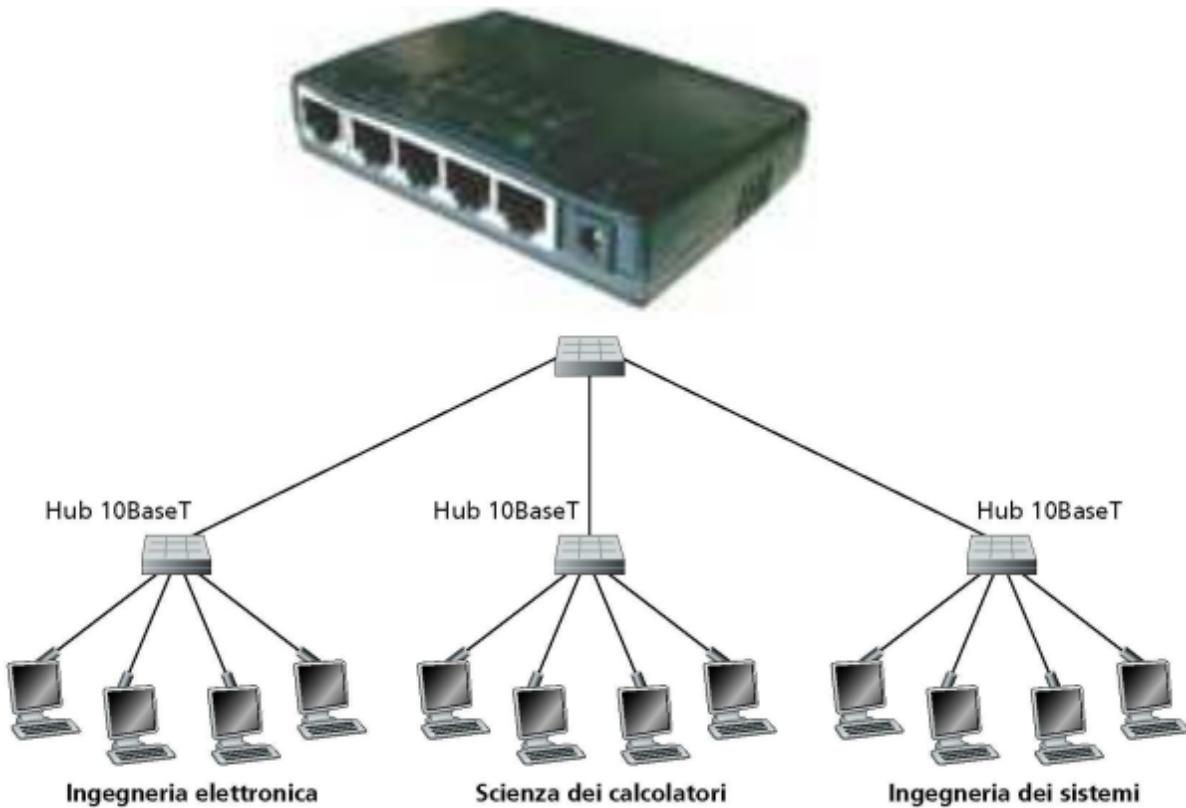
Quindi ormai il concetto di LAN non coincide più col dominio di collisione. Conviene in una **rete uniforme** parlare di LAN (per tutta la zona, compresi gli switch intermedi) dove si parla lo stesso linguaggio: Ethernet.

E se ho un dispositivo che ha un'interfaccia wireless? Cos'è la LAN?

È sempre qualcosa di opinabile, ma è più opportuno parlare di LAN cablata e LAN wireless collegate tra loro che non è più uno switch ma qualcosa di più complicato.

Questo perché il formato della frame WiFi (802.11) è differente rispetto alla frame Ethernet. Infatti ha 4 indirizzi nella frame, invece che 2 (anche se spesso se ne usano 3). Inoltre nelle reti wireless è prevista la presenza di un coordinatore (l'access point), cosa che nelle reti Ethernet non esiste.

Il formato degli indirizzi resta comunque uguale (6 Byte).

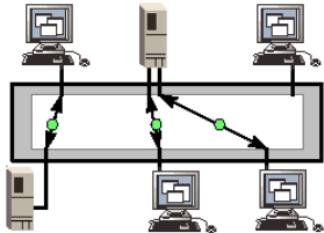


Per estendere di una struttura, di cosa abbiamo bisogno?

Serve un hub che ha un collegamento in downlink verso le singole macchine e uno in uplink verso un altro concentratore. Se questo concentratore è ancora un hub, allora il dominio di collisione coincide con TUTTO. Se al posto dell'hub centrale mettiamo uno switch i domini di collisione vengono limitati alle singole tratte, cosa che permette di aumentare notevolmente la larghezza di banda. Questo perché, con un hub centrale distribuiamo i 10 Mbps su tutte le macchine, mentre invece col bridge le macchine possono parlare tra di loro a 10 Mbps indipendentemente dalle altre.

Sostituendo anche gli hub intermedi con degli switch la situazione migliora ulteriormente perché avremmo solo collegamenti tra i singoli computer e gli switch. Una caratteristica importante infatti è la banda aggregata dello switch.

Switch

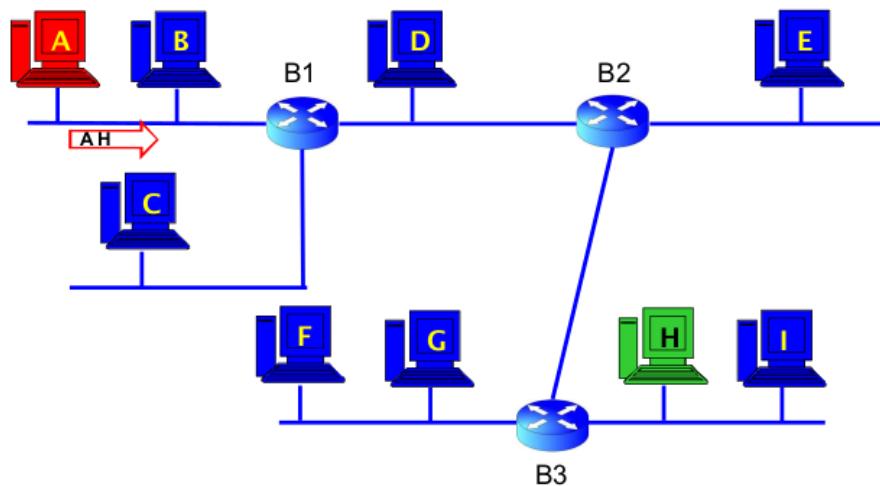


Indirizzo	Interfaccia	Tempo
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
....

Il compito dello switch è quello di analizzare le intestazioni delle frame e trovare le strade migliore per raggiungere la destinazione, quindi prepara al suo interno un percorso dedicato per un'eventuale comunicazione tra due particolari macchine, senza influenzare le altre. Questo significa che, se per ogni comunicazione si può raggiungere un massimo di 10 Mbps, allora la comunicazione tra le macchine più a sinistra andrà a 10 Mbps, mentre la comunicazione tra le macchine al centro andrà a 5 Mbps (si dividono la banda). In totale si avrà quindi una banda aggregata utilizzata di 20 Mbps.

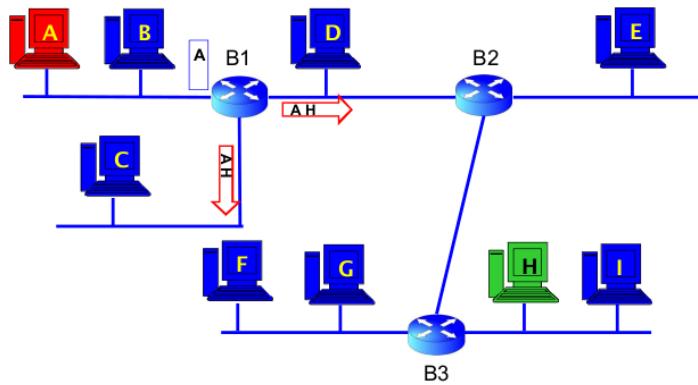
In un hub con 64 porte, quindi 32 possibili connessioni al massimo, dobbiamo avere 320 Mbit di banda aggregata.

Lo switch ha il compito di inoltrare il più possibile verso la destinazione corretta: ci riesce con una Tabella di Inoltre, che ha 3 voci: indirizzo, interfaccia e quando è stata vista l'ultimo info.

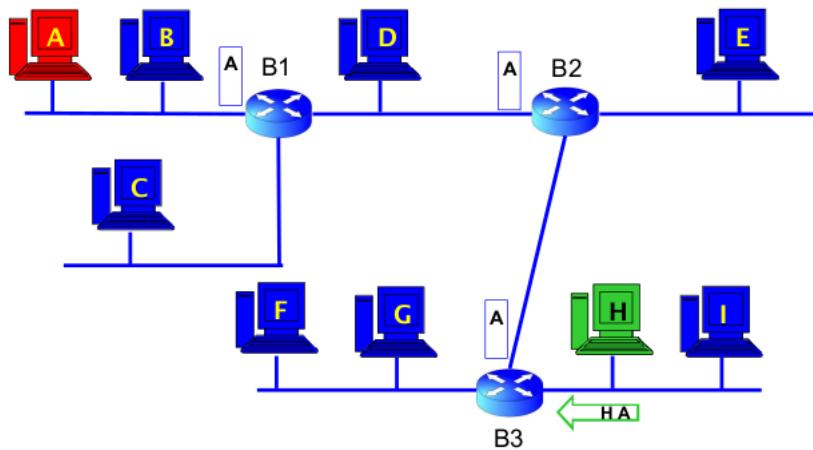


La macchina A vuole parlare con la macchina H, e prepara una frame dove mette gli indirizzi corretti, siamo a livello DLL. Viene inserita la frame sul bus e tutti ascoltano la frame. B la ignora.

B1, B2 e B3 sono switch. Cosa fa B1? Gli switch sono dispositivi Plug and Play, quindi non sa ancora cosa fare, e rimanda la frame su tutte le sue uscite. Nel frattempo prepara una sua tabellina dove si segna che la macchina A si trova sulla LAN a sinistra.



Su C non trova alcun riscontro di H: spazio e tempo perso. Prosegue, D se ne frega, B2 fa la stessa cosa di B1, ma c'è una cosa da notare: si segna di aver visto A sulla LAN alla sua sinistra, ma non sa se si trova esattamente sulla sua LAN o bisogna passare da qualche altro switch.



Infine, H risponde e a questo punto il pacchetto non va più in broadcast. A questo punto tutti e 3 i bridge sanno dove si trova la macchina A, quindi il pacchetto di risposta va direttamente sulla linea corretta.

Una volta arrivato, tutti gli switch sanno dove si trovano A e H. Da ora in poi per le successive comunicazioni non vengono coinvolti i tratti che non servono, quindi ci può essere altro traffico che non viene disturbato dalle connessioni tra A e H.

Questo è il comportamento intelligente degli switch: il collegamento diretto tra due interfacce. La dimensione della tabella di inoltro è infatti un'altra caratteristica importante da considerare.

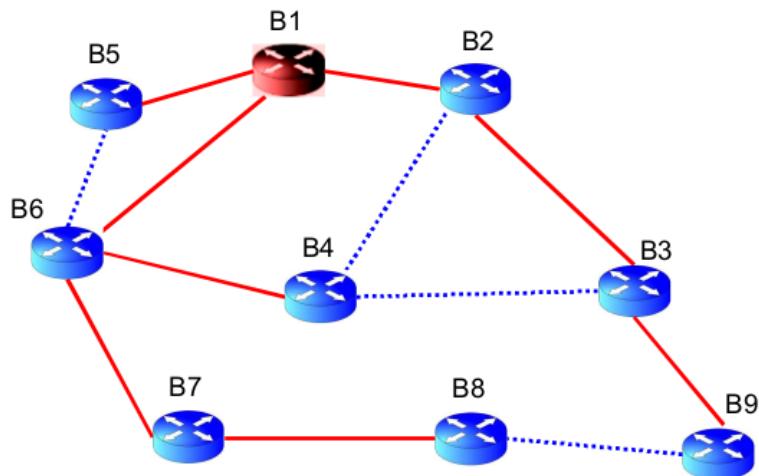
Il TTL viene messo perché le interfacce potrebbero spostarsi.

Problemi / Soluzioni

Non posso realizzare grafi, cioè niente collegamento ridondanti, cosa che però mi conviene in caso di interruzione di un dato collegamento. Devo evitare di creare cicli.

Uno dei modi è lo spanning tree: Da un grafo con cicli genera un albero di copertura con alcuni percorsi tagliati (non è detto che sia l'albero minimo di copertura, per quello dovremmo avere pesi e direzioni).

Un modo molto semplice è quello di cercare di eleggere un nodo radice nel mio schema: ad esempio uno switch con un identificativo più alto o più basso (anche un MAC Address).



Per esempio, B1 sfida B5 a chi ha il numero più elevato, dopo che B1 vince B5 conosce anche il valore di B1 e sa che lui non sarà radice, B5 sfidando B6 potrà scoprire che ha un valore più elevato rispetto a B1, e quindi può dire "ok, non è B1 ma B6".

Propagando questo schema di elezione a tutta la rete, tutti sapranno che B1 è il numero migliore e tutti gli altri recedono da questa operazione.

I nodi collegati direttamente alla radice saranno i nodi di primo livello dell'albero (e per esempio, B5 e B6 non comunicheranno direttamente tra di loro, ma passeranno per B1).

I nodi di secondo livello saranno B4 e B3, raggiungibili da B6, B2 rispettivamente. Non comunicano direttamente. Etc etc.

Differenze tra Repeater, Hub, Bridge, Switch

Repeater: Un amplificatore che serviva per connettere nelle reti con cavo coassiale ripetere il segnale da una tratta all'altra. (Ad esempio in 10Base-5 permetteva di collegare 5 tratte con 4 ripetitori intermedi). Ripete solo il segnale da una parte verso l'altra, con errori e collisioni. Dovrebbe avere solo due porte.

Hub: Ha lo stesso funzionamento del repeater, è sempre un dispositivo a livello fisico. La differenza è che Hub ha una struttura tipicamente a stella. Ha più di due porte. Entrambi non sono più utilizzati, non hanno più mercato.

Bridge: Trasforma le frame da un formato di LAN a un altro formato di LAN. Però considerato che ormai Ethernet ha preso piede, di fatto quello che si ha è lo Switch.

Switch: Equivalente del bridge per Ethernet, multiporta, si trovano switch con 5-8 porte, o 12-48. È un dispositivo intelligente con CPU a livello DLL come il bridge, ha delle memorie in ingresso e in uscita dove accodare pacchetti durante la ricezione o la trasmissione, in attesa che la linea sia libera.

Router: Sono a livello superiore (di Rete), interpretano il significato di pacchetti IP.

Lezione 20 Parte 1 - 25/05/2020

Recap

Ci sono dispositivi di livello 1: (Fisico, repeater e hub. Hub non esistono più perché non conviene produrli), di livello 2 (DLL, Bridge e Switch, servono per interconnettere LAN differenti effettuando se necessario conversioni di protocollo [Bridge]. Gli switch sono equivalenti di Bridge solo per Ethernet, servono per effettuare l'instradamento delle frame in base a meccanismi di autoapprendimento / plug and play).

Una grossa differenza tra Switch e Router è data dal fatto che normalmente viene installato senza necessità di configurazione, funziona ed effettua il routing basandosi sulle info ricavate dalla rete, con lo scambio iniziale delle frame.

Il router invece deve essere configurato e lavora a livello 3 (di Rete, IP).

Indirizzamento flat vs gerarchico

Un'altra differenza da sottolineare è lo schema di indirizzamento di base che si usa.

Abbiamo instradamento a livello DLL e instradamento a livello di Rete.

