

# Appunti Reti di Calcolatori

---

A.A. 2011/2012 – 9 CFU

Questi appunti sono frutto di appunti presi in classe durante la lezione e completati con riferimenti dal libro o dalla rete, per tanto possono contenere informazioni incomplete, errori, refusi e altro. Non sono da considerarsi assolutamente come sostituzione del libro di testo del docente o delle lezioni.

## Sommario

Parte 1 – Introduzione .....	5
Applicazioni aziendali.....	5
Applicazioni domestiche .....	5
Applicazioni mobili.....	5
Hardware di rete.....	6
Reti Locali .....	7
Metropolitan Area Network .....	7
Wide Area Network.....	8
Reti Wireless .....	8
Reti tra sistemi .....	9
Gerarchie dei protocolli .....	9
Progettazione degli strati.....	10
Services connection oriented or connectionless .....	11
Primitive di servizio .....	11
La relazione tra servizi e protocolli .....	11
Il modello di riferimento OSI.....	12
Il Modello di riferimento TCP/IP .....	13
Confronto tra i modelli di riferimento OSI e TCP/IP .....	14
Commutazione di circuito e commutazione di pacchetto.....	15
ARPANET .....	16
Circuiti virtuali ATM (Asynchronous Transfer Mode) .....	17
Il modello di riferimento ATM .....	17
Parte 2 – Il livello Fisico .....	18
Analisi di Fourier .....	18
Analisi del segnale nel dominio del tempo e delle frequenze .....	20
Distorsioni nei segnali .....	20
Larghezza di banda.....	20
Campionamento e quantizzazione.....	20
Teoremi di Nyquist, di Shannon ed il rumore.....	21
Canali ideali, perfetti e canali ideali .....	22
La modulazione (ampiezza, frequenza e fase).....	22
Segnale Analogico o Digitale .....	24
I mezzi trasmissivi .....	24
Trasmissioni ad onde radio .....	27
Le reti telefoniche (sistemi a commutazione di circuito) .....	28
I modem analogici – la modulazione .....	28
Linee ADSL e xDSL .....	29

Tecniche di Multiplexaggio: FDM, WDM, TDM, CDM.....	29
Parte 3 – Data Link Layer .....	30
Framing dei dati .....	30
Codifica 4B5B .....	31
Codifica 8B10B .....	31
Aspetti teorici per la rilevazione degli errori .....	32
CRC e calcolo del CRC (Cyclic Redundancy Check).....	32
Distanza di Hamming .....	34
Codici di Hamming .....	35
Il sottolivello MAC.....	37
Aloha puro e slotted .....	38
CSMA (Carrier Sense Multiple Access).....	39
CSMA/CD.....	39
Protocolli senza collisioni.....	40
Protocolli a turno .....	40
Ethernet (802.3).....	40
Frame Ethernet .....	42
Codifica Manchester .....	42
Fast Ethernet (802.3u) .....	43
Gigabit Ethernet (802.3z).....	43
Hub Ethernet.....	44
Bridge Ethernet.....	45
Spanning Tree Protocol (IEEE 802.1d).....	46
Riepilogo hubs, bridges e switches .....	46
Indirizzi MAC (by wikipedia).....	46
VLAN (Virtual LAN – 802.1q) .....	47
Parte 4 – Livello di Rete.....	48
Servizi Datagram e a circuito virtuale .....	48
Generalità su algoritmi di Routing .....	49
Flooding.....	49
IPv4.....	50
Le maschere di sottorete .....	51
Formato Datagram IPv4 .....	52
DHCP (Dynamic Host Configuration Protocol) .....	54
NAT (Network Address Translation) .....	55
Distance Vectors .....	55
Link State Routing e Algoritmo di Dijkstra .....	56
Confronto Distance Vector e Link State.....	56

Internet Routing.....	57
RIP (Routing Information Protocol).....	57
RIPv1 .....	57
RIPv2 .....	58
RIPng (RIP new generation) .....	58
OSPF (Open Shortest Path First) .....	58
BGP-4 (Border Gateway Protocol v.4).....	59
Indirizzi LAN e IP.....	60
ARP (Address Request Protocol), RARP (Reverse ARP) e BOOTP (Bootstrap Protocol).....	60
Tabelle di routing per host.....	61
Multicast .....	61
Frammentazione .....	62
ICMP (Internet Control Message Protocol).....	63
IPv6.....	63
Firewall.....	66
Parte 5 – Livello di Trasporto .....	67
Indirizzamento nel TL (mux – demux).....	67
UDP (User Datagram Protocol) .....	67
Reliable data transfer.....	67
Protocolli Stop&Wait, Pipeline, Go back N e Ripetizione Selettiva .....	68
Introduzione a TCP (Transmission Control Protocol).....	69
Formato dei pacchetti TCP .....	70
Finestra di ricezione e gestione ACK in TCP .....	71
I tempi di Round Trip, Gestione dei Timer e Fast Retransmit.....	72
Apertura e chiusura delle connessioni.....	74
La congestione: aspetti teorici .....	75
Controllo della congestione in TCP .....	76
Fairness tra connessioni concorrenti .....	77
Parte 6 – Livello Applicativo .....	77
Comunicazione tra i processi .....	78
Indirizzamento nelle comunicazioni tra Processi: Well known Ports, Registered Ports e User Ports .....	78
Requisiti delle applicazioni.....	79
Protocollo HTTP .....	80
Protocollo FTP (File Transfert Protocol).....	81
Protocolli SMTP (Simple Message Tranfert Protocol), POP e IMAP .....	82
Il protocollo DNS (Domain Name System) .....	82
Parte 7 – La sicurezza nelle reti.....	84
Cifrari a sostituzione e a trasposizione .....	85

Metodo one-time pad.....	85
Principi per la crittografia .....	85
Crittografia simmetrica e asimmetrica .....	85
DES (Data Encryption Standard) .....	86
RSA .....	87
Protocolli di Autenticazione .....	88
Autenticazione con chiave segreta .....	88
Scambio di chiavi Diffie-Hellman .....	89
KDC (Key Distribution Center).....	89
Autenticazione in Kerberos.....	90
La firma digitale.....	90

## Parte 1 – Introduzione

La nostra epoca è caratterizzata dall'assimilazione e condivisione delle informazioni. All'inizio le aziende o le università, avevano pochi calcolatori che soddisfavano i bisogni dell'intera organizzazione; ai giorni nostri è stato sostituito da un nuovo modello di pensiero: avere una rete di calcolatori distinti, ma interconnessi per svolgere le operazioni. Questo nuovo sistema viene chiamato **rete di calcolatori.**

**Due computer si dicono interconnessi, quando sono in grado di scambiare informazioni.** Il collegamento non è necessariamente realizzato con cavi di rame; si possono usare anche **fibre ottiche**, microonde, luce infrarossa, satelliti e molto altro che vedremo più avanti.

Bisogna differenziare le reti di calcolatori da un **sistema distribuito**: la differenza sostanziale è che in un sistema distribuito l'insieme di computer indipendenti appare ai propri utenti come un **singolo sistema coerente**. Spesso la realizzazione di questo modello è affidata ad uno strato software posto sopra il sistema operativo, chiamato **middleware**. Un sistema distribuito è in fin dei conti un software costruito sopra una rete.

In una rete di calcolatori, invece, gli utenti vedono le macchine effettive ed il sistema non compie tentativi per far apparire e agire le macchine in modo coerente.

### Applicazioni aziendali

In un'azienda la **condivisione delle risorse**, ovvero condividere risorse presenti su pc differenti, può essere una reale esigenza. Un esempio comune è la stampante di rete. Senza ombra di dubbio le informazioni sono le risorse più importanti da condividere.

Il modello più comune è quello **Client-Server**. I server, frequentemente, sono installati negli edifici centrali o più importanti, mentre i Client sono installati nelle filiali o nei piccoli uffici. Il server mette in "rete" le informazioni da condividere, che sono accessibili esclusivamente dai Client. Le macchine client e server sono collegate, ovviamente, da una rete. Nella maggior parte dei casi i server possono soddisfare le richieste di numerosi client; la comunicazione avviene tramite un processo client che invia una richiesta al processo server e nel mentre rimane in attesa della risposta.

Una rete di calcolatori, oltre che un potente mezzo di condivisione di risorse, può essere anche un **mezzo di comunicazione** tra impiegati.

Le applicazioni **e-commerce** sono un altro esempio di applicazione Client-Server.

### Applicazioni domestiche

Anche se in principio i grandi produttori di calcolatori dell'epoca fossero molto scettici, il **personal computer** è entrato con prepotenza all'interno delle nostre case. In principio la gente acquistava il personal computer per scrivere testi o giocare, ma ora lo scenario è radicalmente mutato; probabilmente ai giorni nostri il motivo principale è la navigazione su internet. Altro motivo è la comunicazione tra persona e persona, un esempio in particolare è il **modello peer-to-peer**; in questa forma di comunicazione, individui che formano gruppi dispersi possono comunicare con altri nel gruppo. Ognuno può comunicare con una o più delle altre persone, non esiste una distinzione rigida tra client e server. La successiva generazione di sistemi peer-to-peer ha eliminato la presenza del database centrale, delegando ad ogni utente la possibilità di poter creare il proprio database personale di contenuti. Un nuovo utente può quindi collegarsi ad un nuovo utente per vedere cosa offre e cosa offrono gli utenti collegati ad esso, se presenti.

Altri scenari di utilizzo sono: l'intrattenimento mediante video on-demand, videogiochi online, e-commerce e tutto ciò che può essere "inserito" in rete.

### Applicazioni mobili

L'importanza delle **reti wireless** è in continuo aumento in questi anni, in quanto permette l'accesso ad internet o a risorse remote, senza ricorrere all'installazione di cavi. Altro utilizzo di una rete wireless, che non deve essere per forza una rete interne, è l'utilizzo di ponti radio per gestire i taxi o i camion di una società, oppure in ambito militare,

in quanto è possibile installare infrastrutture di comunicazione on-fly, ovvero senza l'ausilio di cavi o altri sistemi che rendono la rete fissa e difficile da installare.

Benché le reti wireless e le applicazioni mobili siano spesso correlate, non sono la stessa cosa. Infatti un portatile può essere collegato in modo “fisso” ad una rete con l'apposito cavo, quindi se un viaggiatore collega il suo pc portatile alla presa telefonica dell'albergo ottiene mobilità senza utilizzare una rete wireless.

D'altro canto non è detto che un pc wireless sia portatile, può esistere il caso in cui una azienda voglia creare una rete tra pc “fissi” senza effettuare il cablaggio, allora deve rivolgersi alla tecnologia wifi necessariamente, ma i pc non si spostano dalla loro posizione.

Ovviamente esistono applicazioni mobili per eccellenza, persone che tramite un PDA collegato in wireless, riescono a comunicare con il pc centrale mentre sono in movimento o comunque molto distanti (es. magazziniere che effettua inventario). Altro esempio sono gli smartphone dei giorni nostri, che ci permettono di accedere alla posta elettronica o ad altre risorse di rete, mentre siamo in movimento.

## Hardware di rete

Esistono due tipi di tecnologie trasmissive impiegate a largo spettro:

- **Collegamenti Broadcast:** le reti broadcast hanno un solo canale di comunicazione che è condiviso fra tutte le macchine della rete. Brevi messaggi, chiamati anche pacchetti, sono inviati da ciascuna macchina e ricevuti da tutte le altre. Un campo di questo pacchetto è destinato all'indirizzo, la macchina controlla questo indirizzo e se rivolto a lei allora processa il pacchetto, altrimenti lo passa alla macchina successiva. Il sistema broadcast permette anche di inviare particolari pacchetti a tutte le macchine: nel campo indirizzo è presente un tag speciale che “comunica” a tutte le macchine di processare il pacchetto. Questo modo di operare viene chiamato broadcasting. Le macchine possono anche ricevere il pacchetto e a loro volta propagarlo per una sotto-rete; questo modello viene chiamato multicasting. Un modo per implementare il multicasting può essere riservare un bit dell'indirizzo per indicare il multicasting e gli n-1 bit per contenere il numero di gruppo; ogni macchina può decidere di “inscriversi” a uno o all'altro gruppo. 
- **Collegamenti punto-punto:** i collegamenti punto-punto consistono in molte connessioni tra singole coppie di macchine. In questo modello, il pacchetto, per raggiungere la destinazione, deve attraversare una o più macchine intermedie, per cui un ulteriore problema aggiuntivo è quello di calcolare il percorso meno dispendioso. La trasmissione punto-punto con un solo trasmettitore ed un solo ricevitore è chiamata anche unicasting. 

Un ulteriore criterio utilizzato per classificare le reti è mediante la loro scala. Un esempio è dato dalla figura successiva, più avanti vedremo in dettaglio.

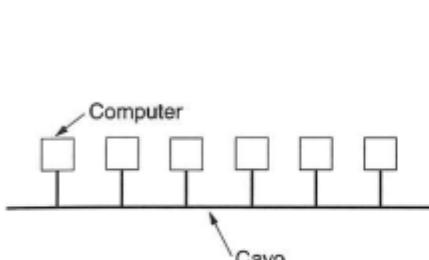
Distanza tra i processori	Processori situati nello stesso/a	Esempio
1 m	Metro quadrato	Personal area network
10 m	Stanza	
100 m	Edificio	Local area network
1 km	Campus	
10 km	Città	Metropolitan area network
100 km	Nazione	
1.000 km	Continenti	Wide area network
10.000 km	Pianeta	Internet

## Reti Locali

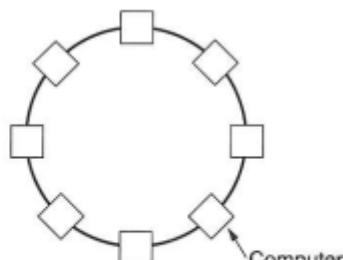
Dette anche comunemente **LAN (Local Area Network)**, sono reti private installate all'interno di un singolo edificio o campus, con dimensione massima fino a qualche chilometro. Largamente utilizzate per collegare pc all'interno di aziende o università, per facilitare la condivisione delle risorse.

Caratterizzate da:

- **Dimensione:** hanno dimensioni contenute, per cui il tempo massimo di trasmissione più sfavorevole ha un limite, che normalmente è noto. Conoscere questo limite permette l'applicazione di diverse tecniche di trasmissione che non sarebbero altrimenti applicabili (es. trasmissione e attesa di risposta).
- **Tecnologia di trasmissione:** Possono utilizzare una tecnologia di trasmissione rappresentata da un cavo che connette fisicamente tutte le macchine. Le LAN tradizionali lavorano solitamente a velocità comprese tra 10Mbps e 100Mbps, hanno bassi ritardi e pochissimi errori. Alcune LAN più recenti lavorano anche a 10Gbps.
- **Topologie:**



(a)



(b)

Nella figura ne sono presenti due tipologie (prese tra le tante disponibili). Il tipo (a) è il modello a **BUS**; formato da una rete lineare che connette i vari pc, esso permette in ogni istante un solo master (ovvero trasmettitore) alla volta, precludendo alle altre macchine di trasmettere. È ovviamente necessario un meccanismo di arbitraggio per risolvere gli ovvi conflitti quando due o più macchine desiderano comunicare. Un esempio è l' **IEEE 802.3**, chiamata abitualmente **Ethernet**, ovvero una rete broadcast a bus con controllo non centralizzato, ma la vedremo in dettaglio più avanti.

Il secondo tipo (b) è detto ad **anello**; in questo modello ogni bit si propaga in modo autonomo, senza aspettare il resto del pacchetto a cui appartiene. Anche in questo caso è necessario un sistema che arbitri la trasmissione, un esempio è il sistema **token ring** ideato da IBM.



Le reti broadcast, a loro volta, si possono suddividere in:

- **Statiche:** una tipica allocazione statica è la suddivisione del tempo in intervalli discreti e utilizzare un algoritmo round-robin, permettendo ad ogni macchina di trasmettere solo quando è il suo turno ed entro quell'intervallo di tempo. Andiamo incontro a sprechi se la macchina non ha niente da trasmettere.
- **Dinamiche:** è una soluzione al problema dello spreco citato poc'anzi, dove le macchine cercano di allocare il canale solo quando è necessario (quindi dinamicamente). Se il metodo di allocazione è **centralizzato**, allora esiste un sistema che si occupa solo di decidere chi ha diritto di "parlare", in mancanza di questo sistema allora si ha un metodo di allocazione **non-centralizzato**.

## Metropolitan Area Network

Una rete metropolitana (**MAN**) copre un'intera città. Un esempio noto è la televisione via cavo presente in molte città degli USA. Quando internet ha iniziato ad interessare le masse, gli operatori delle TV via cavo hanno capito che potevano offrire un servizio internet bidirezionale utilizzando bande dello spettro inutilizzate. In questo sistema il segnale televisivo e internet viene inserito in una **stazione di testa** centralizzata, che si occupa di distribuire il segnale nelle abitazioni.



## Wide Area Network

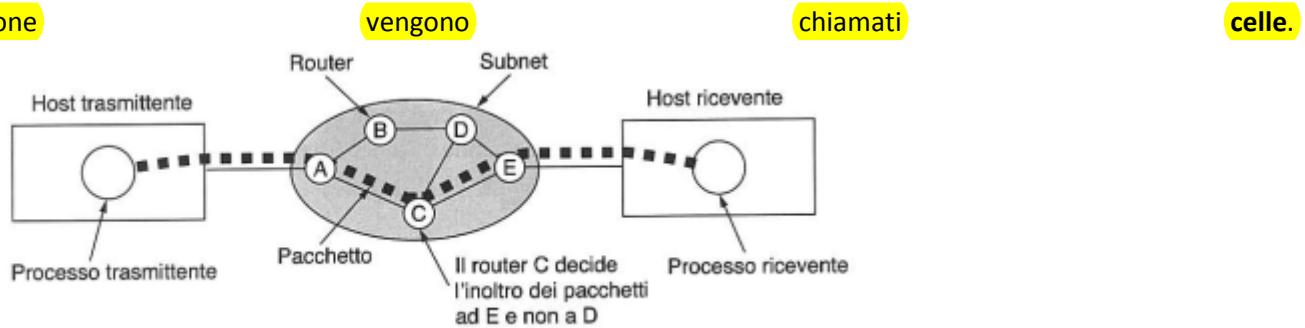
E' una rete che copre un'area geograficamente estesa. Racchiude una raccolta di macchine destinate a eseguire programmi utente. Queste macchine vengono chiamate **host**, esse sono collegate tra loro mediante una **communication subnet (subnet)**.  Host sono di proprietà dei clienti (es. pc degli utenti), mentre le subnet sono possedute e gestite dalle compagnie telefoniche o da un Internet service provider.. Il compito della subnet è di trasportare i messaggi da un host all'altro.

Le **Linee di trasmissione** spostano i bit tra le macchine, possono essere realizzate con cavo in rame, **fibra ottica**, o anche ponti radio.

Gli **elementi di commutazione** sono computer specializzati che collegano tre o più linee di trasmissione. Quando i dati arrivano da una linea l'elemento di commutazione deve decidere in quale altra linea inviare i dati ricevuti. Oggigiorno questi elementi di commutazione vengono chiamati comunemente **router**.

Le WAN vengono molte volte utilizzate in accoppiata con una o più LAN. (es. casa mia all'interno possiede una LAN e poi è collegata ad internet con una WAN, come tutti gli altri)

Una rete WAN **store-and-forward** o **packet-switched** (a commutazione di pacchetto) funziona secondo questo principio: quando un pacchetto viene inviato da un router verso un altro, attraverso router intermedi, il pacchetto viene ricevuto integralmente da ciascun router intermedio, memorizzato finché non si libera la linea di uscita necessaria, e poi inoltrato. Quasi tutte le WAN hanno un sistema del genere, e tutti i pacchetti piccoli e della stessa dimensione



Quando un processo di qualche host ha un messaggio da inviare a un processo di qualche altro host, quello che trasmette inizia l'attività dividendo il messaggio in pacchetti, ciascuno dei quali porta un corrispondente numero di sequenza. Successivamente questi pacchetti vengono inseriti nella rete uno dopo l'altro in rapida successione, vengono trasportati dalla rete individualmente e depositati e ricomposti nell'host ricevente. In alcuni reti tutti i pacchetti di un dato messaggio devono seguire lo stesso percorso, in altre è possibile scegliere strade differenti. Localmente si decide la strategia di instradamento, il modo in cui il router di turno decide per quale strada inoltrare il pacchetto viene detto **algoritmo di routing**.

L'alternativa alla WAN a commutazione di pacchetto è quella satellitare.

## Reti Wireless

Si può avere di diversi tipi:

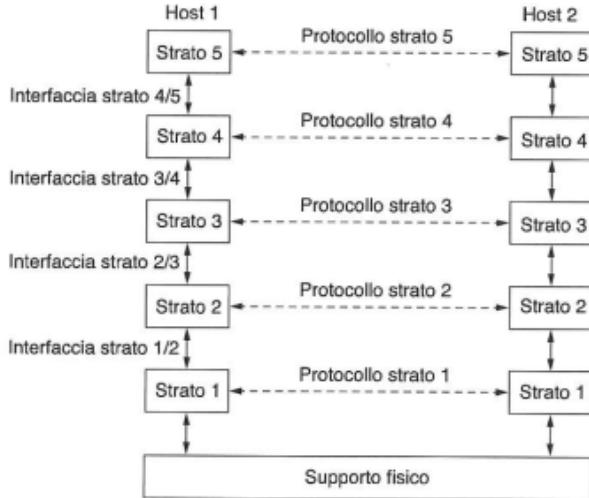
- **Connessioni all'interno di un sistema** dove le varie periferiche sono collegate al pc centrale mediante tecnologie senza fili (es. Bluethooth), per abolire i cavi di collegamento.
- **LAN wireless** ovvero connessione LAN mediante l'utilizzo di tecnologie senza fili; a casa mia ad esempio è presente una LAN mista formata da dispositivi cablati e wifi. Lo standard maggiormente diffuso è lo **IEEE 802.11**.
- **WAN wireless** stesso principio di sopra solo con aree geografiche molto estese.