A livello DLL gli indirizzi hanno un formato 'flat', cioè senza nessuna logica di distribuzione di indirizzi nella rete. Il MAC Address infatti viene scelto dalla fabbrica di produzione della scheda di rete. È proprio l'indirizzo flat che permette ai dispositivi di essere collegati in qualsiasi punto della rete, proprio perché l'indirizzo stesso non contiene info di instradamento.

Invece, gli indirizzi a livello di rete si dicono 'gerarchici' proprio perché in base al valore è possibile ricavare dove si trova la destinazione e quindi seguire un certo percorso.

L'indirizzo IP è fatto da Rete, sottorete e host (3 campi principalmente).

Rete ci fa trovare il primo tipo di routing, fino ad arrivare fino alla struttura di destinazione, dove poi si usa la parte di sottorete e infine al singolo host (che torna ad essere flat).

Paradossalmente on IPv4 il sistema flat è molto più ampio del sistema gerarchico IPv4, come spazio di indirizzamento.

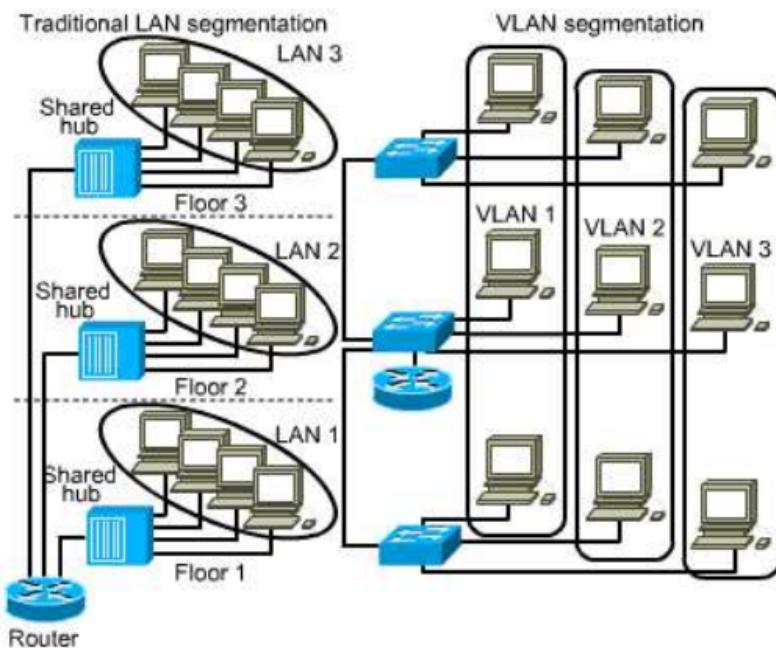
Il routing di internet funziona perché il singolo router non può conoscere tutti gli indirizzi di Internet, ma deve conoscere solo una parte o qualcuno a cui delegare l'operazione. L'indirizzamento flat invece è comodo perché non ci sono regole di posizionamento, quindi l'indirizzo è fisso nell'host ma ha il forte limite della dimensione dello schema di indirizzamento, ovvero una LAN non la possiamo realizzare con milioni di host, perché i Bridge e gli Switch imparano in base al traffico che c'è, ma inizialmente, quando non hanno informazioni, imparano mandando in broadcast tutto il traffico. Quindi con troppi host si perdono i vantaggi della struttura a switch della rete Ethernet, struttura che limita il traffico alle zone realmente interessate alla comunicazione.

Normalmente uno switch ha una tabella di inoltro di 1000-4000 indirizzi possibili, non più di tanto. Non serve farla più grande, dato che viene esaminata in ordine. Questo è anche un altro motivo per cui una LAN non ha un numero di host abnorme.

Se abbiamo un numero elevato di host (ben oltre i 1000-4000) allora conviene coinvolgere anche il livello di Rete.

Il tutto va fatto realizzando una struttura logica corretta, cablaggi compresi, ovvero cercando di capire come devono essere raggruppate le macchine per tipologia di utente. Questa è stata la base per le VLAN.

VLAN



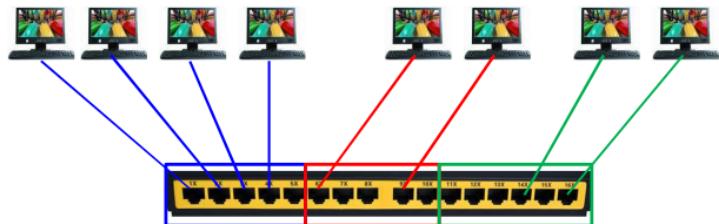
A sinistra si trova una struttura tradizionale semplice, dove ci sono 3 piani e quindi 3 LAN, ogni LAN provvista di un'interfaccia verso il router dell'edificio.

L'idea è che in ogni piano si faccia qualcosa di diverso, per cui la maggior parte del traffico è orizzontale, cioè coinvolge le macchine dello stesso piano. C'è però un tipo di traffico verticale, che va verso il router, che può essere reindirizzato verso altri piani. La struttura logica della divisione per "tipo di lavoro" è comoda perché ovviamente quando si vanno a distribuire gli spazi fisici per le persone si mettono vicine persone che hanno lo stesso tipo di lavoro.

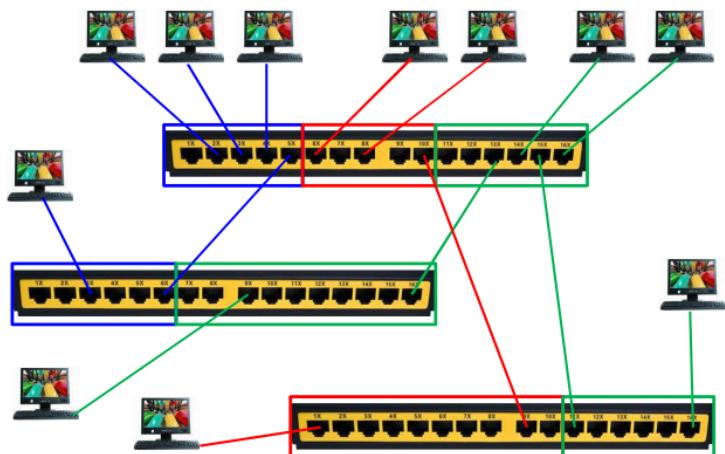
Le VLAN invece permettono di separare il posizionamento fisico della macchina / persona dal posizionamento fisico della LAN di connessione.

Cioè possiamo avere persone che si occupano dello stesso argomento su tre piani separati su tre LAN fisiche differenti, ma con una LAN virtuale che li accomuna; LAN che fa sì che il traffico broadcast sia principalmente limitato alla struttura verticale che coinvolge connessioni in piani separati (a destra).

Bisogna avere uno switch apposito innanzitutto:



Possiamo immaginare come se lo switch venisse configurato (manualmente) in maniera tale che alcune porte appartengono ad una LAN, altre porte ad un'altra LAN e le rimanenti ad una terza LAN, per realizzare strutture di questo tipo:

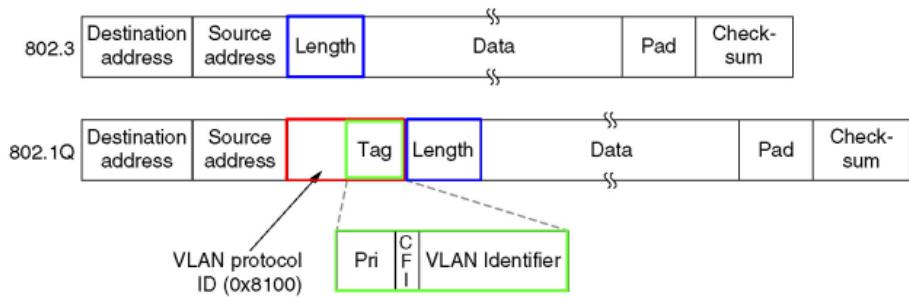


In pratica sono 3 switch presenti in posizioni nettamente separate, che sono collegati tra di loro (blu con blu, rosso con rosso, verde con verde). Non c'è necessità di collegarli tutti tra di loro perché possono anche raggiungersi indirettamente.

I colori differenti isolano le LAN dal resto del mondo.

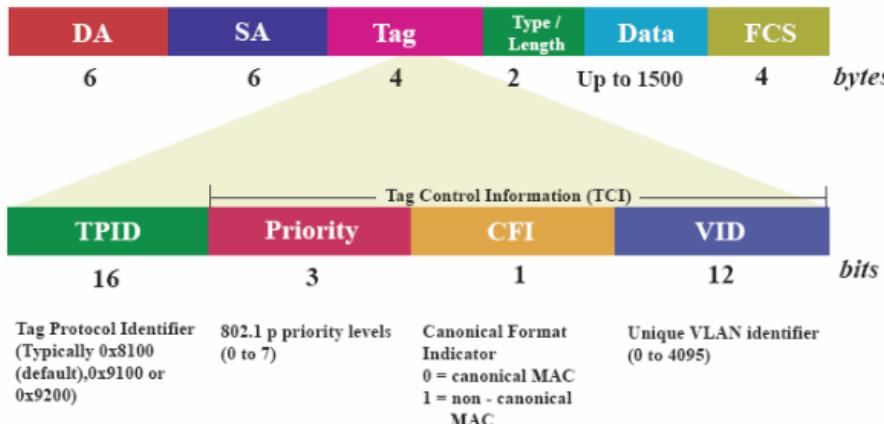
La cosa interessante è che se non c'è un collegamento diretto tra colori diversi, con un cavo vero e proprio o con un meccanismo software interno allo switch (se lo switch non è solo di livello 2), colori diversi sono totalmente irraggiungibili tra di loro. C'è una netta separazione tra i vari colori. Questa VLAN si dice **untagged**, cioè sprovvista di un tag identificativo, è il modo più semplice per realizzare una VLAN. Spreca di più perché almeno una porta dello switch va dedicata come se fosse un “uplink verso il mondo esterno” (o anche di più, essenzialmente quasi una per VLAN a meno di connessioni indirette).

Il protocollo 802.1Q, che permette una variazione al formato delle frame Ethernet, per creare VLAN **tagged**.



Consente di utilizzare gli stessi link fisici per VLAN differenti, inserendo 4 Byte nel frame Ethernet, per poter associare ogni frame ad una VLAN.

La prima coppia di Byte è un valore esadecimale che identifica il protocollo della VLAN.
La seconda coppia prevede altri tre campi:



Priority: (3 bit) Livelli di priorità, da 0 a 7

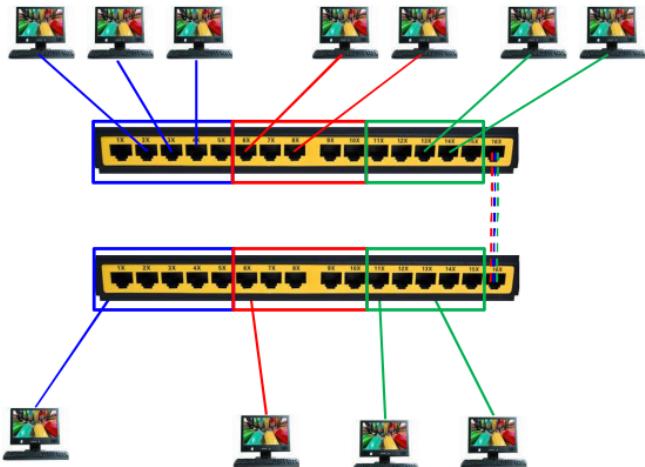
CFI: (1 bit) Canonical Format Indicator

0 = MAC canonico, 1 = MAC non-canonico

VlanID: (12 bit) Identificatore univoco per la VLAN, fino a 4095.

Ovviamente il sistema deve sapere a priori se sta trattando frame Ethernet standard 802.3 o 802.1Q. Noi possiamo dire alla scheda di rete se le frame che riceve su una porta usano uno standard o meno.

Con la Tagged VLAN realizziamo quindi una rete di questo tipo:



Rimane sempre una porta che viene usata come uplink per collegare gli switch tra di loro, ma a questo punto possiamo dire che la porta di uplink non usa 802.3 ma usa il formato esteso VLAN 802.1Q. Potendo realizzare questo schema quello che succede è che possiamo dire a quale VLAN appartengono certe porte, e il traffico che deve andare in broadcast può passare dal canale uplink per tutte e tre le tipologie di VLAN e andare negli switch opportuni. Ovviamente permette di ridurre i collegamenti fisici tra gli switch, e di configurare velocemente le porte.

Scopo delle VLAN:

- Risparmio: riutilizzo le linee e gli apparati preesistenti;
- Flessibilità: facile spostamento fisico degli utenti;
- Prestazioni: il traffico broadcast viene confinato
- Sicurezza: gli utenti di VLAN differenti non vedono i reciproci frame dati.

Physical Layer (3½ lezioni+chiarimenti sparsi)

Lezione 20 Parte 2 - 25/05/2020

Il livello fisico si occupa di trasportare segnali (e fin qui ci siamo). Per poter trasportare delle informazioni da un posto all'altro è necessario avere:

- Un **canale** trasmisivo in grado di garantire il trasporto di segnali;
- Un **generatore** di segnali;
- Un **rilevatore** di segnali.

Il canale riesce a trasmettere i segnali perché deforma una sua caratteristica fisica. Per esempio, se diamo uno scossone a una molla questa si deforma (per propagazione delle onde in questo) in avanti finché arriva a destinazione. Se a destinazione mettiamo qualcosa che percepisce la deformazione allora abbiamo avuto una trasmissione. Stesso discorso si può fare per una corda.

La cosa più semplice che possiamo usare sono i mezzi elettrici, e permettono velocemente di passare da bit a segnale e da segnale a bit; oppure le onde elettromagnetiche per i mezzi wireless; o i mezzi ottici che consentono canali con larghezza di banda molto elevata.

La cosa che lega tutti questi tre mezzi tra di loro è che utilizzano tutte onde elettromagnetiche a bassa frequenza, alta frequenza o altissima frequenza rispettivamente.

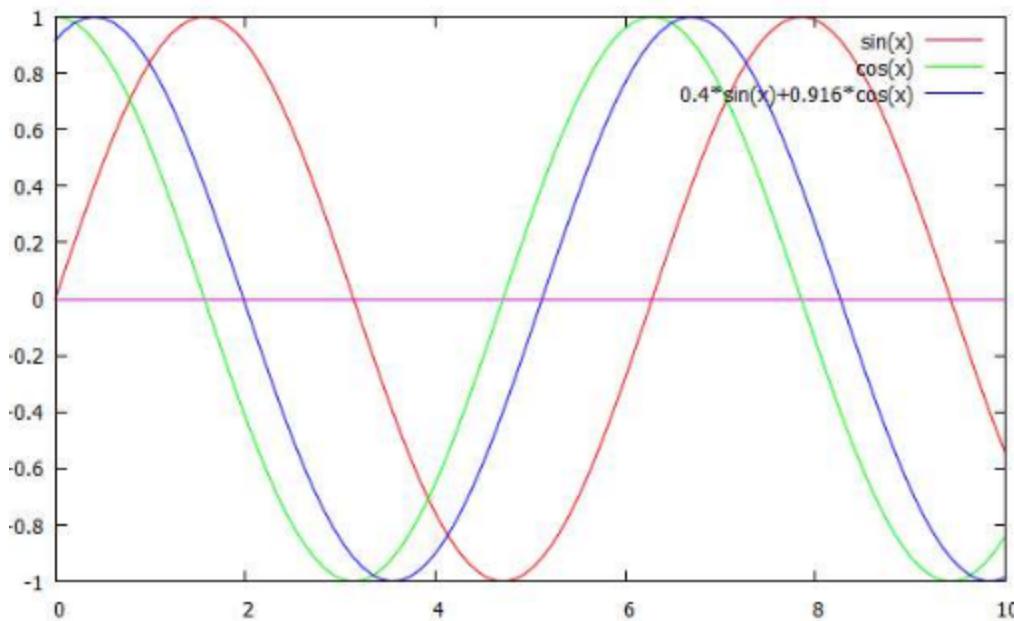
Analisi di Fourier

Un qualunque segnale di periodo finito può essere rappresentato per mezzo di una sommatoria infinita di onde sinusoidali di frequenza ed ampiezza opportune.

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft)$$

f= 1/T è la frequenza fondamentale del segnale.

In altre parole, avendo un segnale periodico che si ripete nel tempo all'infinito lo posso rappresentare in maniera matematica, dove la frequenza è pari all'inverso del periodo del segnale.



In rosso abbiamo un segnale periodico, e Fourier ci dice che possiamo rappresentarlo come una costante + una sommatoria indefinita (e non infinita) di sinusoide + una sommatoria di cosinusoidi.

Nel primo caso, $\sin(x)$ è essenzialmente una sinusoide rappresentata da una sinusoide, perché la costante sparisce, la sommatoria di sinusoide fa 1 e la sommatoria di cosinusoide fa 0. Analogamente il caso verde.

La funzione in blu invece la posso vedere come una sinusoide più una cosinusoide, ma le devo pesare diversamente: la sinusoide la peso 0.4, la cosinusoide la rimpicciolisco di 0.916.

Se faccio la somma di queste due ottengo la linea blu.

I pesi sono tali che i quadrati di sinusoide e cosinusoide fanno 1.

I coefficienti a_n e b_n servono a pesare le varie componenti. Si ricavano dalle seguenti espressioni:

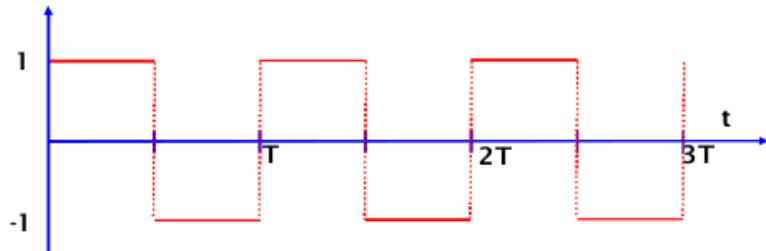
$$a_n = \frac{2}{T} \int_0^T g(t) \sin(2\pi n f t) dt$$

$$b_n = \frac{2}{T} \int_0^T g(t) \cos(2\pi n f t) dt$$

$$c = \frac{2}{T} \int_0^T g(t) dt$$

Risolvendo questi integrali troviamo i coefficienti.

Esempio: Onde quadre:



$$g(t) = \begin{cases} 1 & \text{se } 0 + nT < t < \frac{1+2n}{2}T \\ -1 & \text{se } \frac{1+2n}{2}T < t < (1+n)T \end{cases}$$

NB: Il tratto verticale tratteggiato matematicamente non esiste.

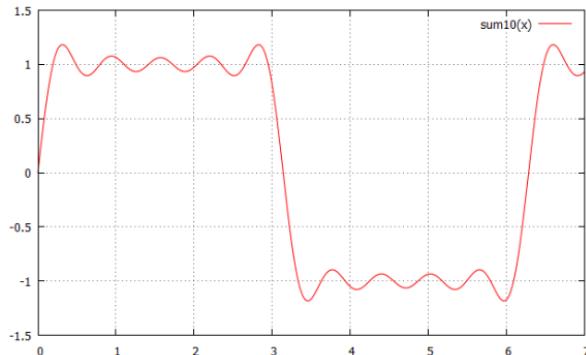
La cosa interessante è che applicando le formule per ottenere i coefficienti si scopre che b_n e c diventano 0, mentre a_n diventa 0 per ogni n pari, mentre diventa $a_n = 4/n\pi$ per ogni n dispari.

Cioè l'onda quadra può essere scritta in questo modo:

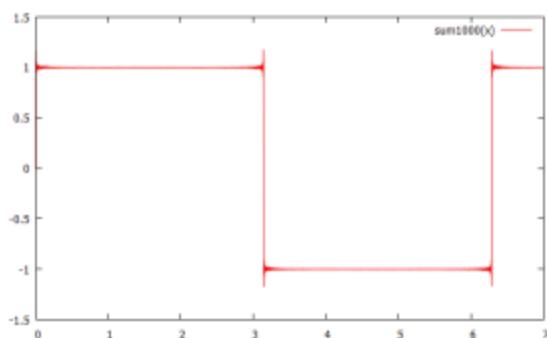
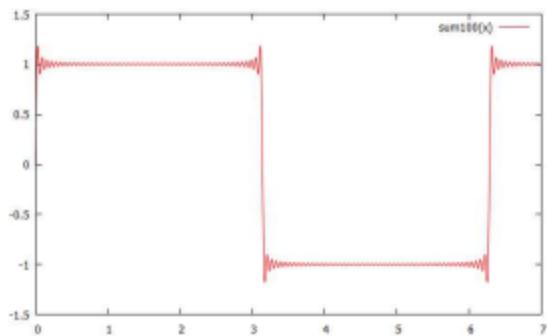
$$g(t) = \frac{4}{\pi} \sin(2\pi t) + \frac{4}{3\pi} \sin(6\pi t) + \frac{4}{5\pi} \sin(10\pi t) \dots$$

In ordine abbiamo l'onda fondamentale, poi c'è la terza armonica, la quinta, etc... I contributi di ogni armonica diventano sempre più piccoli perché il denominatore cresce.

Sommendo le prime dieci armoniche otteniamo:



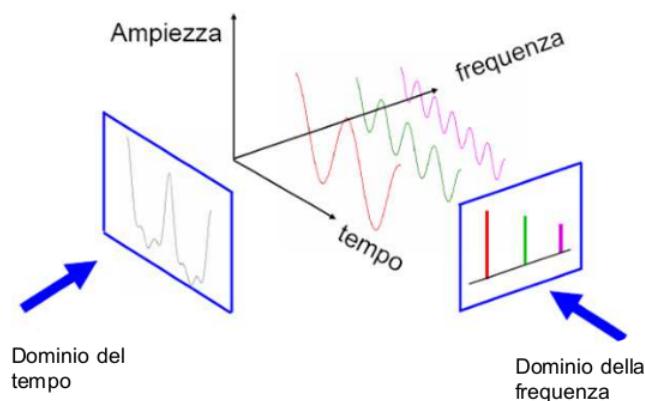
Mentre invece sommando le prime 100 e 1000:



Man mano che aggiungo le armoniche, nonostante la funzione non sia discontinua come l'onda quadra, ottengo un'ottima approssimazione dell'onda quadra (seppur continua e derivabile).

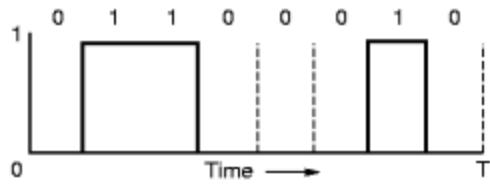
In realtà però, la cosa interessante dell'Analisi di Fourier la posso vedere come onde sinusoidali a frequenze differenti sommate in maniera opportune fra di loro.

Cioè, quello che detto Fourier è che la $g(t)$ si trova nel Dominio del tempo, la scompongo in frequenze differenti, e in maniera equivalente vedo questa funzione nel dominio delle frequenze (il peso di ogni componente di frequenza). Cioè vedo differentemente la stessa informazione: al variare del tempo o al variare delle frequenze. (**IMPORTANTE**)

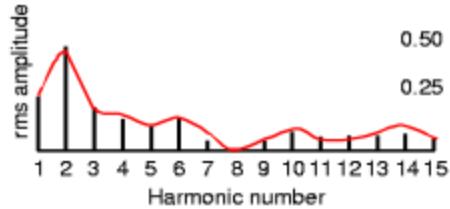


Detto in altri termini, o scrivo $g(t)$ con l'espressione dell'onda quadra, o dico: "data la frequenza base $1/T$ le varie armoniche hanno altezza $4/n\pi$ ", che sono infinite perché n va a $+\infty$, ma anche $g(t)$ lo fa quindi non cambia nulla, però così ho un modo diverso di vedere la stessa informazione, che mi permette di fare qualche considerazione più particolare.

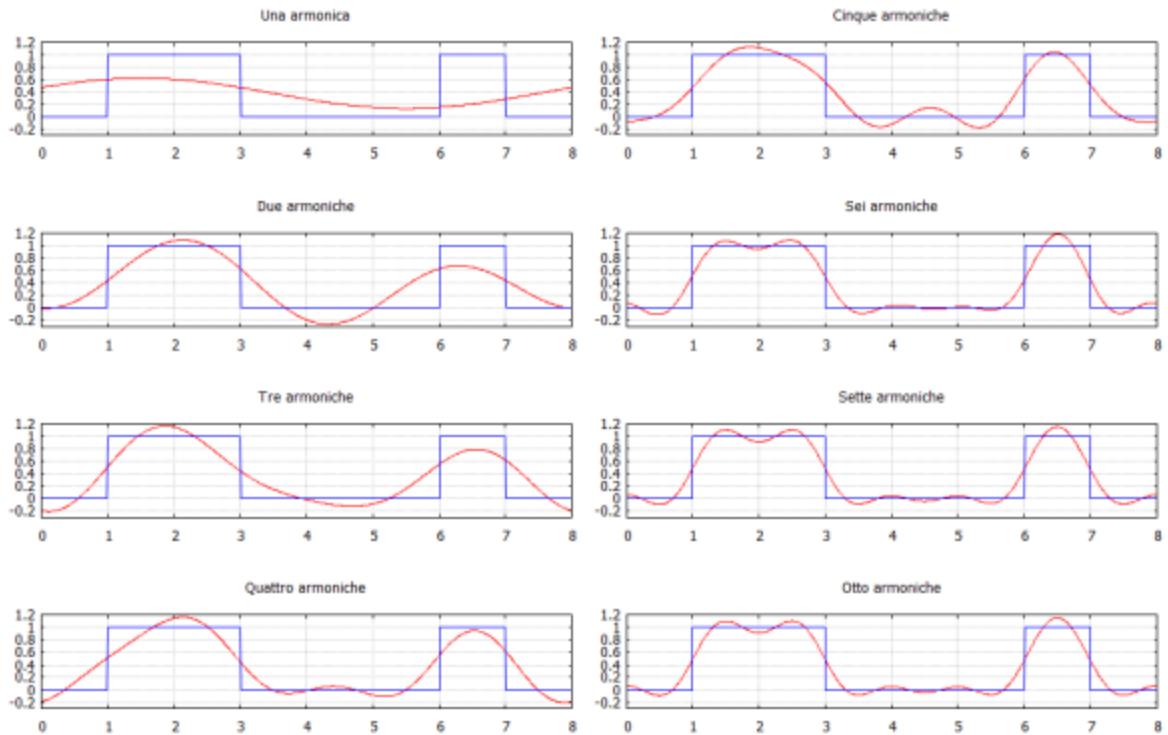
Cerchiamo ora di vedere per i nostri scopi come poter utilizzare questo risultato. Prendiamo un segnale simile a un'onda quadra, e supponiamo sia un segnale da trasmettere. Supponiamo inoltre che la parte che vogliamo trasmettere sia periodica ed infinita per rispettare l'ipotesi di Fourier.



Applicando le formule per ottenere i coefficienti si può ricavare lo spettro del segnale, ovvero l'influenza che le varie armoniche hanno nel segnale base.

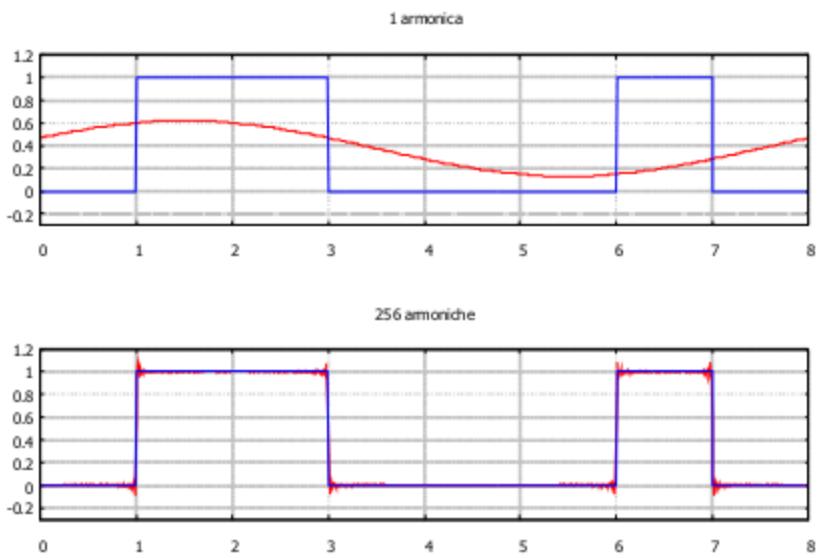


La cosa interessante è che la prima armonica è più piccola della seconda, e poi le altre hanno vanno decrescendo.



In blu abbiamo il segnale che ci interessa, mentre in rosso le armoniche.

Infine otteniamo:



Se stiamo trasferendo il tutto a 8 Mbps vuol dire che 1 bit ha una durata temporale di $\frac{1}{8 \cdot 10^6}$ secondi. Cioè un ottavo di microsecondo.

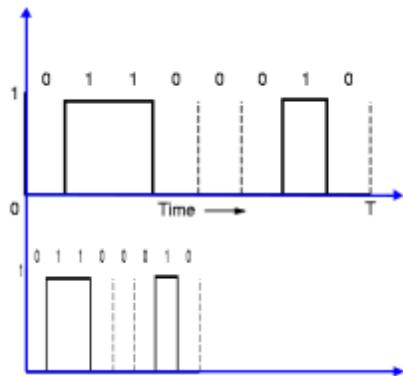
L'armonica fondamentale è della frequenza di 1 MHz, un milione di volte al secondo. (cioè la prima). La 256^a armonica è 256 volte la frequenza base, cioè 256 MHz.

I canali trasmissivi si suddividono in:

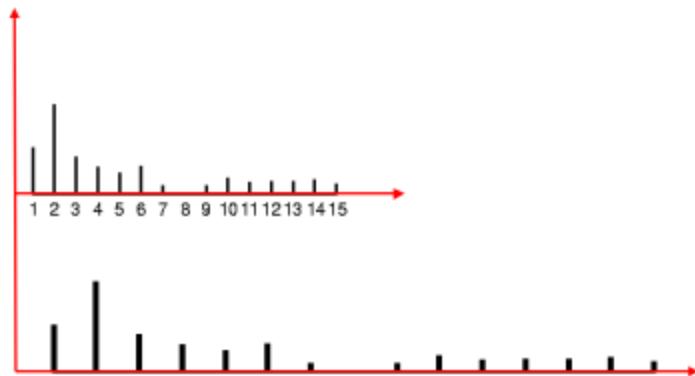
- Canali Perfetti: non causano distorsioni o ritardi nella propagazione dei segnali.
- Canali Ideali: causano solo un ritardo costante nella propagazione
- Canali reali: causano attenuazioni e ritardi, in funzione della frequenza dei segnali. ("le schifezze più immonde")

Larghezza di banda di un canale

Tornando all'esempio di prima, la nostra trasmissione, nel dominio del tempo:



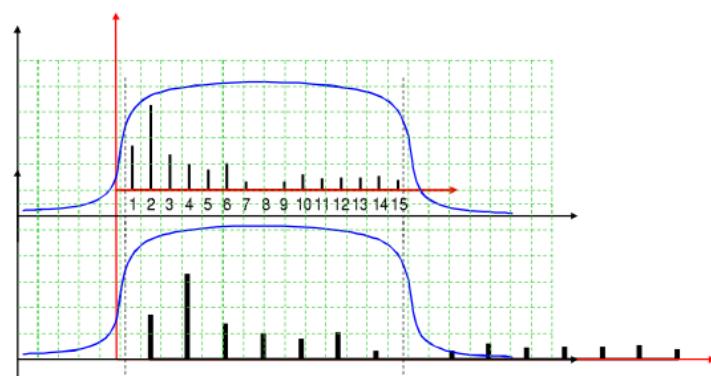
Si rappresenta in questo modo nel dominio delle frequenze:



Prendiamo e raddoppiamo il bitrate di comunicazione. Se andiamo a 16 Mbps i bit diventano la metà, perché nel tempo in cui io ne spedivo 8 ne devo spedire 16.