## Reti tra sistemi

Molte volte un utente desidera connettersi ad un sistema differente dal suo e molte volte anche incompatibile; solitamente la connessione tra questi tipi di sistemi avviene mediante dei **gateway** che stabiliscono la connessione e ed offrono i servizi necessari a gestirla. Un sistema di reti interconnesse si chiama **internetwork** o **internet**. Una forma comune di internet è un insieme di LAN collegate tra loro mediante una WAN. E' importante non confondere il significato di **internet generico** con quello di **internet mondiale**.



## Gerarchie dei protocolli



Per diminuire la complessità, la maggior parte delle reti è organizzata secondo una struttura a **livelli o strati**, costruiti uno su l'altro. Lo scopo di ogni strato è di **offrire determinati servizi agli strati di livello superiore, schermendoli dai dettagli sull'implementazione dei servizi**. In un certo senso, ogni strato è una specie di macchina virtuale che offre certi servizi allo strato che si trova al di sopra.

Generalmente lo strato *n* di un pc è in comunicazione con lo strato *n* dell'altro pc; le regole di comunicazione utilizzate per questa comunicazione vengono chiamate **protocolli dello strato *n***.

Un **protocollo** è un accordo, tra le parti che comunicano, sul modo in cui deve procedere la comunicazione. Violare il protocollo rende la comunicazione difficile e il più delle volte impossibile.

Le entità che formano gli strati di pari livello sui diversi computer sono chiamati **pari (peer)**. I pari possono essere processi, dispositivi hardware o anche esseri umani. I pari comunicano tra loro mediante il protocollo.

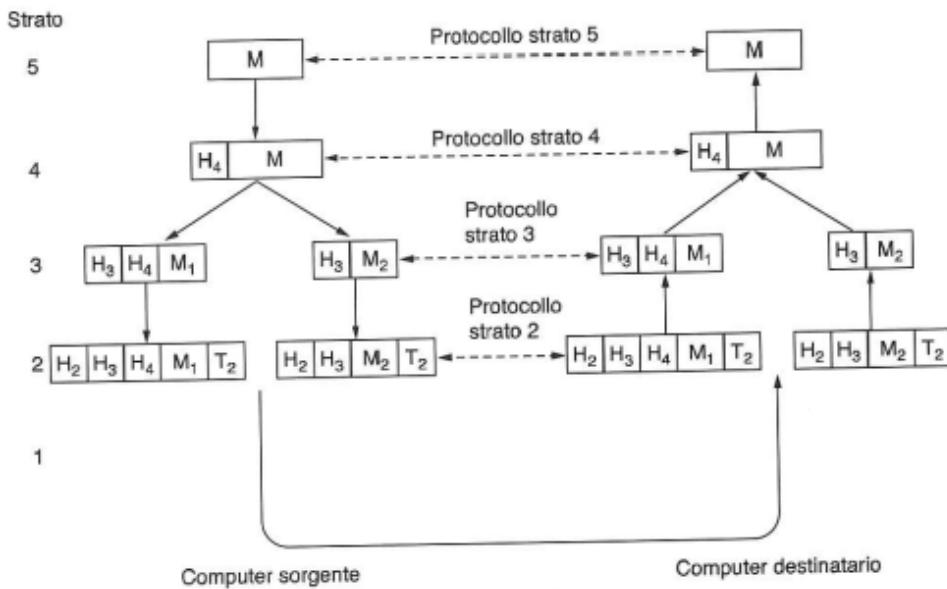
Lo strato *n* non comunica direttamente con il suo alter ego dell'altra macchina, ma bensì, passa le informazioni al suo strato sottostante, in una comunicazione a catena.

Sotto l'ultimo strato si trova il **supporto fisico** ovvero il mezzo fisico vero e proprio (rame, fibra ottica, wifi,...) che permette la comunicazione vera e propria.

Tra ciascuna coppia di strati si trova un'**interfaccia** che facilita la comunicazione tra ogni coppia di strati, esponendo le operazioni necessarie per permettere la propagazione del messaggio verso il livello inferiore o superiore. L'implementazione di interfacce permette anche la possibilità di apportare modifiche al sistema a strati, senza dover modificare gli strati stessi.

**L'insieme di strati e di protocolli si chiama architettura di rete.** Le specifiche di un'architettura devono contenere abbastanza informazioni da permettere agli sviluppatori di scrivere il programma o di costruire l'hardware di ciascuno strato in modo che possa seguire correttamente il protocollo. I dettagli di implementazione e le specifiche delle interfacce non fanno parte dell'architettura di rete, inoltre non è necessario che le interfacce dei vari computer siano tutte uguali, l'importante è che la specifica macchina possa utilizzare i protocolli standard.

Codesti protocolli sono tutti racchiusi in un elenco chiamato **pila dei protocolli**, molto utile per facilitare la progettazione.



Nella figura precedente un esempio di comunicazione tra due computer. La cosa importante da capire è che i vari livelli non comunicano con il loro alter ego delle macchine remote direttamente, ma mediante l'utilizzo dei livelli sottostanti. Il protocollo è quindi, per dirla in maniera molto barbara, una sorta di chiave di cifratura che permette l'apertura del messaggio.

## Progettazione degli strati

Per garantire una corretta comunicazione è necessario affrontare alcuni problemi in fase di progettazione dei strati (o livelli) brevemente quelli più importanti:

- **Indirizzamento:** per garantire la comunicazione tra processi di macchine differenti è necessario implementare un sistema di indirizzi che permetta l'identificazione del giusto destinatario.
- **Controllo degli errori:** il vettore fisico non è un mezzo perfetto; è molto frequente la possibilità di avere disturbi di comunicazione o perdita di pacchetti. Ne esistono di differenti, per cui è necessario che mittente e destinatario si accordino per utilizzare lo stesso sistema di correzione.
- **Controllo di flusso:** è importante controllare che il mittente non sovraccarichi il canale di collegamento (es. destinatario più lento del mittente). Un modo semplice di risolvere è usando una cadenza di trasmissione controllata.
- **Routing o istradamento:** quando esistono più percorsi tra sorgente e destinazione si pone il problema di scegliere il migliore.
- **Lunghezza dei messaggi:** alcuni processi non hanno la possibilità di accettare messaggi arbitrariamente lunghi, per cui urge trovare un metodo per disassemblare, trasmettere e riassemblare i messaggi.
- **Quantitativo di canali di comunicazione:** se possibile si utilizza un canale per ogni connessione, altrimenti si cerca di riutilizzare lo stesso per più connessioni. Se il **multiplexing** o **demultiplexing** sono realizzati in maniera trasparente, si può decidere di utilizzare il metodo multiplexing (Ovvero tutti i segnali in uscita o entrata sono codificati in un unico segnale, è necessario però avere un multiplexer che codifica e un demultiplexer che decodifica).
- **Ordine di invio dei messaggi:** non tutti i canali di comunicazione riescono a conservare l'ordine di invio o ricezione del messaggio, per cui è necessario riordinare i pacchetti una volta giunti a destinazione.
- **Direzione di invio:** se si dispone di una connessione monodirezionale o bidirezionale bisogna tenerne conto.

## Services connection oriented or connectionless

Ogni servizio può essere qualificato in base alla **qualità del servizio (QoS)**. La qualità del servizio è basata sull'effettivo arrivo del messaggio e sulla sua interezza; i meccanismi atti a controllare la qualità, a volte, introducono dei ritardi e appesantiscono la comunicazione, per cui sono inseriti solo quando necessari (es. trasferimento file). Un caso in cui non sarebbero necessari i controlli è la chiamata vocale digitalizzata, in cui i ritardi NON sono accettabili e le piccole perdite di dati si.

- **Servizio orientato alla connessione:**

Assomiglia al sistema telefonico. Per poter comunicare con una macchina remota l'utente deve innanzitutto stabilire una connessione, usarla e poi rilasciarla. In questo caso l'ordine dei dati (bits) è quasi sempre rispettato. In alcuni casi, prima di iniziare la comunicazione, il trasmettitore, il ricevitore e la subnet effettuano una sorta di **negoziazione** sui parametri da utilizzare. Molto affidabile. 

- **Servizio senza connessione:**

Assomiglia al sistema postale. Ogni messaggio trasporta l'indirizzo completo del destinatario ed è instradato attraverso il sistema in modo indipendente dagli altri. Non è garantito l'ordine di ricezione dei dati. Un servizio **senza connessione** non affidabile (ovvero privo di conferma) viene detto **datagram**. In alcuni casi è necessario avere una certa affidabilità, in questo caso si parla di **servizio datagram con connessione** (il meccanismo è simile alla raccomandata con ricevuta di ritorno). Un ulteriore servizio è il **request-reply**, dove la sorgente trasmette un singolo datagramma che contiene una richiesta; il messaggio ricevuto rappresenta la risposta (es. ricerca database).

Istintivamente tutti preferiremmo una connessione affidabile, però molte volte essa non è disponibile in quanto costosa da implementare. Ethernet, ad esempio, non costituisce una connessione affidabile in quanto sovente avviene perdita di dati. E' compito dei protocolli degli strati superiori prendersi carico del problema.

## Primitive di servizio

Un servizio è specificato da un insieme di **primitive** (operazioni) che i processi utenti hanno a disposizione per accedere al servizio. Se la pila di protocolli si trova nel sistema operativo, le primitive sono generalmente delle **chiamate di sistema**. Queste chiamate causano la commutazione in modalità kernel, che a sua volta fa prendere il controllo del computer al sistema operativo per spedire i pacchetti necessari. Non tutte le primitive sono chiamate di sistema, in quanto alcune vengono invocate non da processi utente. **Le primitive di un servizio connection oriented sono differenti di quelle di un sistema connectionless.**

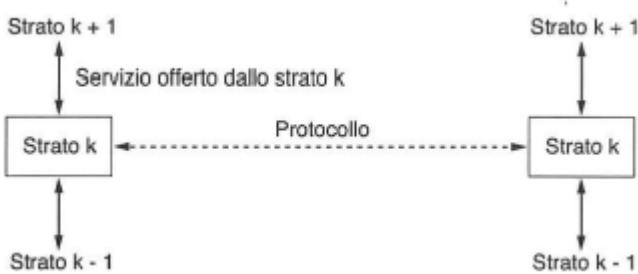
Primitiva	Significato
LISTEN	Attesa bloccante di una connessione in arrivo
CONNECT	Stabilisce una connessione con un pari in attesa
RECEIVE	Attesa bloccante per un messaggio in arrivo
SEND	Manda un messaggio al pari
DISCONNECT	Termina una connessione

Sopra un esempio di primitive.

## La relazione tra servizi e protocolli

- **Servizio:** è un insieme di primitive che uno strato offre a quello superiore. Il servizio definisce quali operazioni lo strato è in grado di offrire su richiesta dei suoi utenti, ma non dice nulla di come queste operazioni sono implementate. Un servizio è correlato all'interfaccia tra due strati, dove quello inferiore è il provider del servizio mentre quello superiore è l'utente.
- **Protocollo:** è un insieme di regole che controllano il formato e il significato dei pacchetti, o messaggi scambiati tra le entità pari all'interno di uno strato. Le entità usano i protocolli per implementare le loro

definizioni di servizi. Sono libere di cambiare i loro protocolli, a patto di non cambiare il servizio visibile agli utenti.



## Il modello di riferimento OSI

Anche se i protocolli ad esso associati sono in disuso, il modello in sé ha un valore generale ed è ancora valido, e le caratteristiche discusse per ogni strato sono ancora molto importanti.

Il modello OSI ha sette strati. I principi che sono stati applicati per arrivare ai sette strati si possono brevemente riassumere:

1. Si deve creare uno strato quando è richiesta un'astrazione diversa
2. Ogni strato deve svolgere una funzione ben definita
3. La funzione di ogni strato va scelta con uno sguardo rivolto alla definizione di protocolli internazionali
4. I confini degli strati vanno scelti per minimizzare il flusso di informazioni attraverso le interfacce
5. Il numero di strato deve bastare per evitare la necessità di radunare funzioni distinte nello stesso strato, ma essere abbastanza piccolo da rendere l'architettura attuabile.

Il modello OSI non è in sé un'architettura di rete, perché non specifica quali sono esattamente i servizi e i protocolli da usare in ciascuno strato, si limita infatti a definire ciò che ogni strato deve compiere.

- **Lo strato Fisico**

Si occupa della trasmissione dei dati grezzi. Deve assicurare che i bit con valore 1 vengano ricevuti con valore 1 e non 0; alcuni problemi tipici riguardano quanti Volt utilizzare per i segnali (es. 1 = corrente, 0 = assenza corrente), quanto deve durare un bit, se la comunicazione può essere bidirezionale o no e tutto ciò che concerne i problemi relativi alla trasmissione. Si parla di interfacce meccaniche o elettriche.

- **Lo strato Data Link**

si occupa di rilevare (ricordiamo che la rilevazione degli errori non è una scienza certa) gli errori di trasmissione, in modo da evitare che tali errori si propaghino verso gli strati superiori. Per cui si occupa di suddividere il flusso in ingresso in **data frame**. Se il servizio è affidabile, il ricevitore conferma la corretta ricezione di ciascun frame restituendo un **acknowledgment frame**. Un altro compito importante è quello di evitare che si intasi la linea, mediante un controllo di traffico. Le reti broadcast in più hanno anche il problema di determinare il momento in cui si può accedere al canale senza disturbare un eventuale utilizzatore.

- **Lo strato network**

Si occupa della qualità del servizio, in quanto controlla il funzionamento della subnet. Risolve problemi quali: la rete ricevente non accetta pacchetti di una certa dimensione, intasamento di pacchetti nella rete. In poche parole cerca di adattare il trasmettitore o il ricevitore all'utilizzo della rete corrente, in quanto esistono diversi modi di inoltrare i pacchetti verso la destinazione (tabelle "cablate" dentro la rete, stabilito ad inizio "conversazione" o dinamicamente a seconda del tipo di pacchetto trasmesso).

- **Lo strato Trasporto**

La sua funzione essenziale è quella di accettare dati dallo strato superiore, dividerli in unità più piccole se necessario, passarli allo strato network e assicurarsi che tutti i pezzi arrivino correttamente all'altra estremità. Lo strato trasporto stabilisce inoltre che tipo di servizio offrire allo strato sessione e, in definita,

agli utenti della rete. Copre tutto il percorso da sorgete a destinazione; in altri termini, un programma sul computer sorgente instaura una conversazione con un programma corrispondente sul computer destinatario, utilizzando intestazioni dei messaggi e messaggi di controllo. 

- **Lo strato Sessione** 

Permette ad utenti diversi di stabilire tra loro una **sessione**. Le sessioni offrono servizi quali: **controllo del dialogo** (tenere traccia dei turni di trasmissione e ricezione), **gestione dei token** (evitare che due macchine tentino di effettuare la medesima operazione nello stesso istante) e **sincronizzazione** (supervisionare una trasmissione in modo da poterla riprendere in caso di crash).

- **Lo strato Presentazione** 

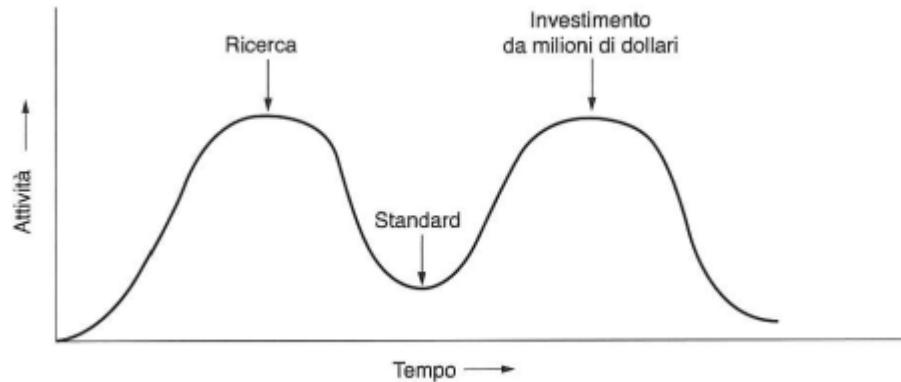
Si occupa della sintassi e della semantica dell'informazione trasmessa. Gestisce le strutture dati astratte che permettono a due computer con diverse architetture di comunicare.

- **Lo strato Applicazione** 

Comprende una varietà di protocolli comunemente richiesti dagli utenti. Un esempio largamente utilizzato è **http (HyperText Transfer Protocol)**.

Ciò che ha decretato la fine di OSI è stato:

1. **Poca tempestività**



Se l'intervallo di tempo tra la ricerca e l'investimento delle aziende è troppo piccolo, chi sviluppa lo standard finisce schiacciato, ed è proprio quello che è accaduto ad OSI. Quando OSI fece il suo debutto, TCP/IP era già largamente utilizzato, per cui è mancata l'offerta iniziale che facesse partire l'utilizzo dello standard; mentre ogni produttore aspettava la prima mossa dell'altro, nessuno è partito e OSI è svanito.

2. **Tecnologia Scadente**

Essendo stata la scelta degli strati più una mossa politica che tecnica, il modello ed i protocolli risultò con molti difetti. Risultando così un modello pesante ed incomprensibile.

3. **Implementazioni Carenti**

Essendo il modello così complesso, le implementazioni e gli aggiornamenti arrivavano con lentezza, per cui venne scavalcato velocemente dal suo concorrente, che al contrario veniva aggiornato velocemente e da un'enorme comunità.

4. **Incapacità Politica**

Ritenuto una creatura del governo, venne subito cassata come una cosa brutta e cattiva (tipo Microsoft vs Linux ai giorni nostri).

## Il Modello di riferimento TCP/IP

**ARPANET** è il progenitore di tutte le reti geografiche. Rete sponsorizzata dalla DoD connetteva centinaia di università e uffici governativi tramite linee telefoniche affittate. Con l'introduzione della tecnologia satellitare e radio è stato molto difficile adattare i protocolli esistenti; dall'esigenza di collegare diverse tipologie di reti tra loro nasce il **modello di riferimento TCP/IP**. Un obiettivo principale era che la rete potesse sopravvivere anche dopo la perdita dell'hardware subnet senza interrompere le conversazioni in corso, ma con il trasmettitore ed il ricevente ancora in funzione. 

A seguire i principali strati:

- **Lo strato Internet** 

Tutti i precedenti requisiti hanno portato alla scelta di una rete a commutazione di pacchetto basata su uno strato internetwork senza connessione (nelle telecomunicazioni la **commutazione di pacchetto** è una tecnica di accesso multiplo a ripartizione nel tempo, utilizzata per condividere un canale di comunicazione tra più nodi in modo non deterministico, specificamente concepita per il trasporto di dati in forma pacchettizzata in reti di calcolatori o più in generale in reti di telecomunicazione). Il suo scopo è quello di consentire agli host di mandare pacchetti in qualsiasi rete, e farli viaggiare in modo indipendente l'uno dall'altro. Questo strato definisce un formato ufficiale per i pacchetti e un protocollo chiamato **IP (Internet Protocol)**; lo scopo di questo strato è di consegnare i pacchetti IP alla destinazione corretta.

- **Lo strato Trasporto**

Progettato per consentire la comunicazione tra entità pari degli host sorgente e destinazione. Sono stati definiti due protocolli di trasporto end-to-end:

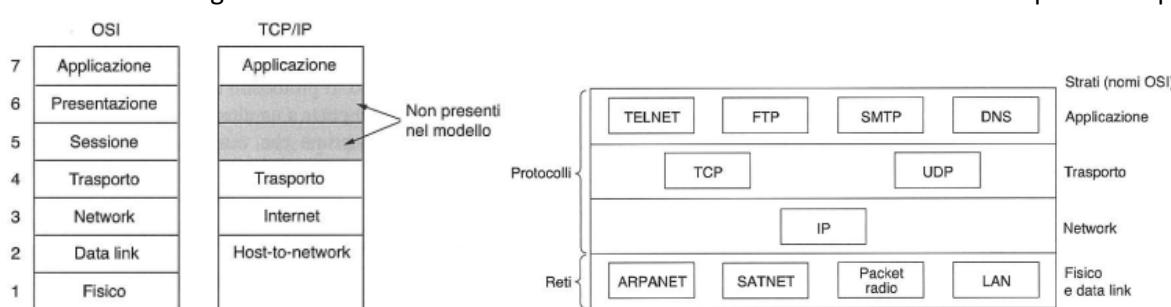
1. **TCP**, è un protocollo affidabile orientato alla connessione che permette ad un flusso di byte di raggiungere la sua destinazione senza errori. Il trasmettitore TCP scomponete il messaggio che viene ricomposto dal ricevente TCP, gestendo anche il controllo di flusso.
2. **UDP (User Datagram Protocol)**, è invece inaffidabile utilizzato per le applicazioni che non necessitano di garanzia di ordinamento e il controllo di flusso, ma che lo gestiscono in modo autonomo. Ottimo per le consegne rapide e lo streaming audio/video.

- **Lo strato Applicazione**

Contiene tutti i protocolli di livello superiore, come: TELNET (terminale virtuale), FTP (trasferimento file), SMTP (email), DNS (corrispondenza tra nome degli host e indirizzi di rete) e molti altri.

- **Lo strato Host-to-network** 

L'host si collega alla rete utilizzando sudetto strato attraverso qualche protocollo.