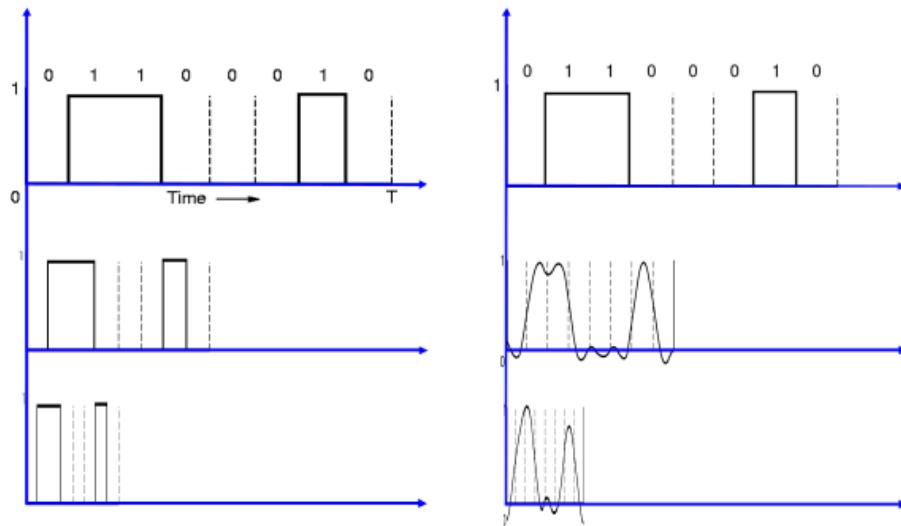
Siccome il tempo è la metà, le frequenze raddoppiano. ($f=1/T$).

Però le singole frequenze devono passare nel canale:



Metà finiscono fuori, il canale non le fa passare (le taglia, azzera).

Quindi, se a una data velocità arriva qualcosa a destinazione, se dimezzo il tempo cambia quello che arriva. Addirittura sfociamo nei valori negativi:



Man mano che aumento il bitrate, il canale comincia ad inserire delle modifiche perché toglie alcuni pezzi in ciò che viene trasmesso, fino ad avere qualcosa che è difficilmente interpretabile come il segnale originale.

Con l'Analisi di Fourier quindi scopriamo cosa potrebbe arrivare a destinazione.

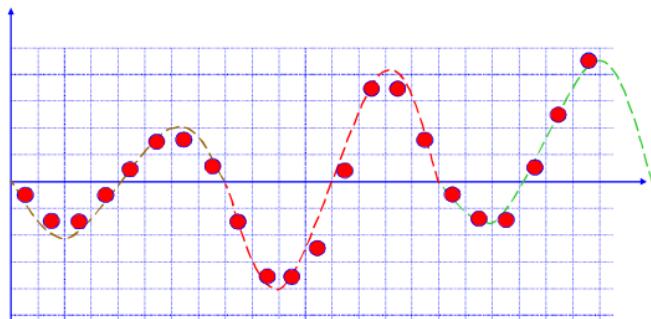
I problemi sono quindi:

- A destinazione riesco ancora a capire le schifezze che arrivano?
- Attenuazioni e malmenazioni del segnale sono causate anche dal trasmettitore e dal ricevitore, non solo dal canale, essendo tutti e tre imperfetti / limitati.

Quantizzazione

Prendiamo un segnale qualunque, mettiamoci sopra una griglia quadrettata e rappresentiamo solo i punti in cui il segnale è più forte.

Quando io devo trasmettere un segnale, un'informazione quello che fa il trasmettitore è decidere a quale livello corrisponde l'informazione che devo spedire.



Ad esempio, nei monitor i colori rappresentabili sono discreti. Un pixel può avere un solo valore, e un valore RGB è limitato da tre terne con valori da 0 a 255, niente nel mezzo e niente oltre.

Non è solo il trasmettitore che quantizza, ma anche il ricevitore, per quanto riesce a capire.

Teorema di Nyquist

Nyquist ha elaborato un'espressione che lega la larghezza di banda di un segnale con la quantità di informazioni trasportabili:

$$\text{Massimo Bitrate} = 2 \cdot H \cdot \log_2 V \text{ bit/s}$$

Dove H è la larghezza di banda del segnale; (quantizz. orizzontale)

V è il numero di livelli differenti nel segnale. (quantizz. verticale)

In altre parole, data la larghezza di banda massima del segnale (cioè la distanza tra quadratini fra di loro) e dato il numero di livelli discriminabili in verticale, allora il bitrate massimo del segnale è dato dalla formula sopra. Il numero di campioni in verticali dipende dalla qualità del trasmettitore.

Se siamo limitati dalla larghezza di banda, possiamo aumentare il numero di livelli, cioè fare più fitta la griglia in verticale, in modo tale da aumentare il bitrate.

Essenzialmente è ciò che è stato fatto in Gigabit Ethernet, cioè quando si è passato dal sistema a 3 livelli al sistema a 5 livelli, per portare 2 bit.

Non possiamo però aggiungere livelli all'infinito per poter aumentare il bitrate però. Il perché è spiegato nella prossima lezione.

NB: Nyquist non pone limiti al bitrate trasportabile da un segnale.

Lezione 21 - 27/05/2020

Recap

Il canale non fa passare tutte le frequenze, una parte le taglia, per cui il segnale può arrivare deformato a destinazione. Inoltre se il generatore e il ricevitore non sono ben in accordo con il canale una parte dell'energia **può riflettersi** e andare a rovinare la parte successiva del segnale, non capendoci più nulla.

Per questo secondo punto esistono delle soluzioni: nei cavi coassiali si mettono i terminatori per eliminare tutta l'energia che arriva alle estremità e limitare le riflessioni. Più basso è lo "smorzamento" ("attrito", assorbimento dell'energia da parte dal cavo causata da distanza, resistenza, etc etc...) più è alta la riflessione del segnale. Con bassissimo smorzamento le oscillazioni riflesse iniziano ad auto amplificarsi.

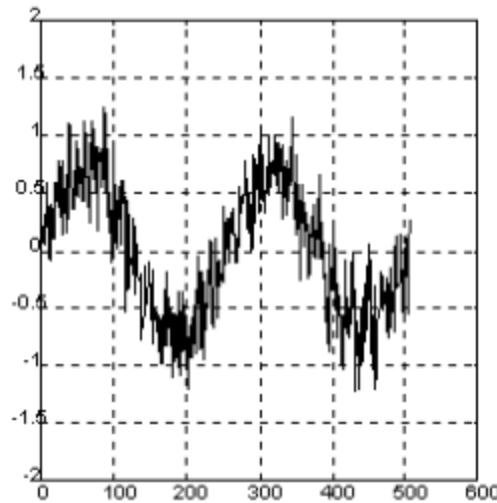
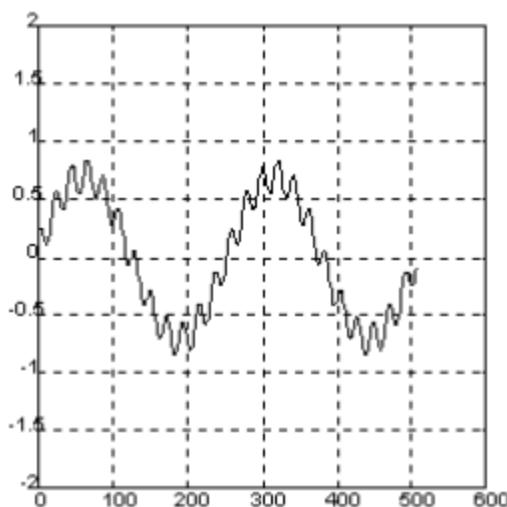
Il problema della Quantizzazione esiste in verticale e in orizzontale, cioè, il trasmettitore ed il ricevitore, che non sono perfetti, possono generare segnali per quanti successivi e ricevere per quanti successivi.

Poi c'è una quantizzazione in orizzontale, perché entrambe le macchine non possono istantaneamente cambiare il proprio stato ma hanno bisogno di un certo tempo per compiere questa operazione. Il massimo bitrate viene fuori dalla formula di Nyquist.

Non posso però utilizzare un cavo Cat3 (cioè con la stessa larghezza di banda H) aumentando all'infinito i livelli (V) perché esiste il rumore.

Rumore

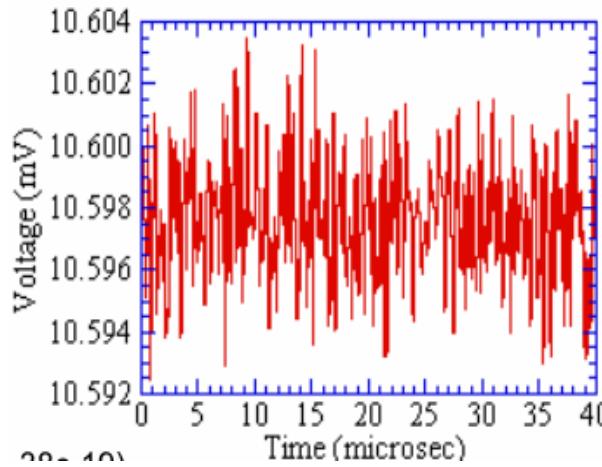
È un segnale che si sovrappone alla comunicazione, disturbandola.



Le forme piccole sono confrontabili col rumore come ampiezza, mentre le forme più grandi riusciamo ancora a definirle nonostante il danno causato dal rumore. Per superare il rumore dovremmo inserire molta più energia rispetto al valore del rumore. Come lo quantifichiamo e cosa è?

Esistono vari tipi di rumore:

Rumore Termico



$$P = 4K T Bw$$

$$V = \sqrt{4 K T R Bw}$$

K: costante di Boltzman's (1.38e-19)

T : temperatura in Kelvin.

R: resistenza in ohm.

È dovuto al fatto che non siamo a 0K di temperatura, e quindi le particelle degli atomi sono eccitate dal calore e non ferme.

A qualunque temperatura superiore allo 0K gli atomi oscillano all'interno della struttura in cui si trovano, e generano disturbi elettromagnetici. Sono ovviamente minimi, ma in alcune situazioni diventano fastidiosi.

Un cavo elettrico in rame che trasmette un segnale ad esempio soffre dell'interferenza del rumore termico.

Per superare l'interferenza dovuta al rumore termico dobbiamo superare le interferenze prodotte dalle oscillazioni totali.

La formula $P = 4K T Bw$ ci dice che la **potenza** del segnale dipende dalla temperatura T, da una costante K, che non ci interessa, ma soprattutto dalla frequenza che vogliamo andare a considerare.

Cioè, il rumore è a tutte le frequenze, ma in alcune di queste il rumore non ci disturba più di tanto.

Inoltre, dipende principalmente dalla lunghezza del cavo: Più è lungo un cavo, maggiore sarà la resistenza che mette nella comunicazione (in ohm).

Quindi, il disturbo causato dal rumore termico sarà proporzionale alla lunghezza del cavo.

Teoricamente potremmo usare un cavo più spesso per diminuire la resistenza, ma costano di più.

Interferenze elettromagnetiche

In un cavo, messo all'aria aperta, è un'antenna. Poco funzionale per i nostri scopi, perché assorbe le radiazioni che ci sono nell'etere e le trasforma in energia elettrica, che va a sovrapporsi all'energia già presente nel sistema.

Ogni cavo essenzialmente forma una spira, cioè una forma circolare che raccoglie energia dall'etere e forma un'antenna. Può essere ben definita nel caso di una forma chiusa, o non definita se il cavo è dritto. Nei cavi però vogliamo limitare questo assorbimento.

Cavo Coassiale



Un modo per limitare l'effetto antenna è quello di utilizzare un cavo coassiale, perché il conduttore interno è protetto dal connettore esterno (treccia) e far sì che non si può individuare una spira all'interno del cavo. Non essendo individuabile una spira, non c'è energia che riesce a passare e arrivare al conduttore interno.

In pratica, questo sistema qui non si comporta come antenna. Infatti viene addirittura usato nelle antenne proprio per trasportare l'energia raccolta dall'antenna per farla arrivare al televisore / amplificatore senza ulteriori disturbi. Questo è un ottimo conduttore per i segnali perché subisce poche interferenze elettromagnetiche dall'esterno. (Praticamente un sistema simile alla gabbia di Faraday).

È stato eliminato come cavo perché si dovevano usare i conduttori interni per realizzare connessioni, e questi conduttori hanno il brutto vizio di creare delle interruzioni del cavo e quindi delle riflessioni, e realizzare anche inaffidabilità del mezzo di comunicazione, nonostante la quasi totale assenza di disturbi elettromagnetici.

Doppino Intrecciato



(a)



(b)

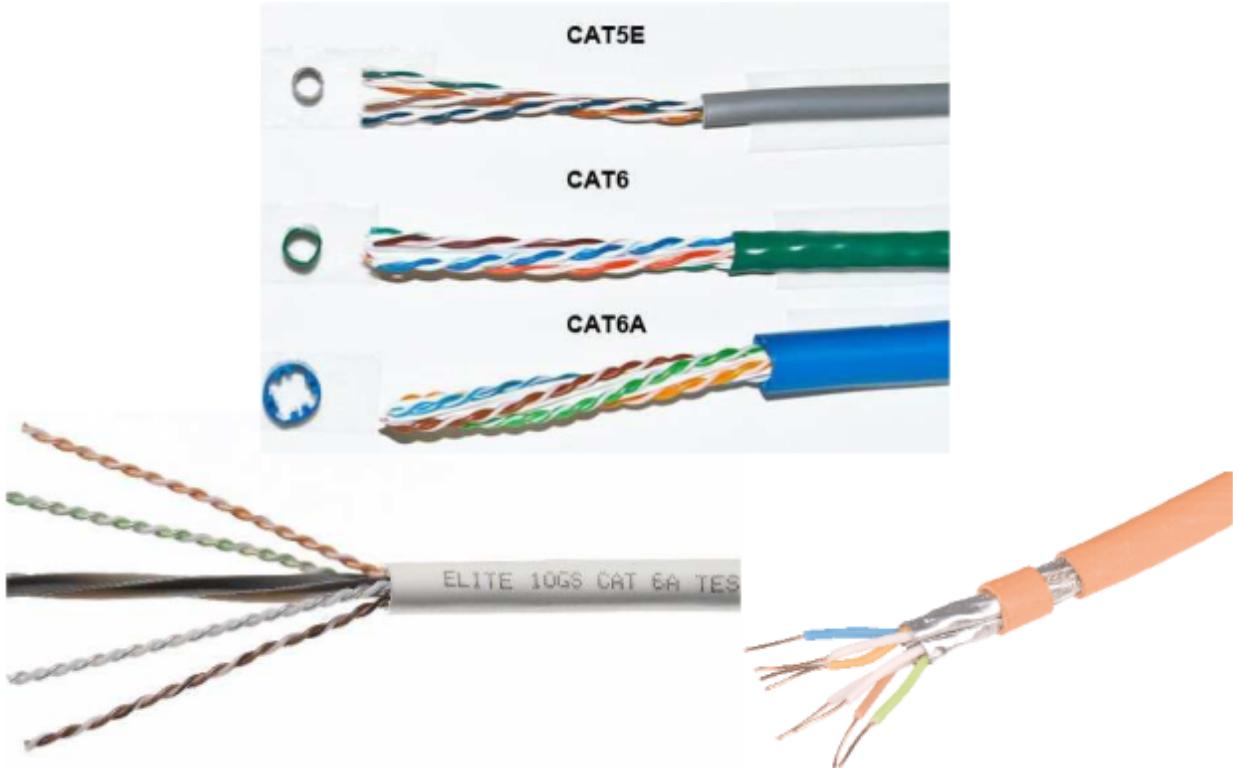
È costituito da una coppia di cavi elettrici intrecciati ad elica fra loro.

È una vera e propria spira, perché i due cavi intrecciati lasciano tra di loro un certo spazio vuoto, più o meno grande, dove appunto viene realizzata una spira e c'è una sensibilità alle interferenze elettromagnetiche.

Si viene a generare una corrente che, per esempio, possiamo immaginare che vada da destra verso sinistra nel cavo grigio, e da sinistra verso destra nel cavo nero. Quindi in pratica si annullano fra loro, anche se l'annullamento non è perfetto a causa della distanza tra le due spire (che nei cavi reali è infinitesimale, ma comunque non nullo).

L'intrecciamento fa sì che il doppino sia una pessima antenna, ma ovviamente per essere realmente pessima le spire devono essere regolari.

Riducendo la dimensione della spira e aumentando la regolarità degli intrecci si hanno cavi migliori (CAT5, etc...). C'è meno interferenza elettromagnetica e meno rumore.



Possiamo ancora migliorare, nel cavo in basso a sinistra (CAT 6A) ci sono 4 coppie, che hanno il difetto di passarsi le informazioni l'un l'altro, dato che sono vicine tra di loro. Oltre ad essere un'antenna ricevente diventa anche un'antenna trasmittente. C'è un fenomeno di passaggio dell'energia da una coppia all'altra.

Se aumentiamo la separazione fra le coppie diminuisce il passaggio di energia da una coppia all'altra.

Se poi tutte queste coppie le facciamo girare in maniera elicoidale ripetiamo il fenomeno dell'intreccio visto precedentemente.

Inoltre, possiamo aggiungere un rivestimento esterno anche a questi cavi, prendiamo le singole coppie e ci mettiamo un rivestimento attorno che limita ulteriormente le interferenze.

Quindi, con piccoli accorgimenti possiamo realizzare cavi sempre più sofisticati che permettono di avere interferenze elettromagnetiche sempre più limitate e quindi andare a velocità maggiore.

Teorema di Shannon

Nella pratica due fenomeni limitano questo tipo di trasmissione:

- Il numero di livelli discriminabili (al contrario di quanto dice Nyquist);
- Presenza di rumore (termico, interferenze...).

$$\text{Massimo Bitrate} = H * \log_2(1+S/N) \quad \text{bit/s}$$

(in un canale)

Dove H è la larghezza di banda del segnale;
 S è la potenza del segnale sul canale;
 N è la potenza del rumore sul canale;

Da questa espressione si deduce chiaramente che per ogni canale esiste un ben preciso limite fisico.

Per aumentare il bitrate si deve aumentare la potenza di segnale sul canale, o diminuire il rumore.

Infatti, se il rumore è basso, possiamo anche trasmettere con poca energia di segnale. Se tutte e tre le variabili sono fisse però, allora non è possibile andare oltre il massimo bitrate ottenuto calcolando la formula.

Esempio: il canale telefonico

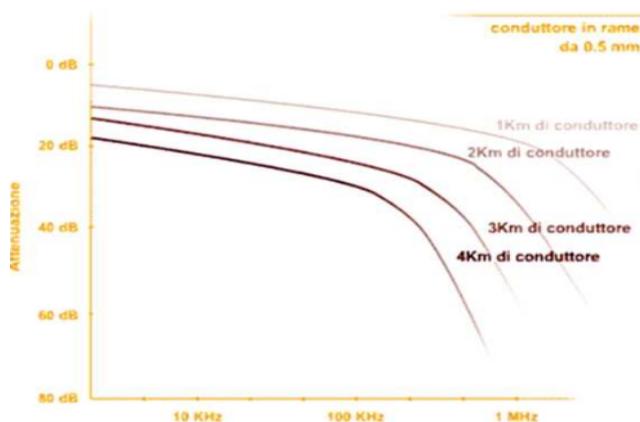
Un normale canale telefonico consente di utilizzare frequenze nel range di 30-3000 Hz, tutto il resto viene tagliato da appositi filtri.

Tipicamente il rapporto segnale rumore (S/N) è di 30 dB, considerando che i telefoni non erano collegati alla presa elettrica ma assorbivano l'energia dalla presa telefonica stessa.

Applicando la formula di Shannon il bitrate massimo è di 30000 bps.

Questo valore è un valore medio, perché dipende anche dalla distanza dalla centrale telefonica, e altri fattori ancora.

I modem migliori potevano arrivare anche a 52 kbps.



Riassumendo, possiamo dire che:

- Più alto è il bitrate, più ampia deve essere la banda del segnale (se il rapporto S/N è costante).
Oppure dobbiamo migliorare il rapporto segnale-rumore,
- Più lungo è il canale trasmissivo, maggiore è la potenza del rumore, minore è la capacità del segnale.

Shannon quindi dice esattamente l'opposto di Nyquist: Avere meno livelli rende i dati meno suscettibili al rumore, perché dato che ci sono meno livelli di quantizzazione verticale il rumore non riesce a “spostare”, o modificare, il valore quantizzato e quindi non riesce a rovinare i dati.

Quindi per Shannon non contano i numeri di livelli, ma solo il rapporto segnale-rumore e l'ampiezza, nient'altro. Con meno livelli abbassi il bitrate ma risulti più immune al rumore che alzandoli.

Lezione 22 - 29/05/2020

Recap

Rumore e interferenze limitano la quantità di bitrate che si può avere da un singolo canale. Questa limitazione è legata al fatto che ogni canale si comporta come un filtro, che taglia le basse e alte frequenze, e fa sì che il massimo bitrate realizzabile da un canale sia dato dalla formula del Teorema di Shannon.

Quindi, in generale più alto è il bitrate, maggiore dovrebbe essere la banda del segnale da utilizzare, oppure più lungo è il canale, maggiore saranno i vari disturbi.

Modulazione

Prendiamo come canale trasmittivo una fibra ottica, dove queste frequenze sono quelle delle onde elettromagnetiche che riescono a passare dentro la fibra ottica.

La fibra ottica è totalmente insensibile alle radio-frequenze in un dato range e diventa un oggetto totalmente opaco per quanto riguarda le frequenze alla luce visibile.

C'è una zona (la luce visibile) che riesce a passare abbastanza bene.

Se vogliamo trasmettere la voce umana, le frequenze della voce non rientrano nel range in cui il canale in fibra ottica si "comporta bene".

La Modulazione serve a far sì che il segnale che vogliamo trasmettere possa rientrare all'interno del range in cui il canale può trasmettere.

$$s(t) \quad p(t) = A \sin(\omega t + \varphi)$$

$$g(t) = A k_a s(t) \sin(\omega t + \varphi)$$

$$g(t) = A \sin((\omega + k_r s(t)) t + \varphi)$$

$$g(t) = A \sin(\omega t + k_p s(t) + \varphi)$$

Indicheremo con una funzione generica $s(t)$ il **segnale** da dover trasmettere. Il segnale varia nel tempo e avrà il suo spettro di frequenze (facendo lo sviluppo in serie o con la trasformata).

Consideriamo **un'onda sinusoidale** rappresentata da $p(t)$. (onda **portante**)

ωt è la frequenza del nostro segnale, e φ è la fase iniziale. La fase viene utilizzata per far sì che questa onda sinusoidale sia:

- per $\varphi=0$ un seno;
- per $\varphi=\pi/2$ un coseno;
- infine per φ compreso nei valori di $p(t)$ un'onda traslata orizzontalmente di un certo periodo (cioè come con la serie di Fourier.)

Quindi $p(t)$ si chiama segnale o onda portante perché deve trasmettere il segnale $s(t)$ che ci interessa.

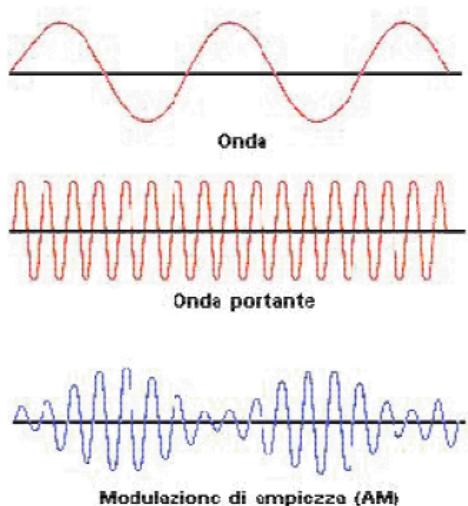
Combinando la $s(t)$ con la $p(t)$ abbiamo una funzione $g(t)$ che poi sarà quella inserita sul canale.

Ci sono tre possibilità di combinazione.

Nella portante abbiamo tre elementi che possono variare: la A , la ω e la φ , e quindi avremo tre modulazioni a seconda di dove inseriamo il segnale nella formula, moltiplicato per una costante k .

Quindi, in altre parole, il segnale fa variare l'ampiezza, la frequenza o la fase dell'onda portante.

Modulazione di Ampiezza (AM)



Come si può vedere, la risultante ha sempre una forma simile a una sinusoide, però è rappresentato da qualcosa ad altra frequenza.

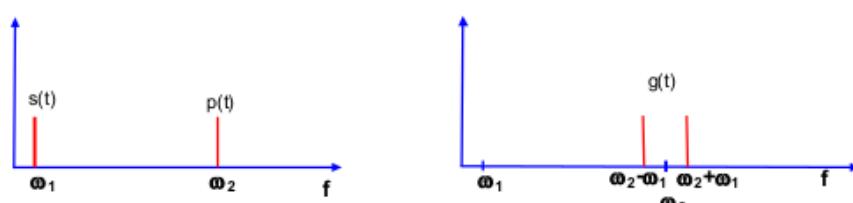
Ponendo l'ampiezza fissa ad 1, otteniamo:

$$s(t) = \sin(\omega_1 t)$$

$$p(t) = \sin(\omega_2 t)$$

$$g(t) = s(t) * p(t) = \sin(\omega_1 t) * \sin(\omega_2 t)$$

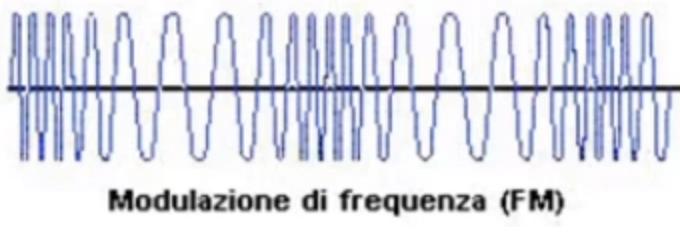
$$g(t) = \frac{1}{2} * [\cos((\omega_1 - \omega_2)t) - \cos((\omega_1 + \omega_2)t)] \text{ (Formule di Werner)}$$



Dai grafici vediamo che per effetto della modulazione passiamo da due elementi lontani tra loro a due elementi molto vicini tra loro (vicini a ω_2 , cioè la frequenza della portante, che supponiamo passi perfettamente all'interno del nostro canale di trasmissione, a differenza di ω_1 che invece non passava).

A destinazione basta fare l'operazione inversa per ottenere la componente originale.

Modulazione di Frequenza (FM)



Modulazione di frequenza (FM)

Modulo la frequenza dell'onda portante, l'effetto è una compressione nelle zone positive dell'onda originale, e una dilatazione nelle parti negative.

Diventa più complesso fare la traslazione alla frequenza della portante.

Modulazione di Fase (PM)



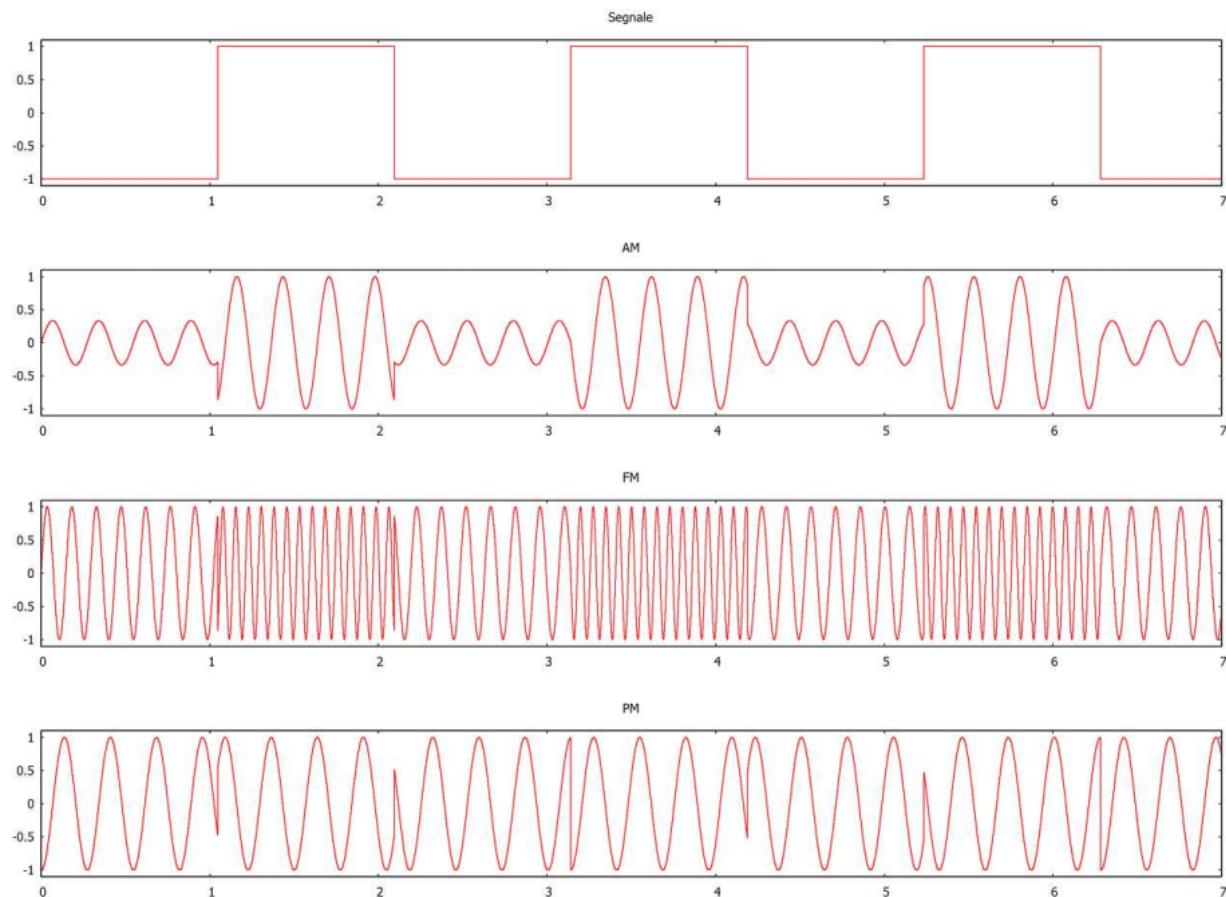
Modulazione di fase (PM)

Risultato quasi identico alla modulazione di frequenza, non è un caso perché partendo dallo stesso segnale e dalla stessa portante si può dimostrare che le onde risultanti modulando frequenza e fase sono legate matematicamente da un'operazione di derivazione / integrazione.