Anche se largamente usato, TCP/IP ha anche lui dei problemi:

1. Il modello non riesce a distinguere in modo chiaro i concetti di servizio e protocollo.
2. Non è adatto per descrivere pile di protocolli diversi da TCP/IP (es. Bluetooth)
3. Lo strato host-to-network non è un vero strato, inteso con il significato che ha nel contesto dei protocolli stratificati; è piuttosto un'interfaccia tra network e data link. Brutta cosa.
4. Non fa distinzione e neanche menziona gli strati fisico e data link, che di norma sono due cose distinte e separate (il primo si occupa dei cavi di rame il secondo si limita a delimitare l'inizio e la fine dei frame ed a spostarli)
5. Molti protocolli presenti nel modello sono fatti con i piedi, risolvendo problemi ad hoc; per cui si ottiene un modello troppo radicato e difficile da adattare a nuove tecnologie.
6. Il modello praticamente non esiste, mentre i protocolli sono largamente utilizzati.

## Confronto tra i modelli di riferimento OSI e TCP/IP

Entrambi basati sul modello di pila (stack) di protocolli indipendenti, e la funzione degli strati è più o meno simile. Ad esempio in entrambi i modelli gli strati di livello pari o superiore a quello di trasporto forniscono ai processi che vogliono comunicare un servizio di trasmissione end-to-end e indipendente dalla rete.

Nel modello OSI sono presenti tre concetti essenziali:

1. Servizi
2. Interfacce
3. Protocolli

Il suo pregio più importante è proprio la capacità di distinguere nettamente questi concetti. Ogni strato offre un servizio a quello che lo sovrasta; la definizione del *servizio* descrive ciò che fa lo strato, e non le modalità d'accesso da parte delle entità sovrastanti o quelle di funzionamento. L'*interfaccia*, invece, spiega le modalità di accesso ai processi sovrastanti. Infine i *protocolli* pari, impiegati all'interno di uno strato, sono l'*essenza principale*. Lo strato può usare tutti i protocolli che desidera, l'importante è che svolge il suo lavoro. Questa mentalità si sposa bene con il concetto di OOP.

Il modello TCP/IP in principio non faceva queste nette distinzioni, anche se alcuni ci hanno provato. La conseguenza è che in OSI i protocolli sono ben nascosti e sono facilmente sostituibili man mano che la tecnologia si evolve. La capacità di operare cambiamenti di questa portata è uno degli scopi principali dell'esistenza dei protocolli stratificati.

OSI è molto generale come modello, non essendo legato a nessun protocollo; il problema è gli strati sono un po' sparati a muzzo.

Nel caso di TCP/IP prima sono arrivati i protocolli e poi è stato realizzato il modello che descrive semplicemente i protocolli esistenti, per cui è risultata qualcosa di costruita ad hoc ed impossibile da utilizzare in modo generico.

Il modello OSI ha sette strati mentre il TCP/IP solo quattro, entrambi hanno (inter)network, trasporto e applicazione, ma gli altri sono diversi.

Altra differenza è insita nella modalità di comunicazione, orientata o no alla connessione. OSI supporta entrambi i tipi di comunicazione nello strato network, mentre nello strato di trasporto supporta solo la comunicazione orientata alla connessione. TCP/IP ha solo una modalità senza connessione nello strato network, ma supporta entrambe in quello di trasporto, molto importante per i protocolli richiesta-risposta.

## Commutazione di circuito e commutazione di pacchetto

- **Commutazione di circuito**

Nella tecnica detta "a commutazione di Circuito" viene realizzato lunga la rete un percorso riservato tra trasmittitore e ricevitore, rimandando la connessione se ciò non è possibile. Ottenuta la linea sul percorso nessun altro nodo può utilizzare il canale, finché la comunicazione non è terminata (come ad esempio nei collegamenti telefonici "a voce"). In questo modo il singolo colloquio avviene alla **massima velocità**, mentre le prestazioni globali della rete **risentono dei tempi di attesa** necessari per ottenere la connessione sul percorso.

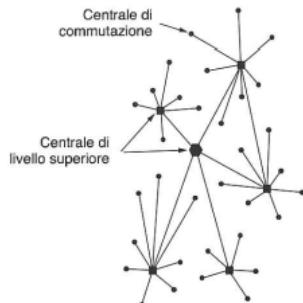
- **Commutazione di pacchetto**

Con questa tecnica il messaggio viene suddiviso in parti di piccole dimensioni, dette **pacchetti**, che vengono inoltrati sulla rete in modo indipendente l'uno dall'altro; ogni pacchetto può seguire un percorso separato, a seconda del maggiore o minor traffico delle varie linee, e giungere al destinatario in istanti diversi. La trasmissione di piccoli pacchetti tutte delle medesime dimensioni consente di ottimizzare l'utilizzo della rete, ottenendo un'alta velocità complessiva. Il protocollo di comunicazione, però, deve prevedere un metodo per la ricostruzione esatta della sequenza di pacchetti in modo da ricomporre il messaggio originario. Questo metodo si è rivelato il più efficiente ed è adottato dalla maggior parte di reti attuali, LAN e WAN.

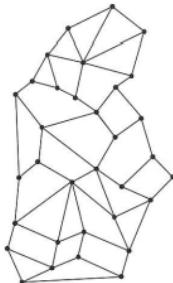
## ARPANET

Un po' di storia:

- Stati Uniti sono alla ricerca di una rete che possa sostituire la tradizionale rete telefonica, in quanto in caso di un guasto di un nodo le comunicazioni risultavano compromesse.



- Paul Baran progetta un sistema a commutazione di pacchetto, altamente distribuito. Purtroppo il progetto non viene approvato dall'AT&T che si rifiuta di costruire un prototipo.



- Nel 1957 nasce ARPA (*Advanced Research Projects Agency*). Nel 1967 ARPA si inizia a dedicare alle reti. 
- Nasce ARPANET di cui a seguire le specifiche.

Il progetto prevedeva una subnet composta da minicomputer chiamati IMP (*Interface Message Processor*) collegati da linee di trasmissione a 56kbps (il meglio che poteva offrire l'epoca); per ottenere il massimo dell'affidabilità ogni IMP doveva essere collegato a sua volta ad almeno altri due IMP, inoltre era basta su datagrammi così in caso di distruzione di un nodo intermedio i messaggi sarebbero stati instradati attraverso percorsi alternativi.

Ogni IMP era collegato, tramite un cavo di corte dimensioni, ad un host che inviava messaggi di massimo 8.063 bit all'IMP, che li avrebbe salvati, suddivisi in pacchetti da 1008 bit massimo e inoltrati. Abbiamo così il primo sistema store-and-forward.

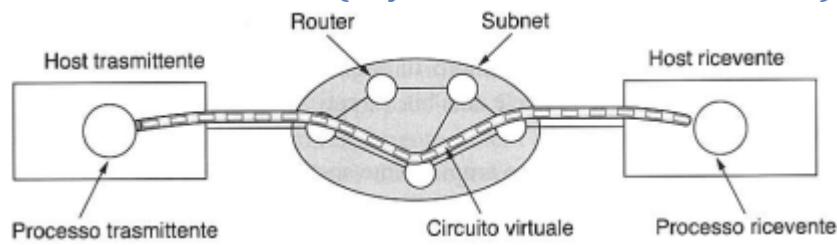
Il software fu suddiviso in:

- **Lato subnet** composto dalla parte connessione IMP-host (da parte IMP), dal protocollo IMP-IMP e un protocollo da IMP sorgente a IMP destinazione progettato per rendere il sistema più affidabile-
- **Lato host** che comprendeva del software di connessione host-IMP (da parte host), il protocollo host-host e il software applicativo.

ARPANET fu fondamentale anche per lo sviluppo del modello TCP/IP, in quanto non era capace di supportare reti multiple, per cui si ritenne necessario progettare nuovi protocolli per le comunicazioni internetwork, che stavano crescendo man mano che ARPANET si espandeva.

Grazie all'uscita di BSD 4.2 con all'interno nuove utility per TCP/IP divenne molto semplice creare reti LAN e aggiungerle ad ARPANET; la rete crebbe a dismisura e negli anni '80 si rese necessario creare il servizio DNS, ovvero organizzare i computer in domini e associare ai nomi degli host gli indirizzi IP. 

## Circuiti virtuali ATM (Asynchronous Transfer Mode)



Sono reti orientate alla connessione, per cui prima di iniziare la comunicazione è necessario inviare un pacchetto speciale che configuri la connessione. Mentre il pacchetto avanza nella subnet, i router lungo il suo percorso aggiornano le tabelle interne per prendere nota dell'esistenza della connessione e per riservare le risorse di cui ha bisogno.



Le connessioni vengono chiamate **circuiti virtuali** e la maggior parte delle connessioni ATM supporta anche **circuiti virtuali permanenti**, che non sono altro che collegamenti virtuali permanenti tra due host remoti.



La comunicazione avviene mediante dei pacchetti fissi detti celle che seguono lo schema seguente



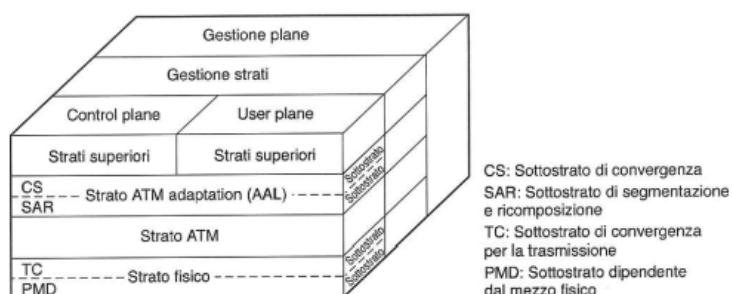
Una parte dell'intestazione è l'**identificatore di connessione**, che serve all'host ricevente, a quello trasmittente e a tutti i router intermedi per identificare le celle che fanno parte di quella connessione.

L'istradamento è effettuato in hardware ad alta velocità, è la dimensione contenuta dei pacchetti aiuta la progettazione di ruoter idonei; i pacchetti IP (che invece sono variabili) devono essere inoltrati via software, mediante un procedimento più lento. Le celle piccole inoltre hanno il vantaggio di impegnare la rete per brevi periodi di tempo.

Tutte le celle seguono lo stesso percorso per arrivare a destinazione e, mentre l'ordine di trasmissione viene preservato, non viene garantita la consegna effettiva; il problema della mancata consegna di un pacchetto è un problema che deve essere gestito dai protocolli di livello superiore.

Le reti ATM sono organizzate come WAN tradizionali, con linee e switch. La velocità varia da 155Mbps a 622Mbps, ma sono supportate anche velocità superiori.

## Il modello di riferimento ATM



CS: Sottostrato di convergenza  
SAR: Sottostrato di segmentazione e ricomposizione  
TC: Sottostrato di convergenza per la trasmissione  
PMD: Sottostrato dipendente dal mezzo fisico

- Strato fisico:** si occupa del mezzo fisico. Le celle ATM devono essere spedite su filo o fibra così come sono, mantenendo così ATM un sistema indipendente dal tipo di mezzo utilizzato. Si suddivide a sua volta in due sotto livelli:
  - PMD (Physical Medium Dependent)** che si interfaccia con il cavo vero e proprio, spostando i bit e gestendo la loro temporizzazione. Esso cambia a seconda del tipo di cavo utilizzato.
  - TC (Transmission Convergence)** che durante la trasmissione delle celle si incarica di mandarle come stringa di bit al sottostrato PMD.

Funge quindi le funzioni di framing che viene svolto dal data link in OSI.

- **Strato ATM:** specifica l'organizzazione della cella e definisce il significato dei campi di intestazione; si occupa inoltre di instaurare ed eliminare i circuiti virtuali e controlla la presenza di congestioni. 
- **Strato AAL (ATM Adaptation Layer):** per evitare che le applicazioni lavorino direttamente con le celle, questo strato si occupa di prenderle e ricomporle (in ricezione) prima di essere inviate all'applicazione, o di scomporle (in invio) prima di spedirle allo strato ATM. A sua volta si suddivide in:
  - **SAR (Segmentation and Reassembly)** che si occupa della segmentazione e ricomposizione vera e propria
  - **CS (Convergence Sublayer)** che permette al sistema ATM di fornire tipi di servizio differenti alle diverse applicazioni.
- **Strato User Plane:** che si occupa del trasporto dati, controllo di flusso, correzione di errore e altre funzioni utente.
- **Strato Control Plane:** che invece si incarica della gestione della connessione.

## Parte 2 - Il livello Fisico

Lo strato fisico definisce le interfacce meccaniche, elettriche e le temporizzazioni di rete. Le informazioni possono essere trasmesse via cavo variando alcune proprietà fisiche, per esempio la tensione o la corrente. Rappresentando il valore di questa tensione o corrente mediante una funzione  $f(t)$  è possibile analizzare matematicamente il comportamento del segnale.

Normalmente come mezzi trasmissivi si usano i **conduttori elettrici**, in modo da utilizzare le onde elettriche come **portanti** per i segnali. E' possibile utilizzare anche mezzi **wireless** per la trasmissione, appoggiandoci alle onde elettromagnetiche. Esistono inoltre anche mezzi **ottici**, che consentono di ottenere canali con larghezza di banda molto elevata. 

### Analisi di Fourier

Una qualsiasi funzione periodica sufficientemente regolare  $g(t)$  con periodo  $T$  può essere ottenuta sommando un numero idealmente infinito di seni e coseni:

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi nft) + \sum_{n=1}^{\infty} b_n \cos(2\pi nft)$$

dove  $f=1/T$  rappresenta la **frequenza fondamentale**,  $a_n$  e  $b_n$  sono rispettivamente le **ampiezze seno e coseno** della  $n$ -esima **armonica**, e  $c$  rappresenta una costante. Questa scomposizione è anche detta **serie di Fourier**.

I coefficienti  $a_n$  e  $b_n$  servono a pesare le varie componenti. Si ricavano dalle seguenti espressioni:

$$a_n = \frac{2}{T} \int_0^T g(t) \sin(2\pi nft) dt$$

$$b_n = \frac{2}{T} \int_0^T g(t) \cos(2\pi nft) dt$$

$$c = \frac{2}{T} \int_0^T g(t) dt$$

Facciamo un **esempio** per capire meglio ciò di cui stiamo parlando: la trasmissione del carattere ASCII "b" codificato in un byte (composto quindi da 8 bit). La sequenza dei bit da trasmettere è 01100010. Il segnale in figura,

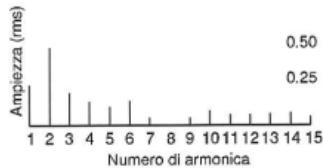
rappresenta la tensione in uscita dal computer trasmittente. L'analisi di fourier di questo segnale produce i coefficienti:

$$a_n = \frac{1}{\pi n} [\cos(\pi n/4) - \cos(3\pi n/4) + \cos(6\pi n/4) - \cos(7\pi n/4)]$$

$$b_n = \frac{1}{\pi n} [\sin(3\pi n/4) - \sin(\pi n/4) + \sin(7\pi n/4) - \sin(6\pi n/4)]$$

$$c = \frac{3}{4}$$

Invece le ampiezze rms (root mean square, valore quadratico medio)  $\sqrt{a_n^2 + b_n^2}$  relative ai primi termini sono visualizzati dalla figura seguente:



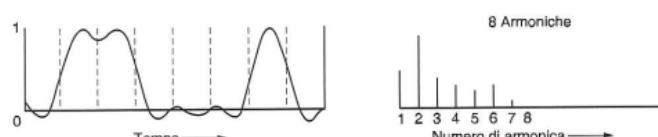
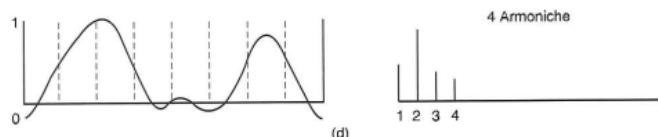
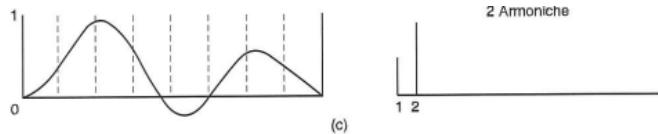
Nessun mezzo di trasmissione è capace di trasmettere segnali senza perdere parte dell'energia durante il processo. Per cui ottenere un segnale come quello dell'onda quadra è assolutamente impossibile, in quanto il segnale non viene attenuato in maniera uniforme, quindi otteniamo una distorsione. Passiamo quindi all'applicazione di Fourier nelle onde reali.

L'intervallo di frequenze trasmesse senza una forte attenuazione è chiamato **banda passante** o più semplicemente **banda**. La banda è una proprietà fisica del mezzo di trasmissione e di solito dipende dalla costruzione, dallo spessore e dalla lunghezza del mezzo. A volte nel circuito viene inserito un filtro per limitare artificialmente l'ampiezza di banda (es. il telefono ha una banda naturale a 1 MHz che viene limitato dalle società telefoniche a 3.100 Hz, in modo da poter fare passare più segnali contemporaneamente nel cavo).

Analizziamo cosa accade se nel segnale preso in esempio, l'ampiezza di banda fosse così scarsa da permettere di prendere solo i primi termini dell'equazione, ovvero solo le frequenze più basse. In figura il risultato:



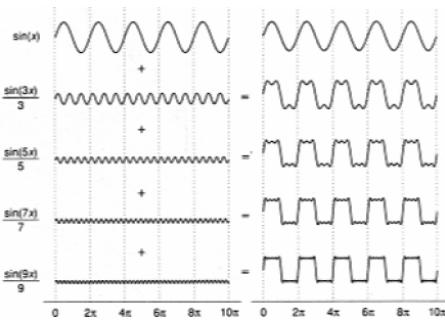
Ovvero avremmo solo il passaggio di un'armonica, quella **fondamentale**, nelle figure seguenti abbiamo cosa accade aumentando l'ampiezza di banda, quindi prendendo in considerazione più termini dell'equazione:



Data una frequenza di bit pari a  $b$  bit/sec, il tempo richiesto per inviare 8 bit, un bit alla volta, è  $8/b$  sec, perciò la frequenza della prima armonica è  $8/b$  Hz.

Tutta questa pappardella serve a farci capire che il limite dell'ampiezza di banda, limita le trasmissioni anche sui canali definiti "perfetti"; esistono comunque degli schemi di codifica che permettono di superare questi limiti.

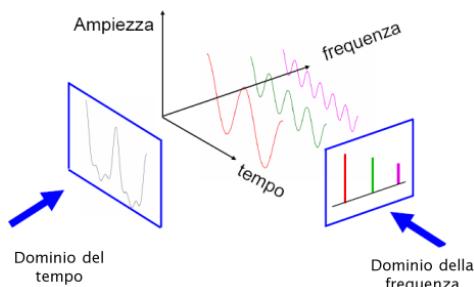
Ecco un altro esempio di quello che accade aumentando l'ampiezza di banda:



Ovvero andando a considerare ampiezze di banda sempre maggiori, si vanno a sommare sempre più coefficienti sen e cos, fino a ottenere l'onda finale.

## Analisi del segnale nel dominio del tempo e delle frequenze

- **Dominio del Tempo** forma che ci è più familiare, in essa appaiono le variazioni subite dal segnale al trascorrere del tempo. Lo strumento più usato e che opera notoriamente nel dominio del tempo è l'oscilloscopio.
- **Dominio della Frequenza** invece analizza le variazioni del segnale rispetto alla frequenza.



## Distorsioni nei segnali

Una distorsione è l'alterazione della forma originale di un'onda. La distorsione di solito è un effetto indesiderato. Essa porta un disturbo nella ricezione di un segnale e quindi ad alterazioni dell'informazione originariamente trasmessa. L'aggiunta del rumore o di altri segnali estranei non sono considerati distorsione, in quanto si sommano con segnale utile, benché gli effetti della distorsione talvolta sono considerati rumore.

Una causa di distorsione sono le interferenze, un modo per risolverle è rivestire il cavo con un sistema (es. gabbia di Faraday) che limita queste interferenze.

## Larghezza di banda

La larghezza di banda è una misura dell'ampiezza di banda dello spettro di un segnale informativo trasmesso o della banda passante disponibile o utilizzata in un canale di comunicazione. La sua importanza nelle telecomunicazioni è legata al fatto che essa è a sua volta strettamente legata alla velocità di trasmissione dei dati.

Nel caso delle comunicazioni analogiche la banda si misura in modo indiretto, ed è data dall'intervallo di frequenze occupato dal segnale (per esempio, una comunicazione telefonica analogica occupa le frequenze che vanno da 300 a 3400 Hz, quindi ha una larghezza di banda di 3100 Hz ovvero la differenza fra 3400 Hz e 300 Hz). Nel caso delle comunicazioni digitali la capacità di trasporto di un canale è misurata in bit al secondo e suoi multipli (kbit/s, Mbit/s, ecc.), e dipende direttamente dalla banda in Hertz del collegamento analogico fisico su cui è realizzata la comunicazione digitale.

## Campionamento e quantizzazione

Dato un segnale periodico, possiamo usare la scomposizione di Fourier per ricostruire il segnale per mezzo delle sue armoniche. Dalle equazioni precedenti si capisce però come è impossibile considerare un insieme di valori infiniti di coefficienti di armoniche, ovvero è impossibile andare a considerare un insieme infinito di armoniche per la ricostruzione del segnale.

Per ottenere un numero finito di valori armonici dobbiamo considerare che lavoriamo con segnali analogici, che vogliamo convertire prima o poi in segnali digitali. Dato un segnale continuo occorre scegliere un numero finito di **campioni** rappresentativi del segnale. Il valore in ogni singolo punto del segnale è un numero reale, occorre scegliere dei **valori discreti** per rappresentare correttamente il segnale.