Spesso si usano due sistemi di modulazione in contemporanea; la televisione analogica usava ampiezza e frequenza.

Alcuni sistemi di trasmissione, come ad esempio i modem PSTN, usavano ampiezza e fase.

Tipi di modulazione



Partendo da un'onda quadra abbiamo esempi di modulazione in ampiezza, frequenze e fase.

Nella modulazione in fase si vede che la frequenza rimane uguale, ma ad ogni passaggio di onda quadra si nota chiaramente che cambia la fase.

Questo fa risultare facile notare cambiamenti nel segnale (e quindi porre bit 0 o 1 adeguatamente).

Analogico o Digitale

Una grandezza è detta **analogica** se può variare senza soluzione di continuità.

Una grandezza è detta **digitale** se può assumere solo valori ben precisi in un numero finito.

Un **segnale analogico** utilizza una larghezza di banda più limitata, ma subisce distorsioni per ogni processo rigenerativo (i.e. amplificazioni). Per esempio, una musicassetta copiata 10 volte, la decima avrà il rumore di dieci duplicazioni consecutive.

Un **segnale digitale** ha una larghezza di banda più elevata, ma può essere rigenerato con estrema precisione (quindi senza introdurre errori). Per questo motivo le comunicazioni digitali stanno soppiantando le analogiche.

Quindi, l'analogico si può dire che è “infinito”, mentre il digitale è discreto in quanto è quantizzato. Riprodurre un segnale analogico è più difficile che riprodurre un segnale digitale.

Attenzione però, Analogico e Digitale sono “modi” di vedere una trasmissione: non esiste nè una nè l'altra. La trasmissione analogica è la trasmissione di una variabile reale (nel senso matematico) che varia nel tempo. Una cosa del genere è impossibile da realizzare nella realtà perché il trasmettitore e il ricevitore vanno avanti per quanti discreti. Insomma, vanno avanti per step infinitesimi, ma sempre finiti (Già nessun trasmettitore può generare un numero infinito di livelli).

Il segnale analogico è un'approssimazione discreta di una variabile reale.

Passando al digitale, dipende tutto dalla precisione nella quantizzazione del segnale. Ad esempio, la separazione dei livelli nel framing dei dati (MLT3 o PAM5), spaziati in modo tale che il rumore non mi fa passare da un livello all'altro.

Essenzialmente, nel **sistema analogico** abbiamo un numero enorme di stati discreti, talmente enorme che il ricevitore/trasmettitore difficilmente riescono a discriminare tra un livello e l'altro.

Invece il **sistema** lo chiameremo **digitale** se il numero di stati che noi consideriamo validi per la comunicazione è un numero molto limitato (4, 16, 32...), in maniera tale che siano sufficientemente distanti tra loro per far sì che il rumore che si aggiunge alla trasmissione non vada a fare interpretare un livello rispetto a un altro.

Dunque, il segnale analogico, non esiste, come non esiste il digitale puro.

Sono tutte e due approssimazioni che facciamo per come consideriamo di voler comunicare. Con il segnale analogico abbiamo molti livelli distinti vicini tra loro, sapendo benissimo che il rumore è molto più forte come intensità rispetto al passaggio da un livello a quello attivo consecutivo. (Se il nostro occhio vede 256 sfumature di rosso, il disturbo che può venire da una lampadina è molto superiore a quello che serve per passare da una intensità a quella subito successiva).

Nel sistema digitale abbiamo un numero piccolo di valori ben distinti e ben spaziati tra loro, e consideriamo validi solamente questi due elementi, sapendo che ciò che arriva è un segnale che sta nel mezzo di queste due entità e che deve essere interpretato.

Più il sistema di interpretazione è efficiente, migliore sarà la possibilità di decodifica.

Mezzi Trasmissivi

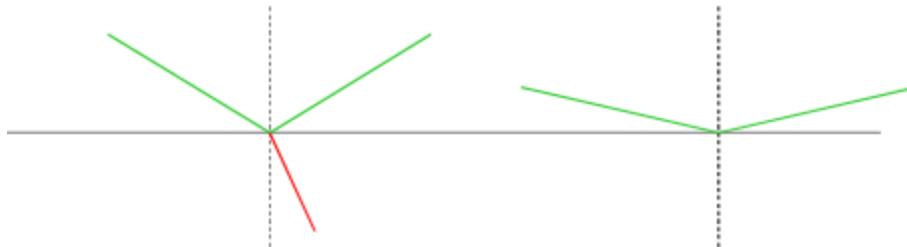
Ne esistono di vari tipi, dai più antichi (segnali di fumo, etc...) ai più recenti (Memorie flash...)

Abbiamo già parlato del doppino intrecciato, con le varie categorie, dove la differenza è nel tipo di intreccio utilizzato (migliore intreccio, migliore immunità a interferenze elettromagnetiche e rapporto S/N migliore). Possono anche avere le spire più piccole, isolante migliore... Infine, quello che succede è che riescono a passare segnali a frequenza maggiore con attenuazioni minori.

I cavi hanno tutti un'indicazione su che categoria è.

Si valuta anche quanta l'energia che passa da una coppia all'altra. Infatti all'interno del doppino è inserito un separatore che tiene le coppie separate a una certa distanza permette di limitare il passaggio di energia da una coppia di cavi a un'altra coppia vicina.

Fibra Ottica

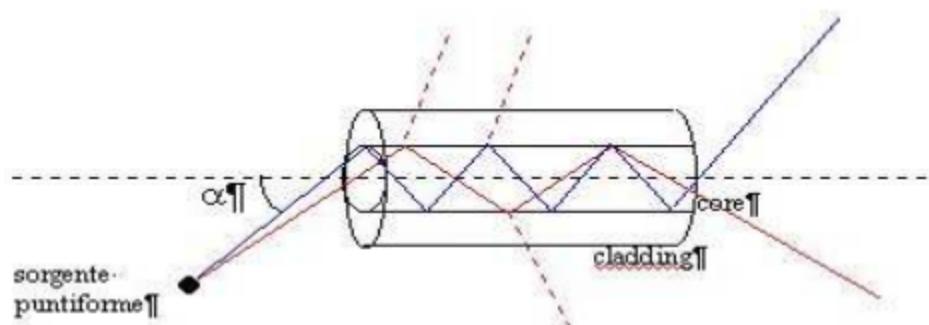


Un raggio luminoso, nell'interfaccia tra due mezzi ottici differenti, si scomponе in due componenti, una riflessa e l'altra rifratta. Oltre un certo angolo d'incidenza, la componente rifratta è nulla.

La componente rifratta penetra nel secondo mezzo quindi, a seconda dell'indice di rifrazione.

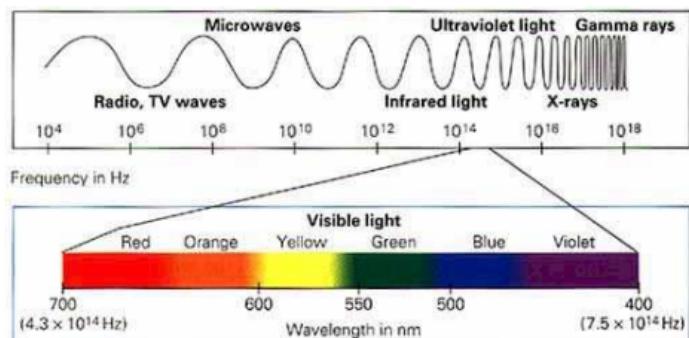
La somma energetica tra queste due componenti è pari al raggio iniziale.

L'idea della fibra ottica è di realizzare un tubo in maniera tale che il raggio luminoso che entra dentro la parte interna di questo tubo rimane intrappolato dentro la parte interna.



Realizzo una parte interna in vetro e una parte esterna sempre in vetro ma con caratteristiche ottiche differenti.

Una volta che il vetro viene ridotto a diametri molto piccoli diventa facilmente flessibile e possiamo far gli fare delle curve (non è rigido come una lastra).



La fibra ottica va bene per tutto lo spettro visibile, anche se ci sono alcune frequenze che passano bene e altre che vengono un po' attenuate.

La fibra ottica è fatta da una guaina esterna di plastica. Poi c'è una struttura in Kevlar che servono per dare resistenza alla trazione (perché altrimenti il cavo si romperebbe). Poi abbiamo un ulteriore rivestimento, un'altra protezione e finalmente la fibra (immagine non in scala):



Prima si mette il cavidotto (un tubo di plastica), dove all'interno passa la fibra. Non possiamo tirare come un normale filo elettrico la fibra ottica perché si romperebbe. Possiamo infilare la fibra nei cavidotti solo grazie alla parte in Kevlar. Le fibre ottiche sono unidirezionali, quindi normalmente si accoppiano. Alcuni cavi in fibra ottica, considerando che la parte più costosa del realizzare una struttura in fibra ottica è passare il cavo, alcuni cavi sono fatti da parecchie decine di cavi in fibra ottica tutti separati tra di loro, non devono essere isolati perché ovviamente non ci sono disturbi elettromagnetici fra loro (e neanche troppi disturbi ottici), quindi è necessaria solo la protezione meccanica in Kevlar. Oltre a tutti i rivestimenti visti sopra a volte hanno anche un rivestimento in metallo per rendere ancora più resistente il cavo ad elementi esterni e animali.

Per "giuntare" una fibra ottica si deve far sì che il connettore della fibra ottica maschio deve allinearsi con quello del connettore ricevente (operazione nell'ordine dei micron). Il centraggio viene aiutato da alcuni elementi



Il problema si pone quando si devono connettere due fibre separate a un singolo armadietto, per esempio. Se le fibre non sono messe perfettamente in asse non funziona niente.

Dato che dotare di microscopio ogni operaio che deve fare innesti risulta poco conveniente, di solito questa struttura meccanica si installa nelle schede / connettori stessi. Sono denominati per tipo di connettore e tipo di fibra da usare.

Giuntare due fibre è diverso dai cavi elettrici proprio a causa di questo allineamento che deve essere perfetto. Viene fatto con un apparecchiatura che è piuttosto portatile:



È effettivamente dotata di microscopio, per poter posizionare le fibre esattamente una di fronte all'altra.

Per prima cosa si tagliano con una piccola ghigliottina, perché il taglio deve essere quanto più regolare possibile (conviene farli diagonali).

Deve essere dritto per non creare increspature microscopiche.

Dopo aver fatto il taglio a volte si procede a fare una lisciatura molto sofisticata per eliminare qualunque forma di increspature, proprio perché farebbero deviare il raggio luminoso.

Per giuntare le fibre, dopo averle tagliate e posizionate, si fa scoccare un piccolo arco elettrico che fonde istantaneamente la fibra in quel punto, e crea una pallina sottile di vetro dove il raggio luminoso riesce a trovare anche la strada per uscire.

Ovviamente non è perfetta e si perde un po' di energia, ma va comunque bene.

Lo strumento permette anche di verificare quanta energia si perde a causa della giunzione.

Le ditte specializzate rilasciano una certificazione dove si specifica quali sono i risultati dei test sul prodotto realizzato. Se sono al di sotto dello standard è possibile non pagare la ditta.

Lezione 23 - 03/06/2020

Fibra ottica cont.

La fibra è comunque un canale, quindi ha sempre gli stessi problemi degli altri canali in rame, ma ha alcune caratteristiche peculiari che la rendono “migliore”.

Intanto, il fatto di essere un mezzo fisico dielettrico (non è propenso ad essere attraversato da corrente elettrica) fa sì che non ci siano interferenze elettromagnetiche. Quindi noi possiamo mettere una fibra vicino all'altra senza interferenze. Inoltre non c'è necessità di avere una protezione da interferenze elettromagnetiche esterne.

Usando il teorema di Shannon, il rapporto S/N è più elevato perché il valore del Noise è decisamente minore, dato che i disturbi esterni sono limitati.

Un altro motivo importante è dato da cosa può generare il rumore all'interno della fibra ottica. In una situazione non ideale (ad esempio nell'acqua), il disturbo è dato da riflessioni irregolari, cosa che non può accadere in una fibra ottica tradizionale.

Pur non essendo un vero e proprio rumore, è qualcosa che contribuisce al diminuire il rapporto S/N.

A causa della natura corpuscolare della luce scoperta da Einstein i fotoni si scontrano con gli atomi delle strutture molecolari del vetro e generano delle piccole riflessioni in tutte le direzioni (anche indietro). Per cui una parte dell'energia inserita nel cavo si disperde nel cavo stesso e viene dissipata in vari modi, tra cui calore. La fibra non si riscalda tanto poterlo percepire facilmente, anche perché l'energia che viene inserita nella fibra ottica è abbastanza limitata. Ciò che arriva a destinazione ci permette comunque di avere un rapporto S/N abbastanza elevato, sicuramente migliore di altri mezzi trasmissivi.

Altro motivo è legato alla frequenza della luce. Ricordiamo che H è la frequenza massima del segnale che attraversa il mezzo, per cui avere una portante ad alta frequenza permette di poter avere un bitrate maggiore a parità di tutte le altre condizioni.

Il fatto di poter utilizzare la luce visibile è un modo di poter migliorare il risultato nella formula di Shannon.

Quindi, i disturbi elettromagnetici non generano rumore in quanto la fibra ottica è un mezzo dielettrico; il rumore termico fa qualcosa perché l'agitazione delle molecole del vetro ovviamente disturba, ma a livello veramente basso; non vengono generate correnti parassite, quindi il bitrate è molto più ampio. In laboratorio si hanno valori dell'ordine di 50.000-100.000 Tbps.

Nell'uso pratico si possono avere collegamenti nell'ordine dei 10 Gbps su distanze di chilometri senza necessità di ripetitori. Dipende ovviamente dalla qualità della fibra e dal numero di connessioni sul percorso.

Dunque, viene passato il cavo in fibra ottica nei cavidotti e viene giuntato l'elemento finale al cavo che arriva, ovvero, non si passa una fibra col connettore già preparato in fabbrica, ma si passa la fibra senza la parte finale che di solito viene aggiunta effettuando una giunzione a cablaggio finito in loco.

Cosa si utilizza come mezzo trasmittivo?

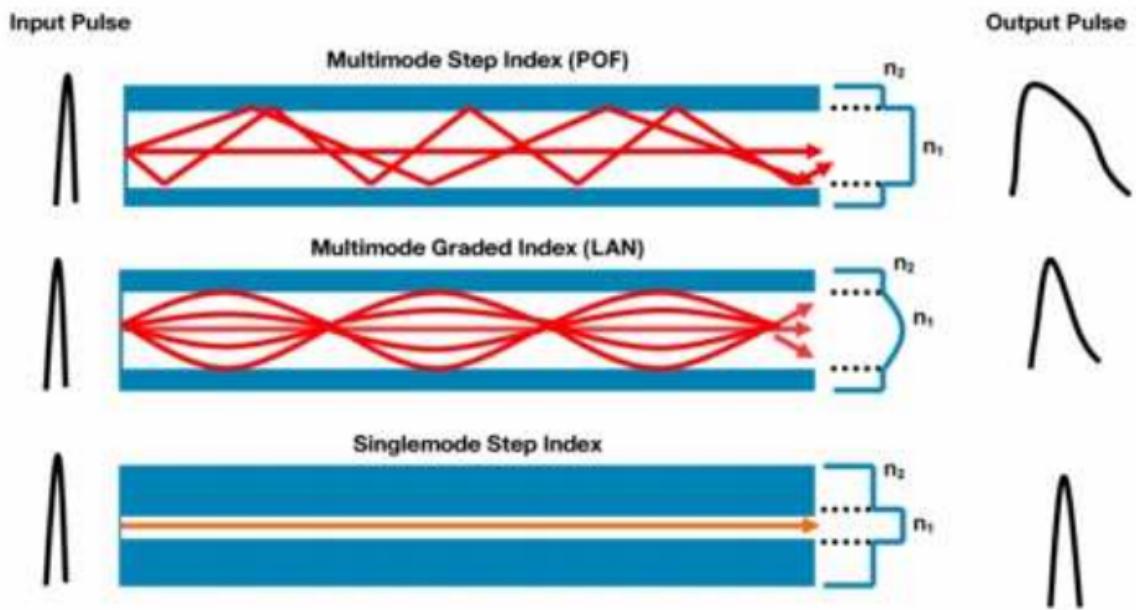
Andrebbe bene qualunque sorgente luminosa, ma:

Le **lampadine a incandescenza** hanno una velocità di commutazione (tempo che impiegano ad accendersi e spegnersi) elevatissima rispetto alle potenzialità della fibra ottica.