Ma che **tasso di campionamento** bisogna scegliere?

- a) Un campionamento troppo basso fa perdere dettagli ed informazioni; sebbene grave una tale perdita è spesso una necessità, in quanto non sempre è possibile conservare e maneggiare milioni di campioni.
- b) Un campionamento troppo basso può far apparire dei segnali **non presenti** nell'originale. Il segnale viene alterato. Si parla di **aliasing**. 

Definendo come **Nyquist rate** la più alta frequenza in un segnale continuo e limitato, il **teorema di campionamento di Shannon** afferma:

*Se si raccolgono campioni con frequenza almeno doppia della frequenza di Nyquist (2H nel nostro caso) il segnale può essere ricostruito fedelmente in ogni suo punto.*

Qualunque segnale è caratterizzato da un intervallo di frequenze nel quale sono comprese le frequenze delle sinusoidi che lo descrivono. Esso va sotto il nome di **banda di frequenza** del segnale. Diversi fattori influenzano le caratteristiche della banda:

- Tanto più è breve la durata T del segnale, tanto più è alto il valore della frequenza fondamentale.
- Tanto più velocemente nel tempo varia la  $g(t)$  (funzione periodica dell'onda), tanto più numerose sono le armoniche necessarie a descriverlo

Anche i mezzi fisici sono caratterizzati da una banda di frequenze, detta **banda passante**. Essa rappresenta l'intervallo di frequenze che il mezzo fisico è in grado di trasmettere senza alterarle oltre certi limiti. Le alterazioni principali sono l'attenuazione e l'introduzione di ritardo, che di norma variano al variare delle frequenze trasmesse.

A volte la dimensione della banda passante dipende dalle caratteristiche fisiche del mezzo, a volte deriva dalla presenza di opportuni filtri che taglano le frequenze oltre una certa soglia. Ad esempio nelle linee telefoniche la banda passante è 3kHz ottenuta con un filtro low-pass.

In generale, i mezzi trasmissivi:

- Attenuano i segnali in proporzione alla distanza percorsa e alla frequenza del segnale;
- Propagano i segnali a velocità proporzionali alle loro frequenze.

Una conseguenza è che, per qualunque mezzo trasmissivo, la banda passante si riduce all'aumentare della lunghezza del mezzo stesso. Perché un segnale sia ricevuto come è stato trasmesso, è necessario che la banda passante sia uguale o più ampia della banda di frequenza del segnale stesso. In caso contrario il segnale viene privato di alcune delle sue armoniche (tipicamente le frequenze elevate) e viene quindi distorto. Se un numero sufficiente di armoniche arriva a destinazione, il segnale è comunque utilizzabile. Ci sono due teoremi fondamentali che caratterizzano i limiti per la trasmissione delle informazioni.

## Teoremi di Nyquist, di Shannon ed il rumore.

Ma a quanto può viaggiare massimo un canale?

Il primo di calcolo è di **Nyquist** il quale secondo la sua formula:

$$\text{velocità massima} = 2H \log_2 V \text{ bit/sec}$$

Dove  $H$  è la frequenza massima del canale e  $V$  il numero di possibili valori del segnali (in caso di segnale binario esso è  $V=2$ ).

*Esempio: un canale a 3kHz senza rumore non è in grado di trasmettere segnali binari a velocità maggiore di 6000 bps.*

Fino ad ora non abbiamo tenuto in conto della presenza di rumore, che è **sempre** presente in un sistema. Non tenere in conto il rumore è grave. Il rumore si misura mediante il rapporto **segnalet-rumore**. Data  $S$  la potenza del segnale e  $N$  l'intensità del rumore, otteniamo il rapporto  $S/N$ , rappresentato anche dalla quantità  $10 \log_2 S/N$  misurato in **decibel**.

Shannon tenendo in conto il rumore ed il lavoro di Nyquist ottenne:

$$\text{massimo numero di bit/sec} = H \log_2 \left( 1 + \frac{S}{N} \right)$$

Ovvero la cadenza dati massima in un canale rumoroso di banda  $H$  Hz e di rapporto segnale rumore  $S/N$ . In pratica in situazioni di rumore, se vogliamo comunicare abbiamo due alternative: aumentare l'ampiezza del segnale (aumentando il bit-rate di conseguenza) oppure, se non ciò non è possibile, abbassando il bit-rate. Un esempio è una persona che vuole comunicare in presenza di forte rumore: o parla urlando oppure parla più lentamente.

Il rumore più fastidioso è di tipo **termico**, ovvero un disturbo causato dall'agitazione delle molecole durante il passaggio dell'energia di trasmissione. Per battere il rumore non basta soltanto aumentare la potenza trasmittiva, inoltre man mano che il cavo si allunga aumenta anche la resistenza, quindi disperde energia nell'ambiente.

Inoltre Nyquist non ha sbagliato, in quanto la sua formula è vera.



## Canali ideali, perfetti e canali ideali

I canali trasmittivi si possono suddividere in:

- **Canali Perfetti**, non causano distorsioni o ritardi nella propagazione dei segnali.
- **Canali Ideali**, causano solo un ritardo costante nella propagazione dei segnali.
- **Canali reali**, causano attenuazioni e ritardi in funzione della frequenza di trasmissione.



## La modulazione (ampiezza, frequenza e fase)

Supponiamo di avere un segnale che esprimiamo come  $s(t)$  in funzione del tempo, l'andamento non ci interessa.

Immaginiamo anche di avere una forma d'onda sinusoidale  $p$  chiamata **portante**. Esprimiamo questa portante nella forma:

$$p(t) = A * \sin(\omega t + \varphi)$$

Questa portante è rappresentata in funzione dell'ampiezza ( $A$ ), la fase iniziale ( $\varphi$ ) e la frequenza ( $\omega$ ), inoltre è in funzione del **tempo** perché c'è la  $t$  e la  $\omega$  è costante in questo caso. È una sinusoide che **non** parte dall'origine, ha una certa ampiezza e una certa frequenza. Combiniamo tra di loro  $s(t)$  e  $p(t)$  ottenendo così la funzione  $g(t)$ :

$$g(t) = s(t) \sin(\omega t + \varphi)$$

Il segnale  $g(t)$  si porta d'appresso in qualche modo  $s(t)$ , la portante viene chiamata così proprio perché serve a trasportare, in qualche modo, il segnale  $s(t)$ .

Possiamo anche fare:

$$g(t) = A * \sin(s(t)\omega t + \varphi)$$

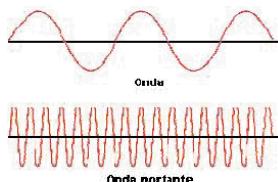
Dove la  $\omega$  non è più costante, ma varia in funzione di  $s(t)$ , anche questa tipologia di portante si porta appresso il segnale  $s(t)$  in qualche modo.

La terza soluzione è invece:

$$g(t) = A * \sin(\omega t + s(t))$$

La portante è sempre la stessa, ma anche qui in qualche modo riesce a portarsi dietro il segnale della  $s(t)$ .

Questi tre metodi vengono chiamati **metodi di modulazione**. Rispettivamente i metodi di **ampiezza**, **frequenza** e **fase**. A cosa ci servono? Vediamo di analizzarli meglio!



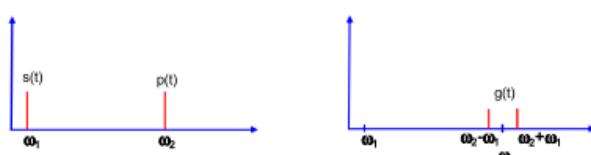
Nel grafico si vede quello che accade nella modulazione d'**ampiezza**, il segnale sinusoidale "entra" nella portante; graficamente si vede, ma matematicamente si vede molto di più! Vediamo come.



Immaginiamo di avere un segnale  $s(t) = \sin(\omega_1 t)$  e una portante  $p(t) = \sin(\omega_2 t)$  sia il segnale risultante  $g(t) = s(t) * p(t) = \sin(\omega_1 t) * \sin(\omega_2 t)$  applicando la formula di Werner otteniamo:

$$g(t) = 1/2[\cos((\omega_1 - \omega_2)t) - \cos((\omega_1 + \omega_2)t)]$$

Graficamente cioè:

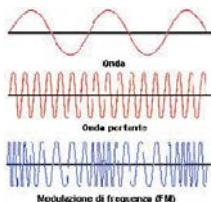


Perché fare "questa roba qui"? ovviamente perché il segnale per viaggiare ha bisogno di usare una portante per usare il canale. Quello che facciamo è "traslare" la frequenza del segnale nella frequenza della portante, che non è altro se non la frequenza in cui opera il canale; uso la modulazione per far sì che il segnale finisca nel mezzo della curva del segnale del canale.

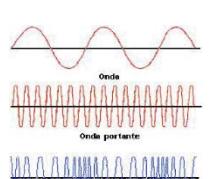
La fibra ottica, ad esempio, passa la luce e non i segnali elettrici, per cui passano quelle frequenze che corrispondono con le frequenze dello spettro solare. Come posso fare? Prendo la mia voce e cerco di modularla con una portante, in questo caso un raggio luminoso, in modo da poter fare viaggiare il mio segnale in un raggio luminoso. In questo caso la portante è fondamentale.



La modulazione di **frequenza** invece causa un fenomeno come in figura:

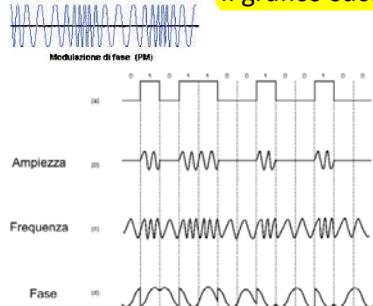


Modulando appunto la frequenza dell'onda. Nel caso della modulazione di frequenza le curve risultanti dalle operazioni applicate in precedenza sono un po' "allucinanti" (rif. Curve di Bessel).



Nella modulazione di **fase** invece sembra molto simile, se non identico, alla modulazione di frequenza, ma sono diverse.

Il grafico successivo sottolinea queste differenze:



Si tratta della trasmissione di segnale binario. Nella linea a) vediamo l'onda quadra che trasmette il segnale originale. Nella linea b) vediamo la modulazione d'**ampiezza**, nella linea c) abbiamo la modulazione di frequenza, nella linea d) la modulazione di fase. In questo caso la modulazione di fase è molto visibile, in quanto il segnale non è regolare continuo, come nel grafico precedente, ma discontinuo.

Purtroppo le somiglianze tra frequenza e fase ci sono, in quanto entrambe sono legate da operazioni di integrazione e derivazione, per cui posso combinare ampiezza con fase e frequenza, in modo da poter portare due segnali contemporaneamente, ma non posso in alcun modo utilizzare fase e frequenza nello stesso momento.



## Segnale Analogico o Digitale

La televisione, il fax e molte altre cose una volta erano in analogico. Ora molte di queste cose sono in digitale. Ma quale dei due è meglio, analogico o digitale? Non è vero che il digitale è meglio dell'analogico, sono due campi di applicazione completamente differenti, ma in molte cose l'analogico è molto superiore. Non esiste un analogico o un digitale puro.

Costruiamo un sistema di comunicazione molto banale, bit 1 e bit 0, se arriva qualcosa di impreciso bisogna decidere quale dei due segnali sia, in quanto il segnale che arriva DEVE essere uno dei due segnali. Con il fatto che io do solo due possibilità, il rumore ad ogni passaggio viene azzerato; questo è il buono del digitale, ogni volta che effettuo una copia del segnale, la piccola quantità di rumore presente viene eliminata. Ovviamente se il rumore è intenso esso verrà trattato come segnale, per cui riprodotto nella copia.

Una grandezza è detta **digitale** se può assumere solo valori ben precisi in un range **finito**. Ha una larghezza di banda più elevata, ma può essere rigenerato con estrema precisione.

Una grandezza è detta **analogica** se può variare senza soluzione di continuità. Utilizza una larghezza di banda più limitata, ma subisce distorsioni per ogni processo rigenerativo.

## I mezzi trasmissivi

I segnali che viaggiano in questi cavi, sono tutte onde elettromagnetiche, ma che hanno comportamenti diversi nei vari mezzi.

Il primo sistema di comunicazione è stato il **doppino telefonico** perché era già presente; è formato da due cavi che sono intrecciati tra di loro. Sono intrecciati perché formiamo una spira, i due cavi così attraggono correnti elettromagnetiche opposte che così facendo si annullano, si riducono i disturbi. Le spire devono essere il più fitto possibile in modo da eliminare tutte le possibili differenze tra le spire dei due cavi, in modo che entrambi attraggono la stessa quantità di energia elettromagnetica e non causano disturbo. Esistono cavi: **CAT3, CAT5, CAT5E, CAT6,...** che si differenziano tra di loro per la distanza tra le spire e la qualità con cui sono costruiti.



(a)

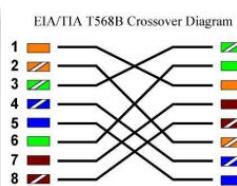
In figura il cavo a) è di CAT3 (quello telefonico), mentre il cavo b) è di CAT5.



(b)

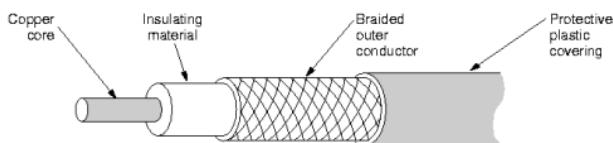
Per capire di che tipo è un pezzo di cavo si legge sul cavo. Perché ci interessa la categoria? La tabella successiva più o meno lo spiega, dove nella prima riga è sottolineato l'andamento della frequenza del cavo man mano che si cambia categoria (CAT).

TIA/EIA-ISO/IEC	CAT5e—Class D	CAT6—Class E	CAT6a—Class E <sub>a</sub>	CAT7—Class F	CAT7a—Class F <sub>a</sub>
Frequency (MHz)	1–100	1–250	1–500	1–600	1–1000
Insertion Loss (dB)	24.0	21.3–21.7	20.9	20.8	20.3
NEXT (dB)	30.1	39.9	39.9	62.9	65.0
PS-NEXT (dB)	27.1	37.1	37.1	59.9	62.0
ACR (dB)	6.1	18.6	18.6	42.1	46.1
PS-ACR (dB)	3.1	15.8	15.8	39.1	41.7
ACRF (dB)	17.4	23.3	23.3–22.5	44.4	47.4
PS-ACRF (dB)	14.4	20.3	20.3–22.5	41.1	44.4
Return Loss (dB)	10.0	12.0	12.0	12.0	12.0
PS-ANEXT (dB)	—	—	60.0	—	67.0
PS-ACRF (dB)	—	—	37.0	—	52.0
Propagation Delay (ns)	548	548	548	548	548
Delay Skew (ns)	50	50	50	30	30
Networks Supported	1000BASE-T	1000BASE-TX	10GBASE-T	N/A	N/A



Questo è il tipo di connettore utilizzato in questi tipi di cavo, chiamato **RJ45** (l'RJ11 è quello proprio telefonico). Sono collegati in coppie; abbiamo in totale quattro coppie; questo connettore è standardizzato.

In passato veniva utilizzato il **cavo coassiale** formato come figura.

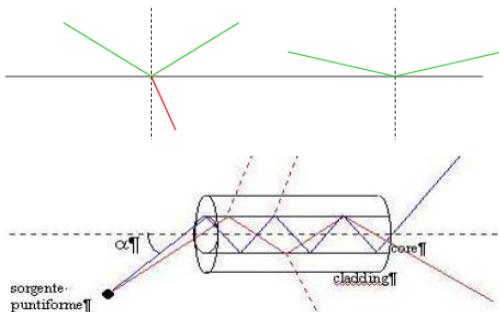


Il cavo coassiale **riesce più a difendersi dalle interferenze elettromagnetiche**. Una volta era lo standard di Ethernet, in seguito non si usò più.

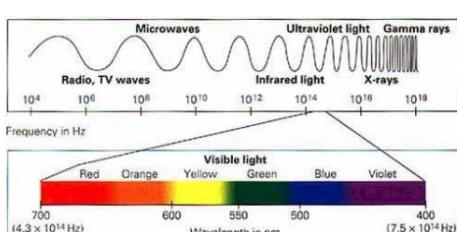
La **fibra ottica** è il metodo di comunicazione del futuro.



Lezione di fisica: data una superficie di separazione tra due mezzi ottici con caratteristiche differenti, se abbiamo un raggio luminoso incidente, che tocca questa superficie, il raggio viene scomposto in componente **riflessa** e componente **rifratta**. Oltre un certo angolo di incidenza, la componente **rifratta è nulla** e la componente **riflessa è totale**, tranne quella che viene assorbita dal materiale. Su quest'ultima proprietà si basa la fibra ottica.



Se avessi una struttura tubolare formata da un vetro, che è un liquido ricordiamo, anche molto sottile da essere duttile, allora otteniamo un mezzo di comunicazione che funziona come la figura seguente.



Perché andare a scomodare la fibra ottica?

Perché intanto **riusciamo ad ottenere una frequenza di trasmissione mostruosa**, come riusciamo a vedere dalla figura con lo spettro luminoso. La seconda motivazione è che **il cavo può essere messo a bollire, ma non succede nulla essendo fatto di vetro**; è sempre presente l'agitazione termica delle molecole, ma in questo caso ha un incidenza sul segnale quasi nulla. I teoremi di Nyquist e Shannon ci confermano che andiamo spettacolari, proprio per quella  $H$  che rappresenta l'ampiezza di banda, nei loro teoremi. Infine le Interferenze elettromagnetiche non esistono, completamente insensibile.

La fibra ottica è eccezionale perché lavora proprio in quelle frequenze. Ricordiamo che la **velocità della luce nella fibra ottica non è la velocità della luce nel vuoto**.

Perché allora non è ampiamente utilizzato?

1. Purtroppo costa ancora troppo passarla, ovvero effettuare gli scassi per far passare il cavo.
2. Quando bisogna effettuare un collegamento tra due fibre, bisogna tagliarle e poi riunirle in maniera abbastanza artificiosa.

Nelle giunzioni sempre qualcosa si perde, ma abbiamo quella frequenza mostruosa che ci permette di perdere qualcosa. Il rivestimento deve essere molto resistente, per evitare che si possa danneggiare la fibra interna che è molto sottile e fragile.

Esistono fibre ottiche:

- **Multimodali** ovvero che permettono al raggio più riflessioni multiple. Il raggio luminoso non arriva ben definito a destinazione, poiché non è proprio un raggio ma un fascio, questo fascio man mano che arriva si allarga sempre di più, ottenendo così un cono. Il raggio quindi si scomponete in varie componenti che possono anche annullarsi tra di loro se sono in opposizione di fase, creando zone di "buio". Per cui in questo tipo di fibra possono esserci problemi.
- **Monomodali** ovvero che la luce avanza senza riflessioni, assialmente. Arriva più energia a destinazione e coprono maggiori distanze. Sono anche più costose.

I collegamenti tipici sono:

- Punto – punto unidirezionale
- Ad anello. Sono tutti collegamenti unidirezionali, ognuno prende, legge e passa avanti.
- A stella passiva. Si collegano al centro con una palla di vetro, il raggio arriva e finisce in tutte le diramazioni, ottenendo così una riflessione non controllata. Abbiamo una perdita di informazioni allucinanti, per cui non si usa tantissimo.

Le fibre di vetro, all'interno di un cavo fibra ottica, sono intrecciate tra loro per un motivo di resistenza meccanica, infatti in questo modo riescono a sopportare sollecitazioni meccaniche abbastanza forti.

Se creo una curva stretta con il cavo posso però avere dei grossi problemi, in quanto il raggio potrebbe non riuscire a seguire la curvatura e così ottenere rifrazione al posto della riflessione.

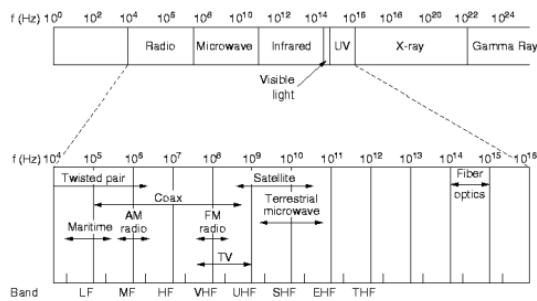
Si utilizza una sorgente **laser** per trasmettere, in modo da concentrare la sorgente luminosa ed evitare possibili dispersioni di energia. Con il led ad esempio, il diametro del fascio sarebbe superiore al diametro della fibra (di gran lunga superiore) per cui, la luce la vediamo dall'altra parte del capo del filo, ma abbiamo un'enorme dispersione energetica. La motivazione per cui il laser (*Light Amplification by Stimulated Emission of Radiation*) è più adatto è data dal fatto che la luce laser ha tutte le onde in fase fra di loro, quindi il fascio viene concentrato puntiforme; si dice che **il laser emette luce coerente**. Il neon, ad esempio, non ha le onde in fase, in quanto il gas presente dentro la lampada ha tutte le molecole che sono per i fatti propri, chi emette o chi non emette, per cui non abbiamo un fascio di luce coerente. Si potrebbe risparmiare di più utilizzando il **led**, ma per poterlo usare avremmo bisogno di un ulteriore dispositivo detto **lente** che concentra il fascio luminoso; a fronte degli enormi costi del cablaggio ottico, spendere 10€ per un laser ottimo non è un problema.

La fibra ottica si utilizza sempre a coppie, upstream e downstream; si potrebbe ideare un mezzo broadcast con un'unica fibra, ma non ne vale la pena. Si predilige il collegamento punto-punto visto prima.

Anche il cavo a fibra ottica si suddivide in CAT a seconda degli intrecci; il connettore inoltre è il punto critico del sistema perché:

- 1) i cavi non sono intrecciati per cui possono esserci interferenze
- 2) avviene proprio il contatto, per cui possono esserci dispersioni in collegamento tra un sistema all'altro. Il miglior conduttore è placcato in oro, in quanto è molto resistente all'ossidazione. Si fa in tutti i modi di diminuire le resistenze, in quanto se si riscalda perdiamo il segnale. Il connettore è una **resistenza** vera e propria.

## Trasmissioni ad onde radio



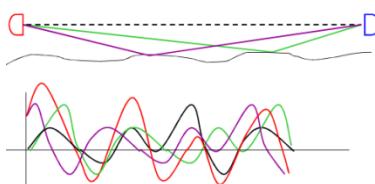
Le onde in natura sono suddivise come in figura; fino a  $10^4$  abbiamo le onde meccaniche, che non vengono utilizzate per trasmettere in quanto avrebbero bisogno di antenne spaventose (friggipiccioni) molto dannose.

Le trasmissioni a **bassa frequenza** vengono usate principalmente dalle navi, perché? La ionosfera che ci protegge, le fa rimbalzare, per cui seguono la curvatura terrestre. Molto utili se non è possibile posizionare ripetitori. L'esperimento di Marconi (Inghilterra-Nuova Zelanda) era importante, proprio perché dimostrò che il segnale è capace di propagarsi per tutto l'emisfero. Bellissimo! Però siamo sempre in bassa frequenza, per cui non possiamo farci più di tanto!

Man mano che saliamo di frequenze gli ostacoli iniziano a farsi sentire, le distanze iniziano ad accorciarsi per cui le curvature della terra non possono essere superate.

Nel 1976 nascono le prime radio indipendenti, perché è importante ciò? All'inizio c'era il monopolista che comunicava in **onde lunghe** e in **onde medie** e piazzava le antenne ripetenti ogni 700km in quanto coprivano tantissimo spazio. Fino ad allora le radio libere non erano possibili in quanto c'erano casini per suddividersi le frequenze; si cominciarono quindi a svilupparsi tecniche sulle **onde corte**, che davano molti meno problemi in quanto gli ostacoli naturali limitavano considerevolmente il raggio di azione. Si poteva trasmettere in tanti senza disturbarsi, l'importante è che ci fossero ostacoli.

Salendo ancora di più troviamo le **microonde**, esse vengono assorbite dall'acqua che si agitano (principio del forno a microonde). In Sicilia esiste un ponte a microonde, che non funziona quando piove, in quanto la pioggia assorbe le onde. Le microonde viaggiano in maniera rettilinea e possono essere facilmente direzionate. Le antenne possono essere molto corte, in quanto la lunghezza d'onda è nell'ordine del centimetro; il paraboloide serve semplicemente per concentrare il segnale, e posizionando l'antenna nel fuoco riusciamo a creare un ottimo tunnel di comunicazione posizionando le antenne a coppie. Concentrando i raggi è possibile migliorare il rapporto S/N, per cui migliorare la comunicazione in maniera considerevole. Purtroppo man mano che si sale di frequenza gli ostacoli diventano critici, tra cui anche la stessa ionosfera che diventa un vero e proprio filtro. Altro problema delle microonde è il **multipath fading**.



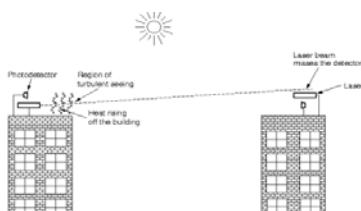
Il raggio, come si vede in figura, potrebbe suddividersi in raggi che seguono un loro percorso. Questi raggi sfasati possono arrivare in ritardo: se il ritardo è nell'ordine della lunghezza d'onda, allora il raggio in ritardo arriva in **opposizione**, per cui annullerebbe completamente il segnale "buono"; se il ritardo è minore allora problemi non ce ne sono.

Saliamo ancora di frequenze e arriviamo agli **infrarossi**. Abbiamo due tipi:

- **bassissimo bit-rate**
- **bit-rate medio elevato**

il problema di queste comunicazioni (applicati alle telecomunicazioni) è che i due dispositivi devono "vedersi" necessariamente, la velocità è molto scarsa e infine non possono essere utilizzati all'aperto, in quanto il sole provoca troppo rumore di fondo essendo una grossa fonte di infrarossi. I dispositivi a bassissimo bit-rate sono largamente utilizzati nei telecomandi, in quanto i problemi visti in precedenza non si pongono; largamente utilizzati perché gli infrarossi vengono bloccati dai più piccoli ostacoli, per cui non possono esserci interferenze tra telecomandi. Inoltre sono antenne che si comprano al kilo a prezzo veramente irrisorio.

Salendo ancora abbiamo le **onde luminose**, già solo con la frequenza, come abbiamo visto in precedenza, Shannon e Nyquist ci dicono che andiamo spettacolari. Valgono tutte le considerazioni fatte con la fibra ottica.



E' possibile trasmettere con il laser senza fibra ottica? Si, il sistema funziona abbastanza bene, l'unico accorgimento è che i trasmettitori laser devono essere puntati bene altrimenti non si "vedono". Altro problema può essere causato dal sole, che provoca un effetto lente (come in figura) che "distorce" il raggio laser facendolo puntale altrove.

## Le reti telefoniche (sistemi a commutazione di circuito)

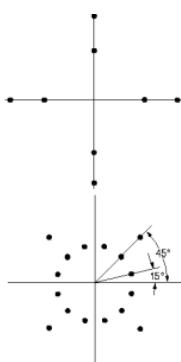
Le reti telefoniche sono un sistema di comunicazione a livello fisico già pronte e disponibili, per cui non conviene passare la fibra ottica; se necessità una nuova cablatura conviene fare passare la fibra. Le corde vocali generano circa 12000Hz, il telefono invece fa passare solo le frequenze da 0 a 4000 Hz, sicuramente abbiamo una perdita di segnale, infatti la voce la sentiamo più cupa, però questo segnale è sufficiente per effettuare la comunicazione. Infatti a volte i familiari si confondono. Conviene fare questo taglio.

Il **tasso di errore** è la quantità di bit che arriva errata su un tot di bit, nella linea telefonica è dell'ordine di  **$1/10^5$**  mentre i cavi tradizionali arrivano anche a  **$1/10^{13}$** , sono probabilità scarse ma bisogna tenerne conto.

Le reti telefoniche sono state pensate per utilizzare segnali analogici, possono passare bit? Si, però non passa niente, perché quel filtro da 0 a 4000 Hz farebbe passare così poche frequenze che non comunicheremmo più; inoltre il filtro è da 300 a 3000Hz in pratica, dove da 0 a 300 Hz c'è una parte ascendente (a 0 non passa niente, a 300 passano tutte e da 0 a 300 qualcuna passa altre no), da 3000 a 4000 Hzabbiamo l'opposto, ovvero una parte discendente. Questa cosa è stata ideata in quanto otteniamo una curva, eliminando un problema che vedremo in seguito. Con questo canale secondo Nyquist possiamo far passare molto poco, alternativamente possiamo usare uno schema di modulazione 1 alto 0 basso, ma anche qui possono passare massimo 3000 bps, che è ben poca roba.

## I modem analogici - la modulazione

Altrettivamente possiamo usare un **sistema di modulazione**, quindi usare una portante, modularla e mettere i nostri bit su questa portante. La portante deve essere per forza compresa tra 300 e 3000Hz. Possiamo usare due tipi di modulazione differenti, in **ampiezza** e in **fase**.



Definiamo 8 punti (senza di 2), ad ogni punto (campione) possiamo associare 3 bit ( $2^3=8$ ). Avrò due ampiezze separate e quattro livelli di fase differenti, così riesco a distinguere questi 8 tipi di punti. Voglio fare di più? Nello schema successivo ho 16 punti, quindi 4 bit. Le ampiezze sono due (il cerchio iniziale il primo e quello esterno il secondo), le fasi invece sono 12. Possiamo vedere che con un minimo sforzo potremmo passare ad una combinazione a 24, ma non ci conviene in quanto non è potenza del 2, quindi si passa direttamente a 32.

Questi schemi sono chiamati **costellazioni**; per comunicare bisogna mettersi d'accordo con l'altra parte di quale costellazione usare per poter comunicare, in quanto ogni costellazione è collegata ad una velocità. Ogni "puntino" mi rappresenta dal lato DLL, ad esempio, una frequenza di 6 bit, dal lato fisico una certa ampiezza e una certa fase. Il protocollo ci dice quale costellazione associare e la velocità massima. Il protocollo **V90** era quello più veloce ed i modem per un periodo venivano fatti tutti secondo questo protocollo. I FAX usano questo sistema. Il modem ogni volta, prima di connettersi, emetteva un fischio puro che era la portante a 2100Hz e poi una serie di rumori. Durante questi rumori il sistema stava applicando Shannon, siccome non poteva fare i calcoli si ricavava il rapporto S/N facendo prove di trasmissione; il sistema provava a chiamare finché l'altro capo non rispondeva. Il FAX fa la stessa cosa, ma solo che non fa tutto questo casino per non tediare l'utente.

Come si può andare più veloci con una così linea così lenta? Entra in gioco la **compressione dati**, in quanto il tempo perso a comprimere, trasmettere e decomprimere è inferiore a quello di inviare i dati così come sono. Facciamo due calcoli:

Ipotizziamo di voler trasmettere 100000b a 28800b/s, senza compressione passano in 3,47s. Introduciamo ora un fattore di compressione (che ricordiamo è applicabile solo se c'è ridondanza nell'informazione da trasmettere). Supponiamo che impiego 0,2 secondi per comprimere/decomprimere in 50000b per cui avremo:

$$\text{Tempo Tot} = 0,2 + (50000/28800) + 0,2 = 2,2\text{s} \Rightarrow 100000b/2,2\text{s} = 45400b/s$$

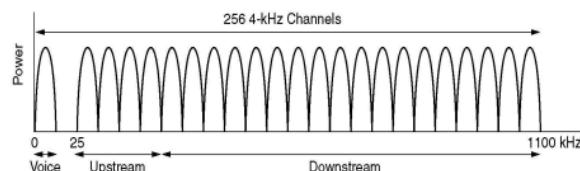
Dal risultato possiamo affermare che **ci conviene comprimere**.

Ovviamente il nostro è un calcolo teorico, però comunque il guadagno è innegabile anche nella realtà.

Il cavo telefonico, se sono vicino alla centrale, può arrivare anche a 50Mb/s, se sono ad un 1,5km allora posso arrivare anche a 10Mb/s. Ma allora perché non ci vado così veloce? Perché la Telecom ha messo il filtro, togliamo il filtro e vediamo che accade.

## Linee ADSL e xDSL

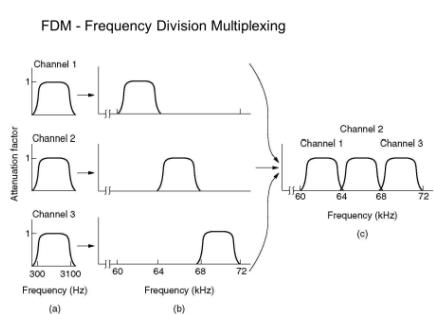
Nasce la **DSL**. Suddividiamo da 0 a 1100kHz in slot 4kHz ciascuno. Il primo lo lascio per la linea telefonica, in modo che sia pulita, lascio un po' di spazio per evitare disturbi; il resto dei canali li lascio per il passaggio dati come se fossero tanti modem tradizionali collegati in parallelo.



Se uno viaggia a 28800b/s e ne metto un centinaio in parallelo riesco ad arrivare anche a 2.28Mb/s. Ho superato il limite senza cambiare mezzo trasmissivo. Alcuni canali li dedico all'upstream e altri al downstream. Se il numero di canali in upstream è **minore** rispetto a quelli downstream allora ho l'**ADSL (Asymmetric Digital Subscriber Line)**, se invece è **bilanciato** allora ho la **XDSL**, che viene normalmente adoperata dalle aziende. Se voglio andare più veloce devo usare una costellazione migliore e una linea più pulita, il rapporto 1/10^5 non mi va più bene. Per cui quando si installano le linee le compagnie assegnano le coppie di cavi migliori alle aziende, in quanto necessitano una linea molto performante. Il **Filtro ADSL** è un condensatore a cui viene attorcigliato un filo di rame intorno, che serve solo per il telefono, ovvero il primo slot; al modem arriva la linea diretta senza filtro. In teoria servirebbe un unico filtro collegato a monte dell'impianto.

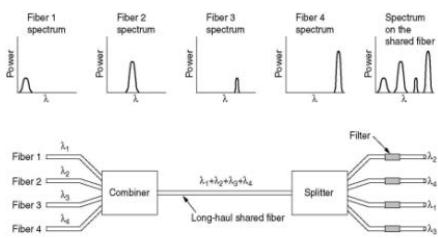
## Tecniche di Multiplexaggio: FDM, WDM, TDM, CDM

In principio Telecom, prima dell'avvento dell'ADSL, utilizzava tutta la banda possibile per far passare le telefonate modulando le frequenze delle varie comunicazioni, dando ad ognuna una frequenza differente. Veniva fatto un **multiplexaggio sulle frequenze (FDM)**, dove il segnale arrivato in centrale veniva separato. In alcuni punti l'ADSL ancora non è arrivata, in quanto c'è ancora un unico cavo per far passare le telefonate, per cui non è possibile attuare il metodo DSL.



Il multiplexaggio è possibile anche effettuarlo su base temporale, mediante il **multiplexaggio TDM**; ovviamente i tempi di turnover devono essere molto stretti.

Si può effettuare il TDM e l'FDM contemporaneamente, questo sistema viene chiamato **GSM**.



Il multiplexaggio con le fibre ottiche è la **WDM** che funziona mediante l'assegnazione dei colori ai vari segnali (mediante la teoria della scomposizione del prisma), solo che questo sistema non è molto utilizzato, in quanto l'ampiezza di banda nelle fibre ottiche è così elevata, che non è necessario attuare un multiplexaggio.

## Parte 3 – Data Link Layer

Il livello DL si occupa di molte cose. I suoi compiti principali sono:

- raggruppare i bit dal livello fisico in modo da formare **pacchetti (framing)**
- gestire l'**accesso al mezzo fisico** nel caso di reti broadcast, mediante il sottolivello MAC (*Media Access Control*)
- fornire un **recapito affidabile**, se richiesto. Recapito affidabile significa che arriva il pacchetto e lui conferma l'arrivo; alcuni sistemi per certi versi se ne fregano.
- Gestiscono gli **errori**. E ci sono due modalità: capire se ci sono stati errori oppure nelle situazioni più particolari andare a correggerli.
- Regolare il **flusso** dei dati tra sorgente e destinazione

Il DLL va da una macchina alla macchina successiva, non direttamente alla macchina di destinazione. Alla destinazione ci arriva con tanti salti; dipende comunque con il mezzo fisico. Se quest'ultimo permette un collegamento diretto allora bene, ma il più delle volte bisogna passare attraverso più macchine (e quindi tra diversi DLL).

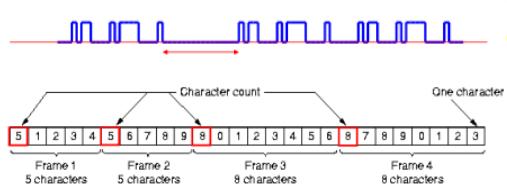


La maggior parte dei protocolli DLL è implementata nella scheda di rete, ovvero l'utente non dovrebbe poter cambiare nulla.

### Framming dei dati



Supponiamo di avere una fibra ottica e supponiamo di avere una codifica semplicissima, luce = 1 e assenza di luce = 0. Nella figura seguente abbiamo uno strano problema, la lunga assenza di luce è: il trasmettitore spento, una lunga



sequenza di zeri oppure altro? Abbiamo un'ambiguità abbastanza pericolosa. Questo sistema di codifica non va bene, in quanto ci mette un dubbio pericoloso; una soluzione a questo problema esiste e sarebbe quello di escludere che il trasmettitore possa essere spento e di inviare, prima di ogni frame, un character count che specifica quanto sarà lunga la frame che sta per arrivare. Ma come fa il sistema a capire

qual è il carattere di controllo? E' fondamentale avere un ottimo sincronismo, altrimenti perdiamo il conto e confondiamo caratteri di controllo e frame. In questo metodo introduco molta ridondanza, in quanto ai dati normali aggiungiamo anche i dati dei caratteri di controllo. Questa è la tecnica a **tre livelli** e la ridondanza pesa circa un 50% di velocità di connessione.

0110111111111111100010

Spezziamo le sequenze di bit 1 con più di cinque 1 consecutivi, in mezzo inseriamo uno zero.

011011110111101111100010

In ricezione se trovo cinque 1 consecutivi e uno 0 allora elimino quest'ultimo, se invece non

lo trovo ma trovo un 1 allora sono a fine frame. A meno che ci sia un errore, allora posso scambiare dati per fine frame, in tal caso quel pezzo lo butto per intero, solo che in questo caso posso recuperare il sincronismo senza problemi. Capito il truccetto posso sempre modificare il protocollo per evolverlo; in questo ultimo metodo introduco pochissima ridondanza. Questa è invece la tecnica a **due livelli** e la ridondanza pesa circa il 20% di velocità di connessione.

## Codifica 4B5B

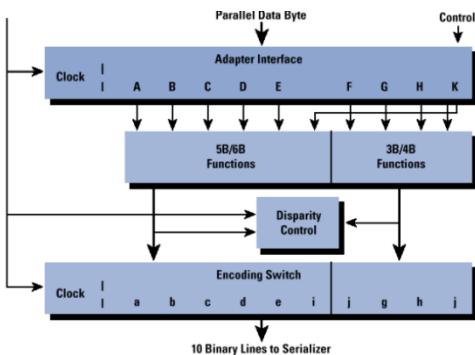
Nome	4B	5B	Descrizione
0	0000	11110	hex data 0
1	0001	01001	hex data 1
2	0010	10100	hex data 2
3	0011	10101	hex data 3
4	0100	01010	hex data 4
5	0101	01011	hex data 5
6	0110	01110	hex data 6
7	0111	01111	hex data 7
8	1000	10010	hex data 8
9	1001	10011	hex data 9
A	1010	10110	hex data A
B	1011	10111	hex data B
C	1100	11010	hex data C
D	1101	11011	hex data D
E	1110	11100	hex data E
F	1111	11101	hex data F
I	-NONE-	11111	Idle
J	-NONE-	10000	SSD #1
K	-NONE-	10001	SSD #2
T	-NONE-	01101	ESD #1
R	-NONE-	00111	ESD #2
H	-NONE-	00100	Halt

Abbiamo un convertitore, alias la tabella seguente. Essa ad ogni possibile combinazione a 4 bit ne assegna una a 5 bit. Sapendo che 4bit = 16 combinazioni, 5bit = 32 combinazioni, allora ci rimangono 16 combinazioni per fare quello che vogliamo, tra cui anche segnalazione. Nella tabella abbiamo un esempio di cosa intendiamo, abbiamo ovvero una vera e propria tabella di conversione. In ricezione faccio l'operazione opposta. Se non riesco a fare la conversione allora evidentemente c'è un errore. Implementazione in hardware di un microcircuito molto semplice, per cui a livello ingresso DLL ho 100 su livello fisico ho 125; per cui ho due velocità di comunicazione quella a livello DLL e quella di segnalazione che tiene conto della ridondanza. In questo caso è determinabile come  $\frac{1}{4}$  in più. Questa è una delle codifiche usata in fibra; questo spreco di  $\frac{1}{4}$  è pesante, ma la fibra ci perdonerà questo spreco viste le frequenze a cui viaggia.

## Codifica 8B10B

Funziona alla stessa maniera della 4B5B, ed è utilizzata da molti standard quali: PCI Express (<3.0) IEEE 1394b (Firewire), Serial ATA, Fibre Channel, Gigabit Ethernet (non tutte le versioni), DisplayPort Main Link, DVI e HDMI, USB 3.0.

E' elettricamente neutra ovvero il numero di bit 1 viene mantenuto uguale a quello di bit 0 in trasmissione nel canale, che cosa significa questo? Supponiamo di inviare una lunga sequenza di bit 1 equivalenti a +5volts, alla fine squaglierebbero il cavo. Avere una sequenza bilanciata di cariche allunga la vita del mezzo trasmittivo; questo



sistema viene implementato mediante un codificatore non troppo complicato. Si prendono 8 bit, i primi 5 li passo ad un convertitore (una tabella) da 5 a 6, gli ultimi 3 invece hanno un trattamento più particolare: avendo a disposizione una tabella 3B4B con più scelte (come quella in figura) se nella conversione della quinina precedente risultiamo avere più bit 1 o 0 (quindi uno squilibrio di cariche), si sceglie l'equivalente 4bit che possa equilibrare la scompensazione. Il bit  $k$  è un ricordo di quello che è accaduto nel passaggio precedente, in quanto non è sempre possibile equilibrare le cariche nello stesso passaggio, per cui potrebbe essere

necessario effettuarla nel passaggio successivo.

Ricapitolando quindi alla fine di un passaggio uno squilibrio può essere invertito (da 1 passa a 0 o viceversa), oppure annullato e per poter fare ciò è necessaria una memoria rappresentata dal bit  $k$ .