NB: Una lampadina spara raggi luminosi in qualsiasi direzione, che potrebbero anche annullarsi tra di loro, o anche solo disturbarsi tra loro. Fenomeno che non osserviamo a causa della quantità di raggi.

LED: Molto economici ma risentono del problema che la luce viene emessa in tante direzioni, non è concentrata. Hanno una velocità di comunicazione abbastanza decente, per cui potrebbero anche essere utilizzati per “bassa” velocità (bassa per gli standard della fibra ottica).

Laser: La luce emessa dai laser è coerente, cioè sono tutti perfettamente in fase tra loro, e quindi di fatto si amplificano a vicenda. Sono tutti coerenti tra loro e non si disturbano. Un trasmettitore laser non professionale infatti può anche bruciare una retina umana a causa dell'alta concentrazione.



Fibre ottiche realmente esistenti, a destra è mostrato il tipo di interfaccia che c'è tra mezzo interno ed esterno, che sono sempre in vetro ma con caratteristiche differenti.

Fibre Multimodali

La prima è una fibra tradizionale dove c'è un “gradino” netto di separazione tra il core interno e il cladding esterno. Questo fa sì che ci siano riflessioni ben nette all'interfaccia. Possiamo notare però che il raggio emesso non è sempre perfettamente in asse con la fibra ottica, ma viene emessa una varietà di raggi che per quanto limitata, dal generatore viene sempre emesso un piccolo cono che si riflette in tutte le direzioni, a partire dalla stessa sorgente. Ci sono più andamenti / riflessioni dalla stessa sorgente dello stesso segnale. Quello centrale ha una lunghezza diversa dalle varie riflessioni. Se le lunghezze diverse diventano proporzionali alla lunghezza d'onda del segnale vanno in opposizione di fase e si annullano tra loro o si disturbano.

Nel secondo tipo si ha un andamento “un po' più continuo” verso il punto di massimo, quasi come se fosse una lente, che si ottiene variando in maniera opportuna la caratteristica ottica della parte centrale della fibra.

Il risultato è che lo stesso tipo di raggio con varia angolazioni di partenza subisce una curvatura. I percorsi sono un po' più regolari rispetto al primo caso e permette di raggiungere velocità superiori, perché migliora il rapporto S/N.

Entrambe queste fibre sono caratterizzate dal fatto che sono presenti più raggi riflessi contemporaneamente, il diametro del core è di 50 micron.

Fibre Monomodali

La luce avanza senza riflessioni, assialmente. Consentono di coprire distanze maggiori con bitrate più alti. Sono però più costose.

Il core ha un diametro di 8 micron.

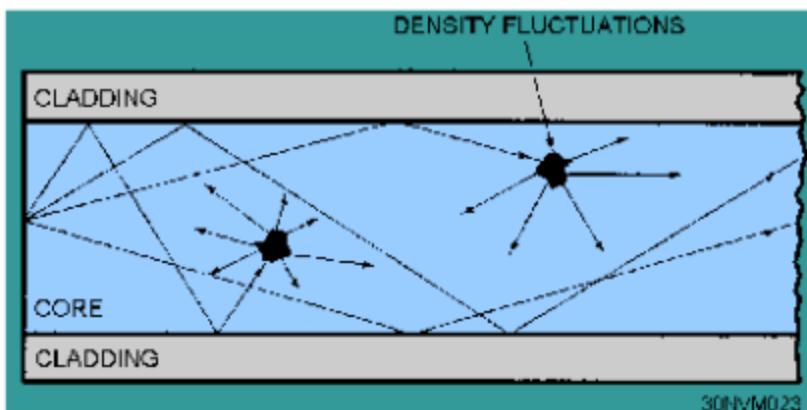
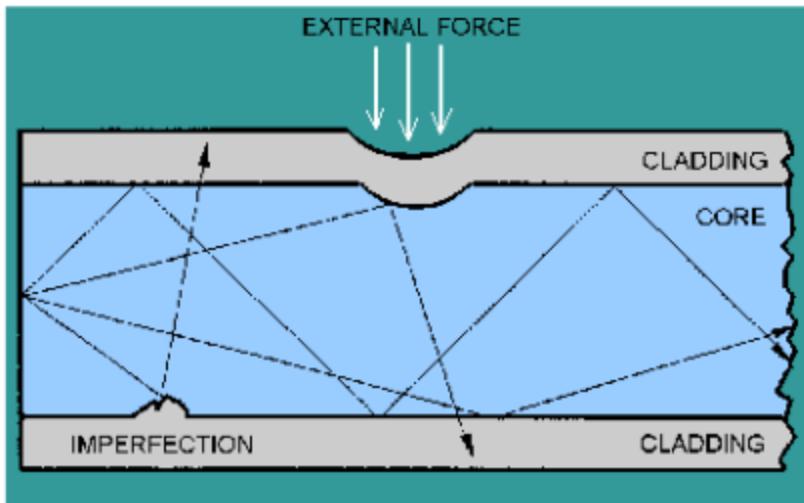
Cabling Standard	Cabling Type	Max Reach
10GBASE-SR	62.5µm OM3 multimode fiber	300m
	50µm OM4 multimode fiber	400m
10GBASE-LR	9µm single-mode fiber	10km
10GBASE-ER	9µm single-mode fiber	40km
10GBASE-ZR	9µm single-mode fiber	80km
10GBASE-LX4	9µm single-mode fiber	10km
	62.5µm multimode fiber	300m
	50µm multimode fiber	
10GBASE-LRM	9µm single-mode fiber	220m
10GBASE-T	Cat 6, Cat 6a or 7 twisted pair	30m
10G DAC/AOC	Copper RJ45	1-10m/up to 20m

Generalmente si connettono con delle patch più apparati in rame per raggiungere i valori della fibra ottica, ma oltre a un certo punto conviene ovviamente utilizzare la fibra. Le monomodali arrivano anche a 80km, le multimodali invece non vanno oltre qualche centinaio.

Tutti questi limiti ovviamente sono il limite in cui il funzionamento è garantito, ma è possibile superarlo a seconda del caso, o abbassando la velocità di comunicazione.

Il costo del realizzare un cablaggio non è il cavo in sé, ma la maggior parte del costo è dovuto all'installazione del cavidotto.

Nelle fibre ottiche si possono comunque avere delle perdite, a causa di deformazioni a causa di forze esterne, irregolarità nell'interfaccia di uscita che genera riflessioni in indietro o rifrazioni in avanti parallele all'asse della fibra stessa; oppure infine fluttuazioni di densità (non è a temperatura OK e quindi c'è agitazione molecolare).



Infine, il cavo spesso deve essere giuntato come detto precedentemente. Si realizza una "pallina di vetro", una sfera, possibilmente delle dimensioni della fibra stessa. Non garantisce il passaggio della stessa quantità di luce quindi ho una leggera diminuzione. Si deve quindi evitare di farne troppe per mantenere la velocità elevata.

La fibra è unidirezionale, si potrebbe avere una fibra bidirezionale sfruttando il prisma ottico, ma non conviene. Il costo di passare 2 fibre unidirezionali è irrisorio, e abbiamo un migliore S/N ratio.

Collegamenti Broadcast con Fibra Ottica

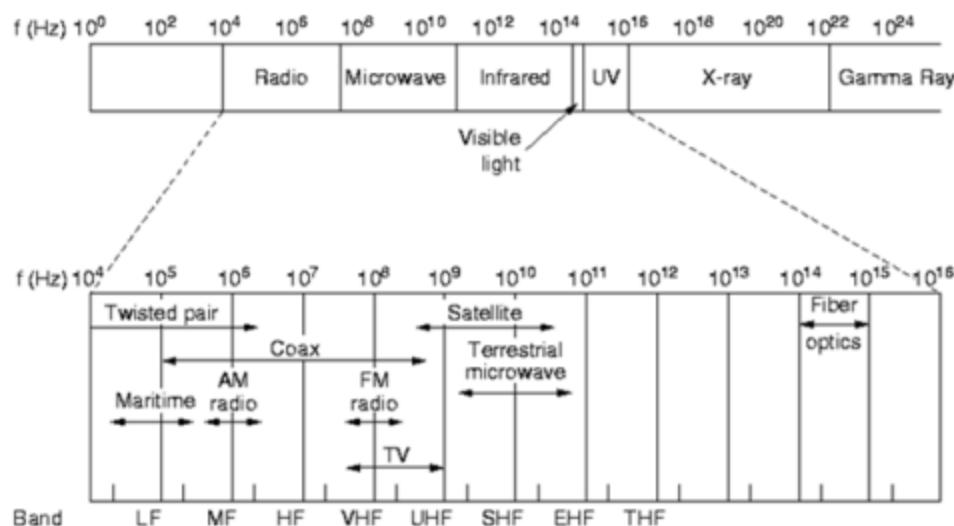
Il modo più semplice è quello di avere collegamenti punto-punto tra una macchina e la successiva, con un'interfaccia ottica di ricezione ottica-elettrica-ottica in maniera tale da realizzare un anello che riesce a comunicare in tempi separati con tutte le macchine. Cioè in pratica una macchina trasmette alla successiva, la quale legge, ricopia e trasmette alla successiva. Questo è un broadcast simulato senza collisioni.

Vi sono altri metodi ma di scarso utilizzo (tipo una fibra in andata e una in ritorno per ogni macchina, poi fuse tutte insieme per creare una stella ottica).

Reti Wireless

In molte applicazioni è utile realizzare reti di comunicazione senza cavi di collegamento fissi.

La sfida maggiore che le reti wireless devono affrontare sono i disturbi a cui la comunicazione è soggetta. Hanno inoltre pesanti problemi per quanto riguarda la qualità del servizio, soprattutto paragonandole alle reti cablate che ormai sono punto-punto, mentre una rete wireless è per definizione broadcast.



Ci concentriamo sulle onde radio invece che sulla luce visibile. Sono tutte utilizzate come mezzo di supporto per la modulazione, per cui abbiamo le onde AM, FM, trasmissioni LTE (al posto delle onde TV analogiche). La fibra ottica si trova nel visibile.

Al diminuire della frequenza, le strutture fisiche tendono a comportarsi meno da ostacolo nei confronti delle onde elettromagnetiche. Le onde a bassa frequenza (marittime, usate dalle navi) riescono a seguire la curvatura terrestre. Le onde ad alta frequenza sono più direzionali.

Le reti telefoniche

Le reti telefoniche sono state progettate per le comunicazioni fatte con la voce umana, per cui non si prestano facilmente alle comunicazioni tra computer. Il canale veniva suddiviso in slot separate e ogni slot assegnato a un utente, in maniera tale che le telefonate erano fisicamente separate su canali con frequenza differente. Un canale era largo all'incirca 4KHz, da 0 a 4K, con una zona di separazione tra 0 a 300 Hz e 3300-4000 Hz, per evitare sovrapposizioni tra canali attivi.

In altre parole, i provider inserivano normalmente un filtro passa-basso che elimina tutte le frequenze superiori a 3 o 3.3 KHz. Dopodiché una telefonata veniva messa nella zona tra 0 e 4 KHz, una seconda telefonata veniva traslata, e così via, fino a riempire il canale. Così nello stesso filo potevano passare tante telefonate differenti. La voce però veniva deformata a causa del taglio delle alte frequenze, cosa che per la voce non era drammatica perché gli esseri umani riescono a riconoscere anche timbri deformati

Il tasso di errore in una rete telefonica è nell'ordine di $1/10^5$ mentre nei cavi tradizionali arriviamo anche a valori di $1/10^{13}$, quindi molto più basso.

Modem

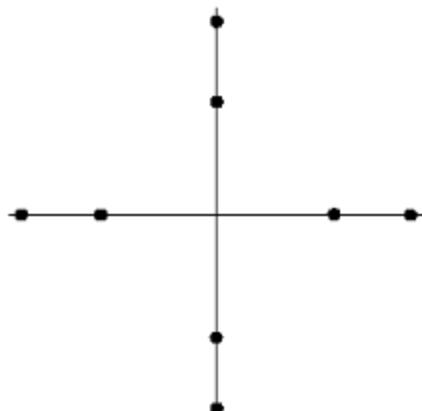
Modulatore-Demodulatore.

L'utilizzo delle linee telefoniche, essenzialmente analogiche, avviene utilizzando i modem, che trasformano il segnale da digitale ad analogico e differenza. Solitamente si usa una portante, opportunamente modulata entro ciò che il filtro fa passare.

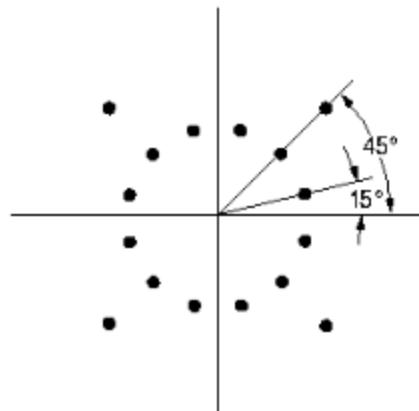
Veniva usata una portante a 2100 Hz, che è tra le frequenze che noi riusciamo a sentire.

Dato che la frequenza è bassa, per la formula di Nyquist dobbiamo inserire più livelli per aumentare il bitrate. Fu anche una questione di modulazioni differenti utilizzate: una portante in fase e una in ampiezza

Costellazioni



(a)



(b)

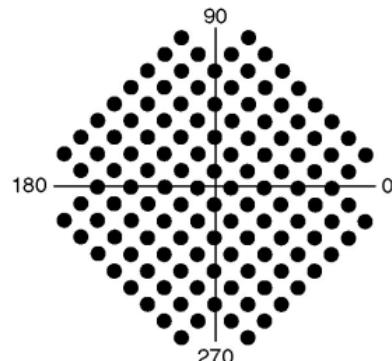
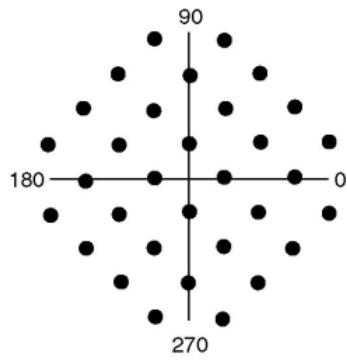
Lo schema (a) porta 3 bit per baud, mentre (b) ne porta 4 per baud, variazione legata alla portante, quindi 16 punti.

Con 2400 baud (variazioni) e un baud al secondo è quindi possibile arrivare a 9600 bps (9.6 Kbps).

L'ampiezza è la distanza dall'origine e la fase è lo sfasamento di gradi.

I modem devono usare la stessa costellazione per parlare tra loro.