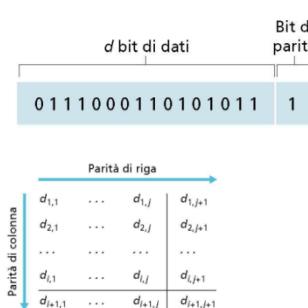
3b Decimal	3b Binary (HGF)	4b Binary (fghi)
0	000	0100 or 1011
1	001	1001
2	010	0101
3	011	0011 or 1100
4	100	0010 or 1101
5	101	1010
6	110	0110
7	111	0001 or 1110 or 1000 or 0111

5b Decimal	5b Binary (EDCBA)	6b Binary (abcdeI)
0	00000	100111 or 011000
1	00001	011101 or 100010
2	00010	101101 or 010010
3	00011	110001
4	00100	110101 or 001010
5	00101	101001
6	00110	011001
7	00111	111000 or 000111
8	01000	111001 or 000110
9	01001	100101
10	01010	010101
11	01011	110100
12	01100	001101
13	01101	101100
14	01110	011100
15	01111	010111 or 101000
16	10000	010111 or 100100
17	10001	100011
18	10010	010011
19	10011	110010
20	10100	001011
21	10101	101010
22	10110	011010
23	10111	111010 or 000101
24	11000	110011 or 001100
25	11001	100110
26	11010	010110
27	11011	101110 or 001001
28	11100	001110
29	11101	101110 or 010001
30	11110	011110 or 100001
31	11111	101011 or 010100a

## Aspetti teorici per la rilevazione degli errori

Un errore può essere **sintattico** o **semantico**; l'errore sintattico è correggibile a livello DLL, però può darsi che dal livello DLL passi una parola che sia sintatticamente corretta, ma semanticamente errata, l'unico che può accorgersi di questo errore è l'applicazione a livello "applicativo" in quanto avrà a disposizione tutte le frame e potrà elaborarle.

Un errore è un **violazione** del protocollo o una sequenza che non esiste, in questo caso l'errore è sicuro; può capitare, infatti, che presa singolarmente una sequenza può essere "probabilmente" corretta. Arrivata una sequenza, con **certezza** possiamo dire se è errata o **probabilmente** se è corretta. Io posso fare rilevazione degli errori solo se ci sono **sequenze non valide**, ovvero se c'è **ridondanza**, in caso contrario non posso fare nulla. Il modo più



semplice per verificare la presenza di errori è il **bit di parità**: aggiungo un bit in modo che la parità deve essere pari (decisione a discrezione dell'utente, potremmo decidere che sia dispari), se c'è un errore su un bit di dati riesco a capire subito che c'è un errore; funziona così bene che molte memorie lavorano così. Se c'è un errore di parità in una frame la butto interamente, non blocco niente (come invece avviene nella RAM) perché altrimenti non riuscirei a comunicare più.

Nessun errore	Errore correggibile del singolo bit
1 0 1 0 1 1	1 0 1 0 1 1
1 1 1 1 0 0	1 0 1 1 0 0
0 1 1 1 0 1	0 1 1 1 0 1
0 0 1 0 1 0	0 0 1 0 1 0

Posso anche provare a recuperare l'errore salvando la frame, come? prendo i miei dati e li dispongo in tabella e poi faccio la parità per righe e per colonne: trasmetto molta ridondanza in più, che mi permette di correggere i dati. Nell'esempio la seconda colonna e la seconda riga hanno un errore di parità (entrambe hanno un numero dispari di 1 e segnano 0 come bit di parità, mentre dovrebbe essere il contrario); facendo l'incrocio come in battaglia navale scopro che il bit errato è quello in posizione (2,2), per cui lo cambio ed ho corretto l'errore. Questo metodo è attuabile se e solo se la probabilità di avere un solo errore in un gruppo di bit è molto superiore di averne due, se avessimo due errori nello stesso gruppo non potremmo fare nulla. Questo metodo è bellissimo ma ha due piccoli problemi:

1. Su una grossa mole di dati la probabilità di avere un solo bit errato in un gruppo è molto bassa
2. Per poter effettuare il controllo di cui abbiamo parlato devo avere necessariamente **tutti** i dati disponibili, in quanto non è un controllo da poter effettuare on-the-fly.

Mi serve qualcosa che allora lavori sul **flusso** di dati, in modo da non bloccare la comunicazione o intasarla (immaginiamo 10Gb di dati mandati tutti di botto).

## CRC e calcolo del CRC (Cyclic Redundancy Check)

E' un algoritmo che si utilizza praticamente ovunque, è talmente potente che non si è mai trovata una alternativa. L'implementazione in hardware è molto semplice, è invece la teoria che ci sta dietro che è una bella mazzata. Si basa sulla divisione.

d bit	r bit
D: bit di dati da spedire	R: CRC bit
Schema dei bit	

$D \cdot 2^r \text{ XOR } R$

Formula matematica

Consideriamo  $d$  bit costituenti i dati  $D$  da trasmettere e supponiamo che origine e destinazione si siano accordati su una stringa di  $r+1$  bit, conosciuta come **generatore**, che indicheremo come  $G$ . E' necessario che il bit più significativo di  $G$  (quello più a sinistra) sia 1. Dato il blocco di dati  $D$  il mittente sceglierà  $r$  bit addizionali,  $R$ , e li unirà a  $D$  in modo da ottenere una stringa  $d+r$  bit che, interpretata come un numero binario, sia esattamente divisibile per  $G$ . Il processo di controllo CRC è semplice: se la divisione da resto diverso da 0, il ricevente allora sa che c'è un errore; altrimenti i dati sono accettati come corretti.

$$\begin{array}{l} 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1 \\ \times^7 + \times^4 + \times^2 + \times + 1 \\ \hline \end{array} \quad \begin{array}{l} 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \\ \times^6 + \times^4 + \times^3 + \times + 1 \\ \hline \end{array}$$

La somma è

$$\begin{array}{l} \times^7 + \times^6 + \times^3 + \times^2 \\ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \end{array}$$

Il prodotto è

$$\begin{array}{l} x^{13} + x^{11} + x^8 + x^7 + x^5 + x^4 + 1 \\ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \end{array}$$

Dietro questo meccanismo c'è un rigore matematico, in quanto definendo ogni singolo bit come coefficienti di un polinomio appartenente al campo  $Z^2$ ; il fatto che sia un campo è importante perché significa che al suo interno sono definite le operazioni di somma e prodotto, che godono delle proprietà commutativa, associativa, etc....

Tra questi polinomi l'operazione di somma si effettua:  $1+0=1$ ,  $0+1=1$ ,  $0+0=0$ ,  $1+1=0$ . In questo caso abbiamo scoperto che la somma equivale allo XOR, non abbiamo definito la somma come XOR, in quanto nessuno ci da la certezza che lo XOR dia vita ad un campo. Per cui abbiamo trovato qualcosa che poggia su solide basi matematiche, casualmente abbiamo scoperto che le operazioni su questo campo sono implementabili con circuiteria da quattro soldi, ovvero un registro a scorrimento ed un circuito XOR.

Oltre a poggiare su solide base matematiche ed essere implementabile con poche risorse, ha la possibilità di essere utilizzato su dati on-the-fly. Non scappa praticamente nulla.

Più in particolare: tutti i calcoli CRC sono eseguiti in aritmetica modulo 2 senza riporti nelle addizioni e prestiti nelle sottrazioni. Questo significa che addizione e sottrazione sono operazioni identiche e che entrambe equivalgono a un'operazione di XOR sui bit degli operandi. Per esempio:

$$1011 \text{ XOR } 0101 = 1110 \longrightarrow 1011 - 0101 = 1110$$

$$1001 \text{ XOR } 1101 = 0100 \longrightarrow 1001 - 1101 = 0100$$

Come nella normale aritmetica binaria, la moltiplicazione di una stringa per  $2^k$  corrisponde allo slittamento a sinistra della stringa di  $k$  posizioni. Quindi dati  $R$  e  $D$ , la quantità  $D * 2^r \text{ XOR } R$  fornisce come risultato la stringa  $d+r$  bit. Ricordiamo che vogliamo trovare  $R$  in modo che esista un valore di  $n$  tale che

$$D * 2^r \text{ XOR } R = nG$$

Vale a dire, vogliamo scegliere  $R$  in modo che  $G$  sia divisibile per  $D * 2^r \text{ XOR } R$  senza resto. Se eseguiamo l'operazione di XOR (cioè sommiamo modulo 2, senza resto) di  $R$  con entrambi i termini dell'espressione precedente otteniamo:

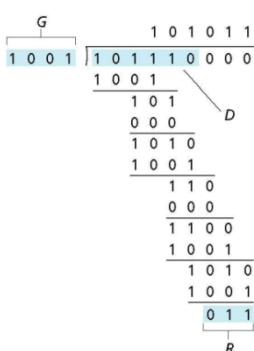
$$D * 2^r = nG \text{ XOR } R$$

Che ci mostra che se dividiamo per  $D * 2^r$  per  $G$ , il valore del resto è precisamente  $R$ . In altre parole possiamo calcolare  $R$  come:

$$R = \text{resto di } (D * 2^r / G)$$

Perché effettuare lo shift e non inserire direttamente il resto nella stringa? Perché sporco inevitabilmente i dati.

Es.



Immaginiamo di avere  $D = 101110$ ,  $d = 6$ ,  $G = 1001$ ,  $r = 3$

- 1)  $1011 \text{ XOR } 1001 = 10$
- 2)  $101 \text{ XOR } 000 = 101$
- 3)  $1010 \text{ XOR } 1001 = 11$
- 4)  $110 \text{ XOR } 000 = 110$
- 5)  $1100 \text{ XOR } 1001 = 101$
- 6)  $1010 \text{ XOR } 1001 = 011 = R$

C'è il resto! Nella trasmissione è successo qualcosa!

Che numero scegliamo come generatore? Bisogna trovare un numero come divisore e quelli più adatti sono i numeri primi, in quanto è divisibile per uno e per se stesso. L'equivalente nei polinomi è un polinomio irriducibile, ovvero che non si può scomporre, in questo caso abbiamo la certezza matematica che il bit più significativo sia pari a 1. Alcuni polinomi sono standard ovvero:

CRC-12  $x^{12} + x^{11} + x^3 + x^2 + x + 1$

CRC-16  $x^{16} + x^{15} + x^2 + 1$

CRC-CCITT

$$x^{16} + x^{12} + x^5 + 1$$

CRC-32

$$100000100110000010001110110110111$$

## Distanza di Hamming

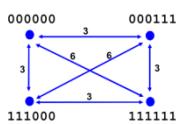


Definiamo la **distanza** come le differenze di bit che ci sono tra le due sequenze (es. 000000 e 000111 distanza = 3).

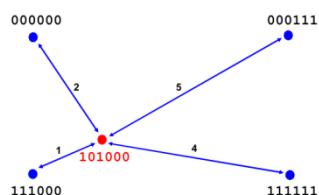
Prendiamo un vocabolario (o codice), ovvero un insieme di keyword, definiamo la **distanza nel vocabolario** come la

Consideriamo il seguente vocabolario:

000 000  
000 111  
111 000  
111 111



$P(e=1) >> P(e=2) >> P(e=3) \dots$



$P(e=1) >> P(e=2) >> P(e=3) \dots$

**minima possibile** tra le distanze di **tutte** le possibili coppie di keyword. Nel caso dell'esempio il minimo è 3. Immaginiamo che in ricezione arrivi la keyword 101000 che è errata, proviamo a correggerla. Supponiamo che qualcuno ci assicuri che la probabilità di avere un solo errore è molto più grande di quella di averne due, che a sua volta è molto più grande di quella di averne tre e così via. Se abbiamo questa certezza allora possiamo operare nel seguente modo: vediamo che la sequenza dista 1 dalla keyword 111000, 2 da 000000, 4 da 111111 e 5 da 000111, ovvero calcoliamo la distanza della sequenza tra tutte le possibili keyword del vocabolario. Sicuramente il trasmettitore ha trasmesso una di queste quattro, quella più probabile è quella a distanza minima, ma è solo una probabilità; abbiamo solo un'ottima probabilità che sia quella a distanza 1, ma certezza mai! La correzione si può fare quindi, ma solo se è presente la regola  $P(e=1) >> P(e=2) >> P(e=3) \dots$  ed esclusivamente su base probabilistica. Se avessimo avuto invece due keyword a distanza 1, nel caso dell'esempio, non avremmo mai potuto correggere l'errore, in quanto avremmo avuto una base probabilistica del 50% che non va bene. Quindi per correggere un errore mi serve un codice che abbia come minimo distanza 3, uno a distanza 2 non mi permette di correggere l'errore.

Generalmente: per poter **correggere** **e** errori è necessario un vocabolario con distanza  $d = 2e + 1$ , mentre per **rilevare** gli errori è necessario un vocabolario con distanza  $d = e + 1$ .

Consideriamo un codewords da 10 bit come in figura. Abbiamo che le parole valide sono 4, ma in realtà potremmo

Esempio: codewords da 10 bit

0000000000  
0000011111  
1111000000  
1111111111

Parole valide: 4.

Distanza di hamming: 5

Correzione di errori: 2 bit

Rilevazione di errori: 4 bit.

averne 1024 ( $2^{10}$ ), per cui ne sto sprecando "solo" 1020. Esiste un vocabolario con la stessa distanza, ma con più parole valide? Posso prendere in considerazione un vocabolario con 8 parole valide. Lo spreco è sempre grande, anche se questa volta riesco a riconoscere due errori. Si può fare meglio? Come faccio a costruirmi il mio codice? Se io ho  $m$  bit di dati (quindi  $2^m$  combinazioni possibili) e voglio fare correzione singola, quanta ridondanza devo metterci? Abbiamo due

Codewords da 10 bit

A	B	C	D	E	F	G	H
A 0000000000 :	A 0 5 5 6 5 6 6 7						
B 0000011111 :	B 5 0 6 5 6 5 7 6						
C 1111000000 :	C 5 6 0 5 6 5 7 6						
D 0011111100 :	D 6 5 5 0 7 6 6 5						
E 1100100101 :	E 5 6 6 7 0 5 5 6						
F 1100110101 :	F 6 5 7 6 5 0 6 5						
G 1111000010 :	G 6 7 5 6 5 6 0 5						
H 1111011001 :	H 7 6 6 5 6 5 5 0						

Parole valide: 8. Distanza di hamming: 5  
Bit di dati 3, bit di controllo 5.

problemi quindi: come si calcola  $r$ ? una volta calcolato  $r$  come si costruisce il codice? Scriviamo  $n = m + r$ , ma ancora non sappiamo quando vale  $r$ , abbiamo semplicemente fatto una posizione, però possiamo dire che le combinazioni possibili sono  $2^n$ , però quelle valide sono solo  $2^m$ , visto che  $m$  sono i bit che ci servono. Prendiamo la keyword

$n+1$  { 10001  
00001  
11001  
10101  
10011  
10000 }

Non valide a distanza 1

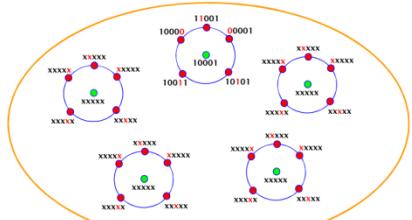
valida in figura, ovvero la prima (verde), ci sono in totale  $n$  bit; se cambiamo i singoli bit ad uno ad uno della keyword, otteniamo altre  $n$  parole tutte a distanza 1, in totale quindi abbiamo un insieme fatto da  $n + 1$  keyword. Graficamente abbiamo un bel cerchietto. L'insieme è formato da  $2^m$  elementi validi (cerchietti verdi), mentre proprio tutto l'insieme è formato da  $2^n$  elementi. Per cui otteniamo:

$$(n+1)2^m \leq 2^n$$

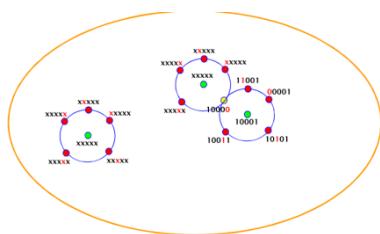
$$(m+r+1)2^m \leq 2^{m+r}$$

$$(m+r+1) \leq 2^r$$

$$m+1 \leq 2^r - r$$



Che è vera solo ed esclusivamente per  $d = 3$ . Se tutti i cerchietti sono disgiunti, allora lo posso scrivere che  $(n+1)2^m \leq 2^n$ , altrimenti se non lo sono abbiamo qualche grosso problema, in quanto i punti dalla parte di  $(n+1)2^m$  li conto due volte, per cui non potranno mai e poi mai essere minori del totale. L'unico modo per averne la certezza è dimostrandolo: supponiamo per assurdo che avvenga qualcosa come nelle figure, per cui 10000 è a



distanza 1 da due keyword differenti, per cui il mio vocabolario non è a distanza 3, ma a distanza 2 (in quanto cambiando 2 bit mi permette di passare da una keyword all'altra), da cui abbiamo l'assurdo. L'assurdo è nato dall'avere supposto che gli insiemi possano non essere disgiunti. Allora ho anche dimostrato che  $(n+1)2^m \leq 2^n$  è vera. La quantità  $d=3$  c'è, ed è nascosta dentro questa dimostrazione! Per cui gli insiemi possono essere disgiunti solo se  $d = 3$  (ma anche  $d \geq 4$ ).

Esempi:

$$m=8 \Rightarrow r=4 \quad (8+4+1) \leq 2^4 \quad 13 < 16$$

$$m=11 \Rightarrow r=4 \quad (11+4+1) \leq 2^4 \quad 16 = 16$$

Per  $d = 5$ , l'espressione vista in precedenza, è FALSA. L'espressione in questo caso sarebbe:  $[1 + n + n(n - 1)]2^m \leq 2^n$

I codici devono essere necessariamente a distanza dispari, in quanto a distanza pari potremmo avere casi in cui il dubbio ancora rimane.

Rimane solo il problema di come costruire il codice ora che abbiamo  $r$ , come si fa?

m	r	n	m	r	n
1	2	3	14	5	19
2	3	5	15	5	20
3	3	6	16	5	21
4	3	7	17	5	22
5	4	9	18	5	23
6	4	10	19	5	24
7	4	11	20	5	25
8	4	12	21	5	26
9	4	13	22	5	27
10	4	14	23	5	28
11	4	15	24	5	29
12	5	17	25	5	30
13	5	18	26	5	31
			27	6	33

Nella colonna  $n$  mancano alcuni numeri, ovvero le potenze del 2. Se io volessi  $n = 16$ , posso farlo? Potrei fare  $11 + 5$  o  $12 + 4$ , solo che se uso 12 bit come  $m$ , la ridondanza deve essere 5 e la loro somma fa 17, per cui questa combinazione non posso usarla. Stessa cosa accade per la combinazione  $11 + 5$ , se ho 11 bit posso usarne 5, però qualcuno può dirmi, perché usarne 5 se ne bastano semplicemente 4? Un bit rimarrebbe inutilizzato, per cui si preferisce la combinazione  $11 + 4$ .

## Codici di Hamming

Scriviamo i bit di dati in altro modo, come in figura, intervallando i bit di dati con i bit di controllo. I bit di dati li

Bit dei dati
numeriamo da 1 a  $n$  e i bit di controllo li mettiamo nelle potenze del 2 e gli altri nei posti che non
  
↓ ↓ ↓ ↓ ↓
sono potenza del 2. Il bit 6 lo rappresentiamo in binario come  $2 + 4$  ( $2^1 + 2^2$ ), per cui stiamo
  
00110010000
rappresentando i dati, in un certo senso, mediante i bit di controllo.

Bit di controllo
Facciamo un esempio: prendiamo la stringa di dati 10010001100. Per quanto detto in precedenza
  
001100100001100
i bit di controllo dovrebbero essere nelle posizioni xx1x001x0001100 (ovvero 1, 2, 4 e 8);
costruiamo la seguente tabella, dove è indicato quali bit di controllo costruiscono i bit di dati
normali, da questa tabella ricaviamo che:

$6=2+4$	$x = 1 + 2 + 4 + 8$
	$3 = 1 + 2$
	$5 = 1 + 4$
	$6 = 2 + 4$
	$7 = 1 + 2 + 4$
	$9 = 1 + 8$
	$10 = 2 + 8$
	$11 = 1 + 2 + 8$
	$12 = 4 + 8$
	$13 = 1 + 4 + 8$
	$14 = 2 + 4 + 8$
	$15 = 1 + 2 + 4 + 8$

$$1 = 3 \text{ XOR } 5 \text{ XOR } 7 \text{ XOR } 9 \text{ XOR } 11 \text{ XOR } 13 \text{ XOR } 15$$

$$2 = 3 \text{ XOR } 6 \text{ XOR } 7 \text{ XOR } 10 \text{ XOR } 11 \text{ XOR } 14 \text{ XOR } 15$$

$$4 = 5 \text{ XOR } 6 \text{ XOR } 7 \text{ XOR } 12 \text{ XOR } 13 \text{ XOR } 14 \text{ XOR } 15$$

$$8 = 9 \text{ XOR } 10 \text{ XOR } 11 \text{ XOR } 12 \text{ XOR } 13 \text{ XOR } 14 \text{ XOR } 15$$

Ovvero:

**1 = 1 XOR 0 XOR 1 XOR 0 XOR 0 XOR 1 XOR 0 = 1**

**2 = 1 XOR 0 XOR 1 XOR 0 XOR 0 XOR 0 XOR 0 = 0**

**4 = 0 XOR 0 XOR 1 XOR 1 XOR 1 XOR 0 XOR 0 = 1**

**8 = 0 XOR 0 XOR 0 XOR 1 XOR 1 XOR 0 XOR 0 = 0**

Da cui la stringa finale di dati:

**101100100001100**

**Abbiamo ottenuto il nostro scopo, siamo riusciti a costruire il nostro codice partendo da  $r$ .**

Facciamo un esempio di funzionamento. Supponiamo che la stringa vista in precedenza arrivi nella seguente forma, ovvero siamo certi che ci sia un errore

**101100100101100**

Costruendo lo stesso schema visto in precedenza sappiamo che:

**1 = 1 XOR 0 XOR 1 XOR 0 XOR 0 XOR 1 XOR 0 = 1**

**2 = 1 XOR 0 XOR 1 XOR 1 XOR 0 XOR 0 XOR 0 = 1**

**4 = 0 XOR 0 XOR 1 XOR 1 XOR 1 XOR 0 XOR 0 = 1**

**8 = 0 XOR 1 XOR 0 XOR 1 XOR 1 XOR 0 XOR 0 = 1**

$x = 1 + 2 + 4 + 8$
$3 = 1 + 2$
$5 = 1 + \quad 4$
$6 = \quad 2 + 4$
$7 = 1 + 2 + 4$
$9 = 1 \quad + 8$
<b>10 = 2 + 8</b>
$11 = 1 + 2 \quad + 8$
$12 = \quad 4 + 8$
$13 = 1 \quad + 4 + 8$
$14 = \quad 2 + 4 + 8$
$15 = 1 + 2 + 4 + 8$

Notiamo che c'è un errore alla posizione 2 e 8, consultando la tabella notiamo che la posizione dei bit di dati che coinvolge contemporaneamente la posizione 2 e 8 dei bit di controllo è la 10. In questo modo abbiamo individuato dove si trova l'errore. Se invece dallo schema precedente avessimo ottenuto come risultato 1011, ovvero la posizione 8 dei bit di controllo, allora l'errore era proprio nei bit di controllo; è errato presumere che i bit di controllo non possano contenere errori.

Operativamente non dobbiamo fare altro che prendere un contatore  $c$  porlo uguale a 0 e fare la verifica: l'1 è corretto  $\rightarrow c=0$ , il 2 è sbagliato  $\rightarrow c = 0 + 2 = 2$ , il 4 è corretto  $\rightarrow c = 2$ , 8 è sbagliato  $\rightarrow c=2+8=10$ . Alla fine il contatore ci indica esattamente il bit errato!

Con due bit errati non funziona più nulla, per cui l'ipotesi è sempre quella, che sia presente solo ed esclusivamente un bit errato nella keyword. In questo livello non si può scoprire.

A volte può capitare che gli errori siano concentrati in una zona ristretta e poi vi siano sequenze enormi prive di errori, in questi casi si può procedere nella seguente maniera:

In output ho la seguente stringa da trasmettere, intervallata dai bit di controllo

```
xx1x110 xx1x110xx0x011xx0x110xx1x000xx0x011xx1x100xx0x001xx0x11
xx0x011 0xx1x011xx1x110xx1x001xx0x011
```

Prendo la sequenza precedente e la scrivo in colonna in una matrice, una volta ottenuta questa tabella la inserisco in uno stream e la spedisco.

```
xx0x110
xx1x011
xx1x110
xx1x001
xx0x011
xxxxxxxxxxxxxx100101001110xxxxxxxxxx101001
0101001110100111010100101010111
```

Ad un certo punto in ricezione avviene un errore:

```
xxxxxxxxxxxxxxxxxxxxxx100101001110xxxxxxxxxxxxx101001
01010000010111101010010101011
```

Utilizzando lo stesso metodo usato in output scompongo la stringa in una tabellina

xx1x100	Ottenendo così gli errori suddivisi equamente in diverse sotto stringhe, potendo così riuscire a correggerli
xx0x01	con i metodi spiegati in precedenza.
xx0x100	
xx1x010	
xx0x001	Ovviamente queste operazioni sono effettuabili solo se è presente un buffer di memoria temporaneo e se
xx1x110	si è disposti a perdere un po' di tempo.
xx0x011	
xx0x110	
xx1x001	
xx0x011	

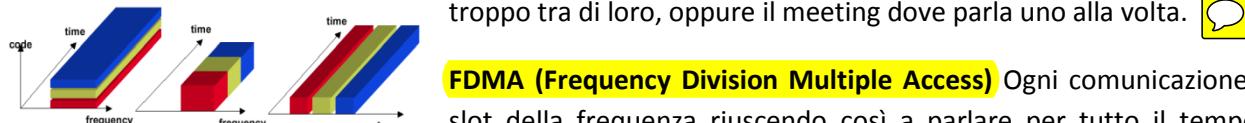
Ricapitolando, per correggere e rilevare è necessaria ridondanza, solo che per correggere è necessaria molta più ridondanza e si perde del tempo; per cui bisogna capire quando è opportuno correggere e quando rilevare soltanto. Avere qualcosa che corregge gli errori è il massimo, ma serve sempre? Nella fibra ottica ad esempio c'è la possibilità di 1/10<sup>13</sup> bit errati, in questo caso la probabilità è così bassa che non ha neanche senso parlare di controllo d'errore. Il controllo però posso sempre approntarlo, in quanto il CRC è un ottimo metodo che non ci fa sprecare nulla. Se al posto della fibra abbiamo il wifi dove il rapporto è 1/1000, se usiamo il metodo di prendo e butto, allora non comunichiamo più; in questo caso ha senso inserire un meccanismo di correzione. Quindi ci sono situazioni a livello DLL in cui il tasso di errore è così basso che è esagerato anche la rilevazione, mentre altri casi in cui il tasso di errore è così elevato che è essenziale avere un sistema di correzione.

## Il sottolivello MAC



Se abbiamo un canale punto-punto non ci sono problemi, A parla quando vuole e B ascolta. Se il canale è half-duplex deve mettersi d'accordo, però con uno solo. Il problema nasce quando il canale è broadcast: in questo caso serve una politica di accesso al mezzo, servono delle regole per evitare conflitti. Nel wifi possono accadere delle sovrapposizioni, ma ancora qualcosa è recuperabile. Come facciamo a gestire il tutto?

Immaginiamo una serie di coppie dove ognuna parla una propria lingua, parlano tra di loro e non si disturbano troppo tra di loro, oppure il meeting dove parla uno alla volta.



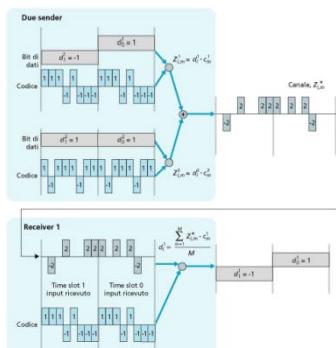
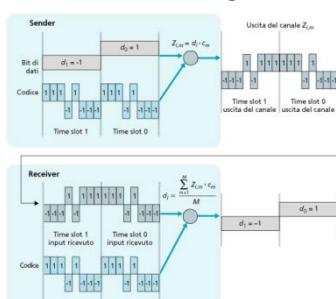
**FDMA (Frequency Division Multiple Access)** Ogni comunicazione occupa uno slot della frequenza riuscendo così a parlare per tutto il tempo che vuole.

L'unico problema è che ogni comunicazione possiede una parte limitata di banda, per cui sta limitando i suoi range di comunicazione.

**TDMA (Time Division Multiple Access)** in questo caso ogni comunicazione ha la banda interamente disponibile, ma è limitata nel tempo, dovendolo dividere con altre comunicazioni in corso; per cui la comunicazione avviene a turni.

**CDMA (Code Division Multiple Access)** è invece uno schema inventato da Hedy Lamarr (attrice austriaca del periodo fascista) dove tutti contemporaneamente usano il tutto il tempo e le frequenze. Serve quindi un nuovo schema per suddividere la frequenza di banda, chiamato **codice**. Il telefono GSM funziona che ogni bts (antenna) prendono le frequenze e le dividono in canali, ogni canale è suddiviso in sotto canali. Ci sono tanti sotto canali, ma non sono sufficienti per ogni singolo cellulare, conviene quindi, ad esempio, assegnare 10 sotto canali a 90 terminali, facendoli girare in qualche modo. Come faccio io a prendere un sotto canale? Lo prenoto inviando un segnale su un apposito canale di servizio, senza disturbare i canali di comunicazione. [Il canale di servizio è sprecato, allora ci faccio viaggiare gli sms] La telefonata è digitalizzata, per cui prendo la durata della telefonata e la stringo aumentando il bit rate, occupo così meno tempo effettuando così una suddivisione di tempo. GSM usa un sistema di multiplexaggio su frequenza e su tempo, però è necessario la prenotazione sul canale di servizio. Con il sistema CDMA si tende ad

avere un approccio al contrario; questo sistema si chiama UMTS. Ho una serie di trasmettitori ed ad ogni trasmettitore assegno un codice formato da diversi chips, inoltre utilizzo una tecnica particolare dove i bit 0 e 1 li codifico con un valore positivo e uno negativo e permetto la sovrapposizione del segnale: se due utenti trasmettono contemporaneamente 1 e -1 il risultato è 0, se invece trasmettono 1 e 1 il risultato è 2. Il trasmettitore ha un codice che deve essere lo stesso del trasmettitore, il codice deve essere molto piccolo, il bit di dati deve essere grande quanto tutto il codice. Se deve trasmettere -1 inverte il codice, se deve trasmettere 1 lo lascia invariato. A destinazione invece si fa sort a bit a bit è si ottiene il bit di dati originario.



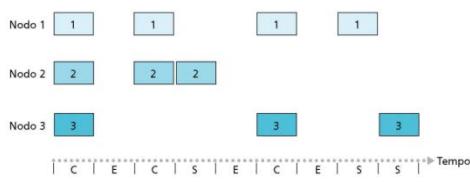
Cosa accade invece con più comunicazioni in corso! Si sommano i codici risultanti delle comunicazioni dei due sender, in trasmissione; in ricezione invece il receiver, mediante il suo codice riesce a decriptare il segnale e ricavare il segnale che gli interessa. Il funzionamento è esplicabile in figura. Il trasmettitore 1 applica la sua maschera ai bit di dati da trasmettere e la stessa cosa fa il trasmettitore 2. Una volta ottenuti i dati dei due trasmettitori si effettua una somma algebrica tra i due segnali e si ottiene il segnale risultante. Questo segnale viene poi preso dai singoli trasmettitori a cui applicano la propria maschera ed effettuando poi una media aritmetica, da cui si otterrà il bit dati originario.

In questo caso non è necessario chiedere il canale, in quanto è sempre disponibile. Ovviamente se la velocità di comunicazione massima è di 8MB/s e abbiamo 8 comunicazioni, ogni comunicazione avrà a disposizione 1MB/s; banalmente, la velocità del canale non è infinita per cui bisogna ripartirla tra le varie comunicazioni. Questi codici sono fatti in maniera tale, che se ne prendiamo 3, il terzo non deve essere una combinazione lineare degli altri due, ovvero i codici devono essere fatti in modo tale che formano una base **ortogonale**. UMTS non è solo questo, ovvero questa è solo la base (ad esempio CDMA è a velocità costante, mentre UMTS no, ovvero a seconda di quanto mi trovo lontano dall'antenna posso andare più velocemente o no).

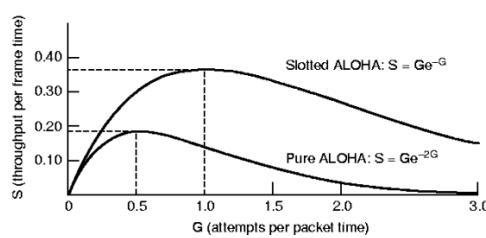
## Aloha puro e slotted



L'università delle Hawaii aveva necessità di mettere in comunicazione i diversi edifici che si trovavano nelle varie isole. Posare un cavo tra le isole era un'impresa assurda, per cui si decise ad utilizzare un'antenna centrale e tante antenne periferiche (una per ogni building) per comunicare con la centrale. Il protocollo era molto semplice, se il canale è libero allora la comunicazione con la centrale andava a buon fine, se era occupato ed avveniva una collisione, si buttavano i dati e si riprovava. Sistema ottimo per trasmissioni sporadiche, pessimo in caso di rete pesantemente utilizzata; non vi era nessuna garanzia, ovvero non vi era alcun protocollo di accesso al canale. Lo spettro era suddiviso in upload (verso antenna centrale) e download (dall'antenna centrale); la centrale sapeva più o meno cosa succedeva al pacchetto, in quanto l'antenna riceveva il pacchetto e lo ritrasmetteva in broadcast a tutte



quante. Per cui la trasmettente invia il pacchetto e aspetta, se sente il suo pacchetto trasmesso dalla centrale pulito allora vuol dire che tutto era andato a buon fine. Sul canale di download invece non esiste congestione in quanto usato solo dalla centrale. Tutta questa era l'idea del protocollo **ALOHA**.



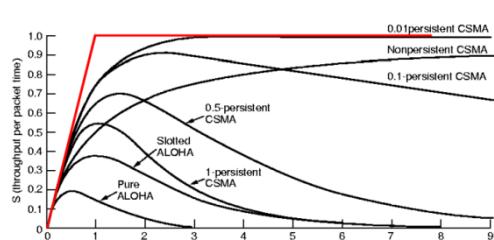
La situazione però si può migliorare; l'antenna centrale manda dei segnali di temporizzazione, dove ad ogni segnale chi vuole può trasmettere. Questo sistema si chiama **slotted Aloha**, che in effetti migliora il sistema. A basso traffico, quanto il throughput **G** è piccolo, la frame è probabile che passa indenne; man mano che il traffico aumenta c'è la possibilità di avere collisioni, fino ad arrivare al punto in cui nessuno riesce più a comunicare.

L'andamento di ALOHA puro e slotted è rappresentato in figura. Il canale viene utilizzato al più per il 18,4% delle sue potenzialità, nel caso del puro e 36,8% nel caso dello slotted. Il problema di base è che ognuno parla quando vuole, la cosa banalissima da fare per evitare problemi è ascoltare il canale per verificare che nessuno stia parlando.

## CSMA (Carrier Sense Multiple Access)

E' l'evoluzione di ALOHA, prima di trasmettere questo protocollo ascolta la portante per verificare che nessuno stia comunicando. Dopo possono esserci diversi comportamenti:

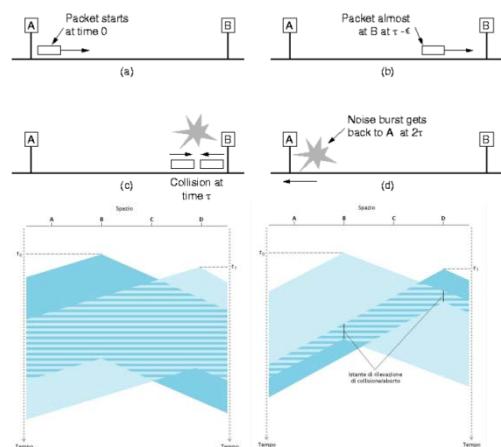
- **CSMA 1-persistent** appena la precedente comunicazione termina trasmette immediatamente, il problema è che se ci sono più sistemi che aspettano, comunicheranno nello stesso istante causando collisione.
- **CSMA p-persistent** ovvero trasmetti con una certa probabilità, altrimenti aspetti ancora. La probabilità è random.
- **CSMA non-persistent** la stazione aspetta un tempo random prima di ricontrolare il canale.



Il comportamento ottimo è la curva rossa, in quanto occupo tutto il canale usandolo al massimo e senza perdere niente. Ovviamente abbassando la probabilità di trasmettere aumenta le prestazioni (0.1-persistent è migliore di 0.5-persistent). Lo 0.01-persistent è il migliore di tutti, ma abbiamo solo l'1% di probabilità di trasmettere.

## CSMA/CD

L'idea è la seguente: due macchine A e B sono sullo stesso canale. A controlla la portante e se nessuno parla inizia a

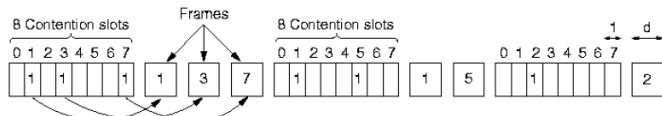


spedire, il pacchetto arriva vicino a B, ma B ancora non sa che gli sta arrivando qualcosa per cui ascoltando la portante non se ne accorge, per cui inizia a trasmettere. Avviene la collisione, l'idea che lo differenzia dal CSMA semplice è che quando avviene la collisione, chi trasmette libera il canale il prima possibile. Durante le comunicazioni può capitare una collisione, si smette di trasmettere e c'è un attimo di silenzio finché qualcuno non riprova e così via. Il periodo critico per una frame è il tempo di propagazione massimo, se supero questo tempo la frame passa fissa, in quanto tutti hanno capito che c'è trasmissione, per cui seguendo questa filosofia conviene inviare frame grandi. Il problema è che se si ha una frame grande, il canale che trasmette non lo lascia più, per cui abbiamo un blocco della risorsa. Per cui bisogna avere una dimensione

che non sia eccessiva, in modo da lasciare spazio anche gli altri. Questa è CSMA/CD, per gli amici **Ethernet**. Nel grafico in figura, vediamo che le zone tratteggiate sono dove è avvenuta una collisione, CSMA/CD fa in modo tale che questa zona, ovvero dove il canale è sprecato, sia il più piccolo possibile. **CD** spara un segnale, **jamming**, in modo da segnalare che è avvenuta sicuramente una collisione e interrompe la comunicazione il prima possibile; una volta che due stazioni sono andate in collisione devono attendere un po' di tempo prima di riprovare. Utilizzano un sistema chiamato **backoff esponenziale**, dove: il primo tentativo è andato male, al secondo tentativo i due contendenti sorteggiano un numero da 0 a 3; se il primo ottiene 1 e il secondo 3, allora non ci sono problemi, in quanto il primo aspetta 0 e 1 intervalli di tempo e trasmette, il secondo aspetta 0-1-2 e 3 intervalli di tempo e trasmette, solo che sente la portante occupata dal primo per cui rimane ancora in attesa finché il primo termina. Se sfortunatamente i due contendenti sorteggiano lo stesso numero, allora avviene una nuova collisione, per cui bisogna fare un terzo tentativo sorteggiando stavolta un numero da 0 a  $2^3 - 1$ . Il massimo numero di tentativi è 10, che è un evento veramente raro; arrivati a 10 tentativi si smette di provare e si segnala che c'è un problema sul canale, per cui non si può comunicare. Se durante questo tentativo c'è un terzo che si intromette, ad esempio al nono tentativo, questo terzo vince subito perché lui estrae un numero da 0 a 1, per cui frega i due contendenti. Il collision detect è una caratteristica peculiare dei canali cablati, infatti non può essere applicato ad un mezzo wifi in quanto quest'ultimo

può solo trasmettere o ascoltare, per cui non può trasmettere ed ascoltare se ci sono collisioni contemporaneamente. La velocità di propagazione del pacchetto è dell'ordine del microsecondo( $\mu s$ ), ma non è da trascurare, in quanto poiché non è istantanea la comunicazione, può capitare che un altro ancora non si rende conto che qualcuno sta trasmettendo e trasmettere di sopra, causando una collisione. Se avessimo un canale lungo 1000 metri ad una velocità di propagazione di 200MB/s ci vogliono circa 5 $\mu s$ , che sembrerebbe essere un tempo trascurabile, ma non lo è. Più è lungo il canale, più peggiora la latenza, però man mano che si allunga il cavo si tende ad utilizzare una serie di protocolli che sfruttano meglio il canale, ad esempio una tecnica è quella di usare frame molto grandi che occupano tutto il canale, oppure di diminuire al massimo lo spazio fra una frame e la successiva.

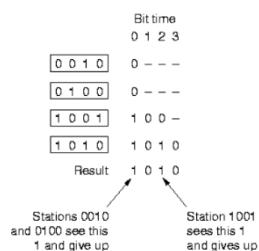
## Protocolli senza collisioni



Abbiamo una zona chiamata **zona di contesa o pre-allocazione**, e sono di numero pari al numero di macchine presenti nel mio sistema. Chi vuole trasmettere, ogni macchina è numerata ovvero ha assegnato uno di questi

slot, appena ha necessità e vede passare il suo slot ci mette 1, e siccome quello slot è suo solo lei può mettere 1. Tutti leggono tutti gli slot, per cui sanno chi deve parlare e in che ordine. Ad alto traffico l'efficienza di questo sistema è massimo, a basso traffico abbiamo il canale occupato da slot di contesa. Il problema è che il sistema non è equo, in quanto gli ultimi hanno più tempo di decidere se parlare o no. Altro problema è che devo sapere quante macchine ci sono e se sono troppe le macchine (es. 10000) diventano troppi i bit di contesa. Questo sistema viene detto a **mappa di bit**.

Altro sistema invece è quello a **conteggio binario a ritroso**. Vediamo l'esempio in figura. Ogni macchina ha un numero, ma lo vede in binario; prima le macchine mettono il bit più significativo, se c'è il bit 1 parlano se c'è 0



stanno zitte. In questa maniera dopo il primo turno tutte le macchine che hanno 1, parleranno e non c'è collisione in quanto dicono la stessa cosa, quelle che hanno bit 0 stanno zitte. Nel secondo passaggio le macchine che avevano bit 1 mettono ora 0 per cui il "gioco continua", nel terzo passaggio solo una mette 1 mentre l'altra mette 0 "perdendo il gioco". Rimane solo la macchina che aveva messo 1 e guarda caso sul canale rimarrà proprio il suo indirizzo, per cui le basterà semplicemente aggiungere la frame e trasmettere. In questo caso sono favorite quelle che hanno indirizzo alto. Per cui non è equo neanche questo sistema.

## Protocolli a turno

Un protocollo a turno molto famoso era il **Token Bus** usato molto in ambito industriale. Ho sempre un mezzo

broadcast, però ho anche un Token, ovvero un gettone virtuale. Questo messaggio è un diritto a parlare che passa di macchina in macchina, che viene passato tra le varie macchine. E' un protocollo così complesso che il manuale del protocollo è enorme, perché? Cosa accade se il token si perde? Mettiamo una macchina che sorveglia, che in caso genera il token. E se per caso ce ne sono due?