Le costellazioni sono poi andate a evolversi:

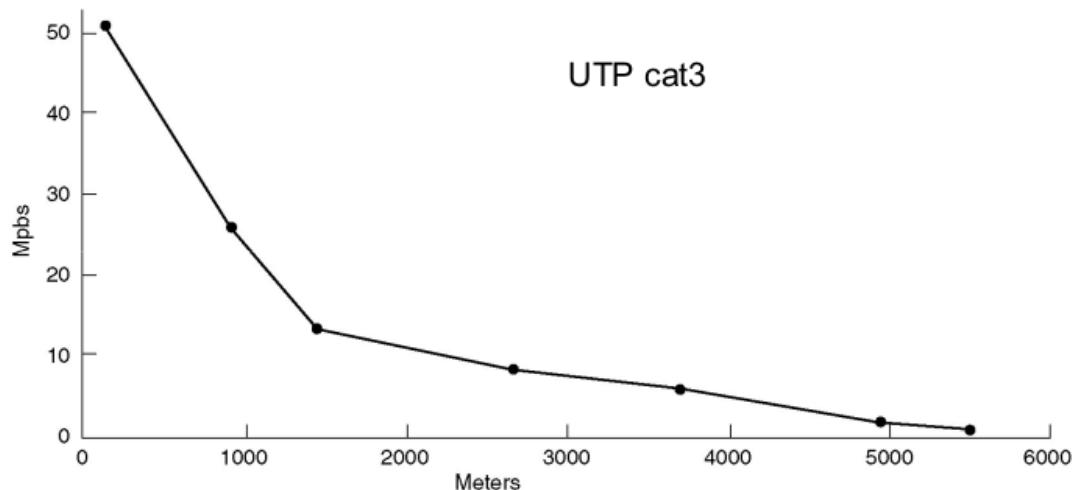


Le distanze sono tali che il livello del rumore inserito nel canale non fa passare da un pallino a un altro vicino e quindi interpretare in maniera scorretta l'informazione.

Il "fischio" e i conseguenti rumori di fondo che sentiamo usando un modem (o un FAX) è di fatto quello che permette di identificare i modem fra di loro. Quindi questo fischio è la portante. Poi iniziano una serie di test per capire qual è la velocità massima della linea. Si parte da una velocità più bassa e si aumenta piano piano fino a che viene aumentata la massima velocità possibile.

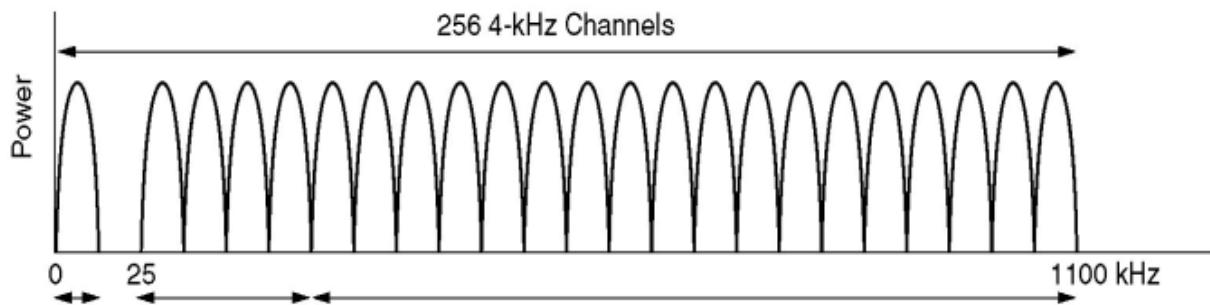
I modem stanno cioè testando la linea per capire a quale velocità si può arrivare, per adattarsi alle caratteristiche della trasmissione in particolare. Erano previsti anche standard di compressione dati in tempo reale per aumentare il throughput.

DSL



Uno dei problemi dei modem era il fatto che utilizzava il canale telefonico e di fatto il telefono era inutilizzabile.

Con le linee DSL hanno deciso di lasciare il canale telefonico libero (quindi tra 0 e 4K il telefono è disponibile per la comunicazione). Poi hanno lasciato un po' di spazio vuoto per evitare disturbi, dopodiché il resto del canale è stato suddiviso in blocchi da 4 K da utilizzare come fa il modem tradizionale, solo che è come se avessimo tanti modem in parallelo tra di loro per la trasmissione.



I vari slot sono poi stati suddivisi in slot di upstream e di downstream, differenziandoli perché l'idea era che il download doveva avere più banda rispetto all'upload.

I canali vengono testati tutti separatamente e per ogni canale viene trovata la velocità migliore di comunicazione.

Quindi, il telefono e il fax continuano a funzionare tradizionalmente, il modem DSL utilizza tutti i canali ma non quelli bassi.

Considerando che il telefono però non è di per sé bloccato sulle basse frequenze va inserito un filtro per bloccare per la parte di connessione che andava al telefono e al fax tutte le alte frequenze; non tanto per evitare il disturbo sulla linea telefonica, ma per evitare che il telefono disturbasse il modem, facendo calare la linea.

Sulla linea dati il modem aveva un suo filtro in ingresso per cui tutte le basse frequenze venivano eliminate, per cui di fatto il filtro serviva a bloccare tutto ciò che andava verso la linea telefonica tradizionale. Si realizza facilmente in elettronica con 3 componenti.

Da parte della centrale telefonica c'era un dispositivo denominato **DSLAM**, che serve per accoppiarsi correttamente al modem DSL. La DSL veniva denominata ADSL perché era Asimmetrica, per via della differenza di banda tra Upload e Download. Nella XDSL sono uguali invece.

ADSL Link	Downstream	Upstream
Connection Speed	2272 kbps	288 kbps
Line Attenuation	16 db	4 db
Noise Margin	31 db	31 db

ADSL Link	Downstream	Upstream
Connection Speed	7616 kbps	480 kbps
Line Attenuation	28.0 db	12.0 db
Noise Margin	13.1 db	26.0 db

ADSL Link	Downstream	Upstream
Link Rate	19252 Kbps	999 Kbps
Line Attenuation	24.0 dB	12.8 dB
Noise Margin	6.0 dB	16.3 dB

Velocità di linea di un modem DSL e stime sul margine di rumore e attenuazione.

Ratio/Factor	Decibels
2	6db
3.16	10db
10	20db
31.6	30db
100	40db
316	50db
1000	60db

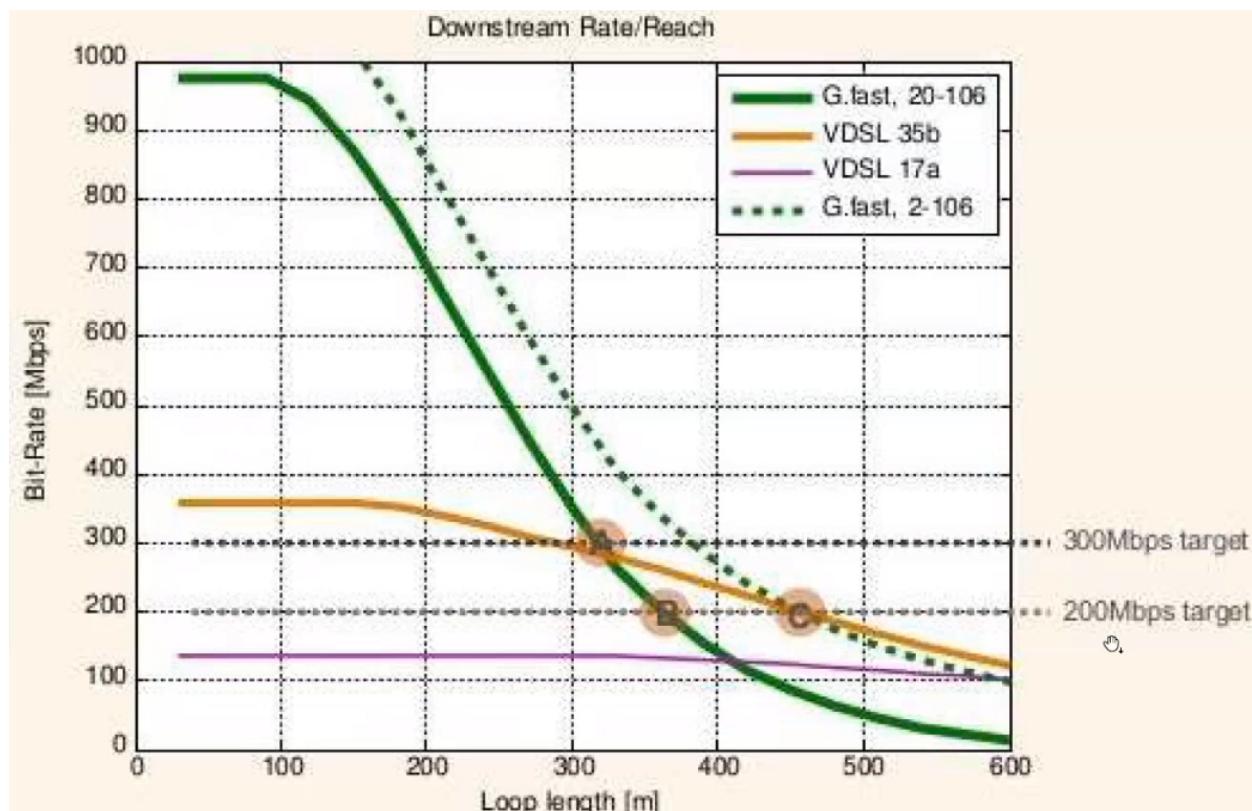
Margine di rumore (SNR)

- 5db o inferiore - pessimo
- 8db-13db - medio
- 14db-22db - molto buono
- 23db-28db - eccellente
- 29db-35db - dentro la centrale

Line Attenuation (in db)

- 20-30 - eccellente
- 30-40 - molto buono
- 40-60 - medio/scarso
- 60-65 - pessimo

Inoltre ci sono le linee VDSL, che propongono velocità molto superiori, e si è collegati ad armadi di strada (FTTC, Fiber to the Cabinet), e dopo il cabinet si prosegue in rame.



Teoricamente, la fibra migliore sarebbe la FTTH, che garantisce la velocità di 1 Gigabit
Con la VDSL si cerca di utilizzare di più lo spettro disponibile nel cavo di connessione:



Chiariimenti Post-Lezioni

12/06

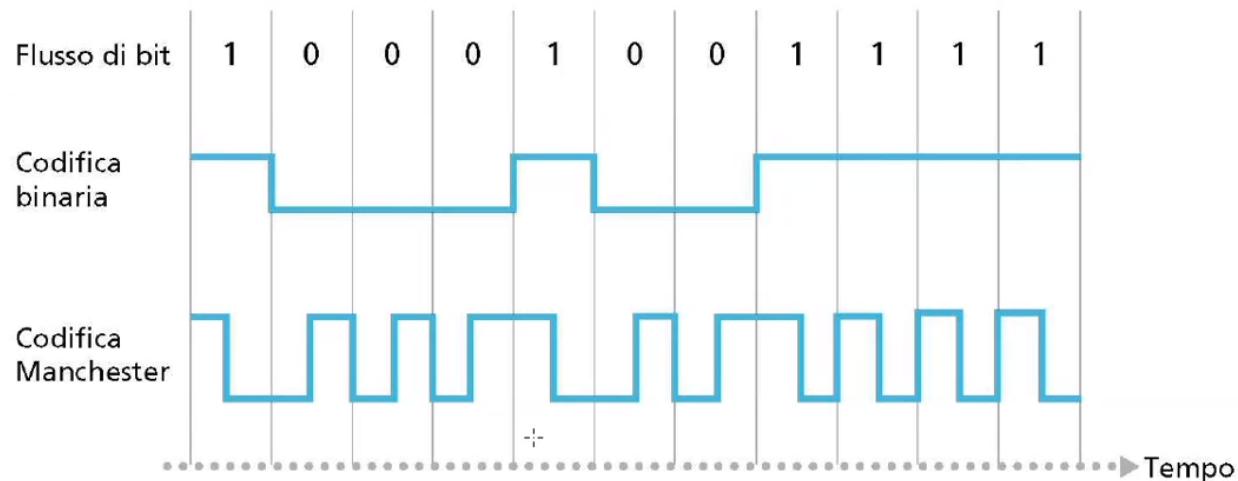
Controllo del Flusso e Controllo della Congestione

Essenzialmente, il controllo del flusso serve a evitare di mandare informazioni inutili quando il destinatario non è in grado di leggere queste informazioni, perché ha già il buffer di ricezione pieno. Questa è un'informazione facilmente reperibile da parte del destinatario perché usa la RWND proprio per informare quanto spazio c'è a disposizione. Non è un sistema perfetto perché l'informazione impiega tempo per arrivare a destinazione e soprattutto potrebbe perdere, perché non è detto che il pacchetto arrivi a destinazioni.

Il controllo della congestione non riguarda il destinatario ma la rete in generale, ed è dovuto al fatto che la rete viene intasata da duplicati che non fanno altro che rallentare la propagazione dei singoli pacchetti, per cui le applicazioni non riescono ad andare avanti perché ricevono sempre duplicati della stessa e vecchia informazione, che vanno a saturare i collegamenti in rete (**ATTENZIONE, È UN RIASSUNTO ESTREMAMENTE VELOCE E INSUFFICIENTE!!**)

In altre parole, il controllo del flusso è un problema end-to-end, il controllo della congestione no, ma a livello di rete. Dato che TCP non può accedere alle informazioni della rete l'approccio è vedere cosa succede nella connessione end-to-end tramite gli ACK di ritorno.

Codifica Manchester



Siamo agli inizi dei sistemi di comunicazione digitali e dobbiamo risolvere il problema del framing. In questo caso il problema era di identificare il bit 01.

È più semplice rilevare la transizione piuttosto che rilevare uno stato stabile, per cui codifico il bit 01 con una transizione alto-basso o basso-alto.

Il problema è il fatto che tra il passaggio di due bit uguali c'è una transizione basso-alto e poi un'altra alto-basso, e questi pezzi qui non devono essere interpretati come un bit 1, ma come "confine" tra due bit consecutivi. Di fatto nella figura riusciamo a capire tutto perché noi abbiamo le linee di separazione, ma il ricevitore non le ha ovviamente.

Il ricevitore vede solo una sequenza di bit, non ha dei marcatori di confine tra bit successivi. Devo identificare queste due zone differenziandole da quelle centrali.

L'idea è quella di mandare, prima dei bit di comunicazione, una sequenza perfettamente nota, in maniera tale da far sincronizzare i timer delle due macchine che comunicano (e.g. il preambolo di Ethernet).

Il passaggio 01 dà luogo a un'onda quadra. Di fatto una sequenza dove esistono solo i marcatori centrali. Questo consente al ricevente di capire che deve vedere solo i punti centrali, e non vedrà quelli di "frontiera".

La coppia 11 viene spedita per concludere il preambolo.

Fairness

“Le cose da dire sulla fairness sono”: Se non ci fosse la fairness non funzionerebbe Internet, totalmente. È una proprietà fondamentale di Internet.

Fairness vuol dire che la banda in ogni istante tende ad essere suddivisa in maniera equa tra tutte le connessioni che dividono quello stesso mezzo trasmissivo. Quindi se attraverso a un router passano 10 connessioni TCP differenti, non importa quando siano iniziate queste connessioni, istante per istante il sistema in automatico cercherà di equilibrare il traffico che viene dato alle varie connessioni.

Questa proprietà non è raggiunta tramite collaborazioni tra connessioni TCP (infatti non si parlano per niente), ma proprio per come è fatto il controllo della congestione. Se qualcuno tende a sforare, cioè chiedere molta più banda degli altri, prima o poi finisce in timeout e la sua banda viene dimezzata. Questo fa sì che lo spazio che lascia in qualche modo viene utilizzato dagli altri (**descrizione molto semplice, bisogna fare la dimostrazione grafica per aggiungere qualcosa di formale**).

Si deve comunque sottolineare cos'è la fairness e perché è importante.

Made with LOTS of love by Darakuu. <3

536F6D652070656F706C652068617665206D61646520756E6C617766756C20636F70696573206F66207468697320646F63756D
656E742C20636C61696D696E6720746F20626520746865206F726967696E616C20617574686F722028796F75206B6E6F772077
686F20796F75206172652C20201C4D644D201D203C3329