Allora la macchina che si accorge di questo toglie quello in più e se un'altra macchina fa la stessa cosa si perde un'altra volta il token. E se una macchina esterna al ciclo vuole entrare come fa se non può avere il token? Meglio non parlarne perché sarebbe una cosa allucinante. L'unico vantaggio del sistema è che è possibile gestire le priorità.

Altro sistema era **Token ring** dove le macchine erano messe però ad anello.



## Ethernet (802.3)

E' un implementazione di CSMA/CD, la filosofia è quella. Ethernet era il nome del protocollo che Robert Metcalfe aveva pensato quando aveva scritto la sua tesi di dottorato, in seguito fu istituito il comitato IEEE 802.3 e inglobato tutto il lavoro di Metcalfe, volgarmente lo chiamiamo Ethernet, ma il nome ufficiale è 802.3. 802 è il nome di una serie di comitati di standardizzazione. 802.1 si occupa di questioni di alto livello, per quanto riguarda le LAN; poi ci sono una serie di comitati che definiscono una serie di protocolli di accesso al mezzo di comunicazione. In principio

Token Bus e Token Ring erano molto utilizzati perché davano delle garanzie che CSMA/CD non davano, ad esempio quest'ultimo non garantisce che una macchina riesca a comunicare. Ad esempio una cosa che può succedere in

- 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']
- 802.10 Standard for Interoperable LAN Security (SILS) [in 'hibernation']
- 801.11 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

Ethernet è che una macchina trasmette a tempesta impedendo così agli altri di comunicare, infatti collocare un sensore importante o comunque una macchina che abbia un'altissima priorità in caso di emergenza potrebbe essere una follia.

La prima versione di Ethernet era a 10MB/s ed erano previsti 4 tipi di cablaggi (10 sta per la velocità di comunicazione; Base sta per Based ovvero comunicazioni in banda base, cioè il segnale veniva inserito così com'era nel cavo, i restanti valori identificano i vari tipi di cavo):

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

- **10Base5** era un cavo rigidissimo che usava, per connettersi ai vari dispositivi, una presa a "vampiro" che mordeva il cavo e tramite un punteruolo si connetteva allo strato di cavo per la connessione. C'erano dei punti ben precisi dove inserire il connettore, in quanto in quei punti la guaina metallica che avvolgeva la parte interna era più larga. Bisognava evitare di tagliarlo in quanto, le unioni causano delle resistenze che possono disturbare il segnale.
- **10Base2** invece era più sottile del suo predecessore, permettendo così di poter tagliare il cavo e di innestare le varie macchine con un particolare connettore a "T".

Questi tipi di cavo sono stati abbandonati in quanto, alla fine del cavo, era necessario inserire una sorta di tappo. Se una frame si perdeva e colpiva questo tappo, rimbalzava andando a disturbare la comunicazione; la soluzione al problema era inserire una sorta di resistenza che "mangiava" il segnale perso. Altro problema era dettato dai guasti, essendo collegate in serie se si guastava una terminazione, essa faceva da tappo disturbando tutto il canale e individuare la terminazione guasta era un grosso problema. Nella tabella sopra, sono presenti anche i valori massimi di lunghezza di un segmento di cavo e della quantità di nodi massima che si possono inserire, questi valori sono quelli entro cui il segnale dovrebbe viaggiare senza problemi, senza causare dispersioni; possiamo creare dei segmenti più grandi o mettere più nodi, però non è garantito che il segnale viaggi intatto.

- **10Base-T** E' il tipo di cavo Ethernet odierno. Utilizza un doppino di tipologia 3 (si poteva utilizzare anche il doppino di categoria 5, ma era esagerato). Una coppia è usata per trasmettere, un'altra coppia per trasmettere e l'ultima è inutilizzata. I dispositivi non sono più collegati in serie, ma a stella; questo elimina completamente la necessità del tappo e semplifica le problematiche legate ai guasti, in quanto guastandosi una terminazione le altre continuano a funzionare. Il centro stella viene chiamato Hub, che è un sistema non intelligente.
- **10Base-F** E' la variante del 10Base-T, ma utilizza la fibra ottica.

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

I vari cavi si suddividono in CAT, che si differenziano tra di loro per il tipo di intreccio e la qualità. Ogni cavo fa passare un dataRate diverso tra loro, ad esempio il cavo Cat5 con una piccola miglioria può arrivare anche ad 1Gbps.

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	--

Tutto ciò deriva sempre dalla teoria di Shannon. Come si può vedere dalla tabella successiva, i cavi sono di diversi colori e suddivisi a due a due, il cavo di colore pieno si accoppia con quello dello stesso colore con una striscetta di bianco (White/Orange + Orange).

Nelle coppie inutilizzate si potrebbe far passare del segnale, mediante dispositivi fuori standard, anche se danno un po' di disturbo in quanto passa comunque del segnale.

## Frame Ethernet



- 8 byte **Preambolo**
- 6 byte **MAC Destinatario**
- 6 byte **MAC Mittente**
- 2 byte **E-Type/size** originariamente indicava la dimensione massima della frame, reso poi inutile in quanto il calcolo si può effettuare con l'algoritmo di framing. Questo campo va a indicare quanti dati stai utilizzando, perché se trasmetti 1 Byte ne hai 45 di carico inutile e questo campo segnala che solo quel byte è di carico utile.
- X byte **Payload** ovvero il campo dati, che può andare da 0 a 1500 byte, però ha un numero minimo che è 46 byte, perché deve per forza esserci una dimensione minima della frame; non possiamo avere una frame più piccola di 64 byte.
- 4 byte **CRC**

Perché la frame deve essere di 64 byte minimo? Nello standard il ritardo massimo è di 51,2 µs, alla velocità di 10MB/s quanti bit passano?

$$1s : 10MB/s = 51,2 * 10^{-6}s : x$$

$$x = 51,2 * 10^{-6} * 10^7 = 512 \text{ bit} = 64 \text{ byte}$$

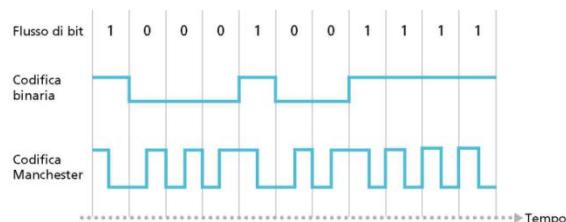
Per cui la lunghezza della frame deve essere abbastanza affinché, il doppio della sua lunghezza sia più grande della quantità di byte che si possono trasmettere in 51,2 µs. Se la frame è più lunga del canale e avviene una collisione, allora il trasmittitore della frame si accorge che la collisione si è "fagocitata" la sua frame inviata. Se questo tempo fosse più piccolo e avviene la collisione allora non se ne accorge, ad un certo punto la rileverà ma non capirà che è stata colpa sua; se la macchina si accorge che la sua frame è finita male allora sa che deve ritrasmettere il suo messaggio. Il preambolo non viene calcolato nella dimensione della frame.

0x0800 IPv4
0x0806 ARP
0x0842 Wake-on-Lan
0x0835 RARP
0x089B EtherTalk
0x08F3 AppleTalk Address Resolution Protocol (AARP)
0x8100 VLAN-tagged frame (IEEE 802.1Q)
0x8137 Novell IPX
0x8138 Novell
0x86DD IPv6

Sul campo E-Type (Ethernet Type) possiamo dire che se i valori payload sono fino a 1500 allora i valori sopra questo numero sono tutti liberi, non possono esistere. Con l'aiuto di due byte possono fare numeri superiori a 1500, per cui nell'E-Type viene inserito un codice, come quello in figura, che rappresenta che tipo di payload è presente. È una trovata necessaria, ma è una chiara violazione del sistema a livelli, in quanto inseriamo qualcosa del livello 3 (ovvero diciamo già cosa dobbiamo aspettarci al

livello 3) nel livello 2.

## Codifica Manchester



Il bit 1 e lo 0 vengono rappresentati da una combinazione di alto/basso o viceversa; hanno risolto il problema del framing quindi non ci sono dubbi se avviene trasmissione o no. Il secondo motivo per cui hanno scelto questo metodo, stiamo parlando della preistoria delle comunicazioni, è che la cosa più semplice per rilevare un segnale è vedere che c'è una transizione (banalmente rileviamo precisamente i bit). Ai nostri giorni si può fare molto di meglio, in quanto la velocità di segnalazione qui è a 20Mbps, e noi trasmettendo a 10Mbps ne stiamo sprecando giusto il 50%. L'ultimo problema da risolvere è riconoscere la fine di un bit dal suo punto intermedio; l'unico modo per risolvere è **sincronizzando** mittente e

ricevente.

destinatario allineandoli perfettamente mediante il **preambolo**. Invio una sequenza nota per un tot di tempo in modo da sincronizzarmi con l'altra macchina in modo da distinguere la metà di un bit o il passaggio ad un bit successivo. Il segnale da inviare è:

**10101010-10101010-10101010-10101010-10101010-10101010-10101010-10101011**

Ovvero una sequenza di onde quadra a 10Mhz (7 byte) seguite da una sequenza finale (8° byte) a 20Mhz. Il preambolo quindi **non** viene contato nella dimensione del pacchetto proprio perché è un pacchetto di sincronizzazione.

## Fast Ethernet (802.3u)

Se saliamo a 100Mbps la frame minima sale a 640byte, ma non viene fatto (quindi si rimane a 64byte) per avere una sorta di retro compatibilità. Per cui è stata solo aggiornata la velocità. Utilizza quattro tipi di cablaggi come sotto:

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

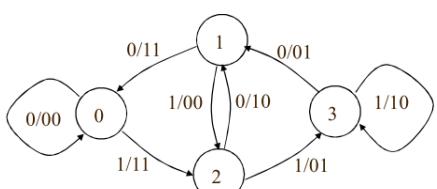
**FX** è la fibra ottica. **TX** è il doppino intrecciato di CAT5 (due coppie in andata e due al ritorno, le ultime due inutilizzate); la **T4** viene usata per retrocompatibilità con alcuni cablaggi esistenti, il problema è che bisogna utilizzare una codifica particolare chiamata **8B6T** (identifica 8 bit con una sestina di 3 valori), utilizzo il cavo in maniera particolare: una coppia solo in andata, una coppia solo in ritorno e due coppie o in andata o in ritorno, per cui posso avere massimo 3 coppie che vanno a 33Mbps in una sola direzione. Quindi 33Mbps in full-duplex e 66Mbps in half-duplex, a seconda di quello che serve. La codifica Manchester non serve più in quanto non devo fare più sincronizzazione e posso utilizzare codifiche tipo la 4B5B, avendo uno spreco molto più basso e quindi sfruttando meglio il mio cavo. Ovviamente devo cambiare l'hardware in quanto devo generare frequenze maggiori, inoltre le distanze, tranne la fibra ottica, sono scese, però i valori non sono proprio pochi.

## Gigabit Ethernet (802.3z)

- Rimossa la codifica 4B5B 100Mbps --> 125Mbps
- Utilizziamo tutte le coppie del cavo 125Mbps --> 500Mbps
- Trasmissione Full-duplex 500Mbps full-duplex
- Utilizzare 5 livelli per bound invece di 3 500Mbps --> 1Gbps

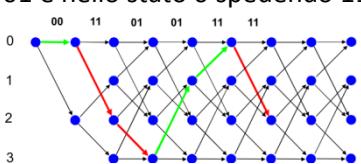
Arrivati a questa velocità, però ci ritroviamo con molti errori, per cui si preferisce inserire molta più ridondanza e utilizzare un forward error correction – FEC – per recuperare 6dB. Come funziona questo sistema?

Utilizziamo una macchina a stati finiti come in figura (detta anche **schema di Trellis**). Devo trasmettere 0, allora



trasmetto una coppia 00 e rimango nello stato 0; se devo trasmettere 1, allora trasmetto 11 e passo allo stato 2, dove devo necessariamente passare allo stato 1 trasmettendo 10 o allo stato 3 trasmettendo 01, e così via.

Esempio: devo trasmettere la sequenza 011001. Parto dallo stato 0 e ci rimango trasmettendo 00, passo nello stato 2 e spedisco la coppia 11. Vado nello stato 3 spedendo 01, successivamente passo nello stato 1 spedendo 01 e infine passo nello stato 1 spedendo 01 e nello stato 0 spedendo 11. Dopo tutti questi passaggi mi ritrovo con la sequenza 00 11 01 01 11 11 che è lunga il doppio dell'originale. In figura il cammino in dettaglio.



Spedisco la sequenza ed il canale me la modifica in 01 11 01 11 11 11, riesco a scoprirla? Vediamo...per farlo utilizzo la **decodifica di Viterbi**.

Parto dallo stato 0 e devo utilizzare il cammino 01, e immediatamente mi rendo conto che c'è un errore (la macchina a stati finiti non ha nessun cammino che partendo dallo stato 0 mi porti in un altro stato spedendo 01), per cui faccio un'ipotesi ovvero che la sequenza iniziale poteva essere 00 o 11. Se era 00 dovevo rimanere allo stato 0 altrimenti dovevo andare nello stato 2, in ogni caso la differenza tra i due stati è di 1, per cui mi riporto

D=1, ancora non ho mezzi per scegliere. Analizzo la coppia successiva e mi rendo conto che, tenendo conto del risultato precedente, ho una D=3, due D=2 e una D=1; la prima è poco probabile, la seconda è probabile mentre la terza è molto probabile, per cui scelgo quello con distanza minima. E se per caso ci fossero stati due D=1? Impossibile, in quanto la macchina è costruita in maniera tale da **evitare** che possano venir fuori due **distanze minime** uguali, in quanto da ogni stato si esce con due coppie a distanza massima; inoltre a distanza 2 ho tutte le possibili coppie, per cui non posso avere ripetizioni, ottenendo solo le 4 possibili combinazioni. Tutto questo sempre nell'ipotesi che ci sia un solo bit errato nella quaterna, altrimenti non funziona niente. Riepilogando questa macchina si basa su queste due regole:

- 1. Da uno stato emetto due coppie di valori a distanza massima**
- 2. A distanza due salti devo poter avere tutte e 4 le possibili combinazioni**

Con questo metodo correggo un bit errato su quattro, mentre con Hamming ne servono una valanga. CRC viene comunque utilizzato perché niente mi assicura che in quella quaterna ci siano 2 o più errori.

Ovviamente utilizzando un cavo Cat6 tutti questi problemi scompaiono.

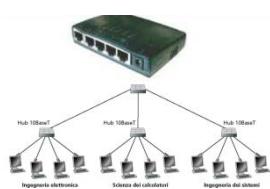
Tecnologia	Massima lunghezza del link	Codifica	Topologia del mezzo		Bit rate (bps)
10Base5	500 m	Manchester	bus	50-ohm coax	10 M
10Base2	185 m	Manchester	bus	50-ohm coax	10 M
10BaseT	100 m	Manchester	star	2 pair UTP cat. 3,4,5	10
100BaseFL	2000 m	Manchester	star	Multi-mode fiber*	10 M
100BaseT2	100 m	PAM 5x5	star	2 pairs UTP cat. 3,4,5	100 M
100BaseT4	100 m	8B/6T	star	4 pairs UTP cat. 3,4,5	100 M
100BaseTX	100 m	4B/5B with MLT-3	star	2 pairs UTP cat. 5	100 M
100BaseFX	412/2000 m	4B/5B with NRZI	star	Multi-mode fiber*	100 M
1000BaseT	100 m	PAM 5x5	star	4 pairs UTP Cat 5	1000 M
1000BaseSX	275 m	8B/10B	star	Multi-mode fiber†	1000 M
1000BaseLX	316/550 m	8B/10B	star	Multi-mode Fiber‡	1000 M
1000BaseCX	25 m	8B/10B	star	Twinax	1000 M

In questa tabella è presente il riepilogo di tutti i tipi di Ethernet e i cablaggi con i tipi di codifica utilizzati e tutto il resto.

## Hub Ethernet

La terminologia tende a evolversi nel tempo. All'inizio abbiamo detto che la LAN è una rete che si può estendere per massimo pochi chilometri, quando abbiamo parlato di Ethernet abbiamo stabilito una lunghezza massima di massimo 2500m, su un cavo unico coassiale è facile pensare ad un concetto di LAN come un unico mezzo broadcast, a cui tutti accedono e ben definito nello spazio. Quando il doppino sostituì il cavo coassiale, la LAN ha cambiato ulteriormente significato; l'Hub è un **concentratore** a cui arrivano tutte le connessioni dirette che vengono convogliate verso un'uscita. Questo dispositivo non fa altro che prendere il segnale e diramarlo così com'è per tutto il mezzo fisico ad

esso collegato, cosa accade in caso di collisione? Se due macchine generano un pacchetto, queste due frame si sovrappongono ed avviene una collisione, che viene sentita da tutte le macchine, per cui il sistema base di detect delle collisioni è invariato rispetto a quello del cavo coassiale. La LAN, per noi, è tutto il sistema presente in figura. Esistono diversi tipi di Hub:



- Attivo: alimentato elettricamente al suo interno presenta un amplificatore di segnale che permette al segnale di rimanere "quasi invariato" nel transito
- Passivo: non presenta l'amplificatore, per cui si limita a passare il segnale, che può perdere di intensità.
- Ibridi: sono particolari ed avanzati hub che permettono il collegamento tra più tipologie di cavo.

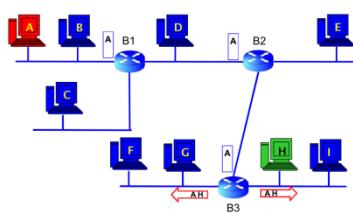


## Bridge Ethernet



E' un dispositivo che serve a connettere LAN differenti, sia dal punto di visto fisico che logico. Dovendo interconnettere una LAN Token Bus e una LAN Ethernet, serve un dispositivo che capisca come far viaggiare le frame da una parte all'altra, che non è una cosa assolutamente semplice. E se abbiamo due LAN che gestiscono le priorità e al centro una LAN che non gestisce le priorità? Come si risolve il problema? Cerchiamo di realizzare qualcosa che sia il più semplice possibile. Un bridge può interconnettere anche reti dello stesso tipo, ma anche qui non è una cosa semplice in quanto può generare problemi. Il bridge segue questa filosofia: trasmette la frame, se si perde pazienza, verrà ritrasmessa. Ogni cosa che si affaccia sul bridge è una LAN, ma se viene collegata una macchina che cos'è? A questo punto cambia la definizione di LAN, in quanto intendiamo tutto ciò che è visibile direttamente a livello DLL (es. una macchina trasmette una frame, fino a dove la frame può arrivare, quella è la LAN a cui la macchina appartiene), non importa quanti dispositivi bridge ci sono nel mezzo. Se la frame deve salire di livello, ovvero arrivare al livello di rete, allora lì la LAN termina. Solitamente il bridge presenta una porta che punta verso l'esterno, chiamata **uplink** che è di potenza superiore rispetto a tutte le altre presenti, in quanto convoglia tutto il traffico presente sul bridge verso l'esterno (es. 8 porte a 10Mbps e 1 porta a 100Mbps).

In bridge (o switch) se siamo solo ed esclusivamente in ambito Ethernet ha un interfaccia di rete per ogni collegamento in ingresso, per cui sente la frame in ingresso, la trasforma a livello DLL e la immagazzina in una memoria; in seguito un processore periodicamente esamina tutte le code in ingresso, guarda la frame e cerca di capire dove è diretta, se lo sa prende la frame e la inserisce nella coda di uscita. Ha, quindi, un processore, una memoria e un sistema operativo, anche quello da supermercato. Se conosce la destinazione è tutto ok, ma se non lo sa? Il bridge viene chiamato **dispositivo trasparente**, ovvero che una volta collegato non necessita di alcuna configurazione per funzionare; ovviamente prima di funzionare qualche ragionamento se lo deve fare da solo.



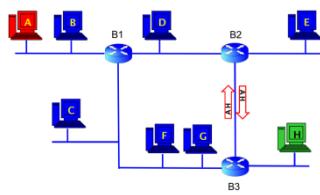
Supponiamo di avere la rete presente in figura dove la macchina A vuole comunicare con la macchina H; prepara quindi la frame dove inserisce il suo nome ed il destinatario e la mette sul canale, se la macchina H sta nella sua LAN, allora risponde immediatamente perché sente il pacchetto. Nel nostro caso non la sente, il bridge B1 se sa che A e H sono dalla stessa parte allora non deve fare nulla, ma se invece non sa nulla allora prende la frame e la ripropone per tutte le sue uscite, anche quella da cui ha ricevuto la frame, inoltre in una tabellina si segna in che uscita si trova A (sta imparando da dove viene). B2 fa la stessa cosa e impara dove si trova A e non sapendo dove si trova H ripropone il segnale da tutte le parti, la stessa filosofia viene applicata da tutti i bridge che il pacchetto incontra nel suo cammino. Questo viene anche chiamato **schema di indirizzamento flat**. Bisogna notare che il bridge non conosce la topologia della rete, sanno soltanto chi si trova nelle varie porte, ha una

visione **molti locale**. Una volta che H riceve il suo pacchetto, risponde e mediante le tabelline interne nei vari bridge, sanno dove si trova A e quindi in quali porte instradare il pacchetto, nel frattempo si scrivono dove si trova H. Gli hub invece semplicemente sentono e ripropongono, in pratica gli hub sono scemi, inoltre A e H comunicando mediante hub disturbano sempre anche gli altri. Il dispositivo di memorizzazione pacchetto del bridge, permette di rinviare il pacchetto direttamente dall'ultimo bridge attraverso (se dovesse verificarsi una collisione), evitando così di ripercorrere tutto il percorso. Esistono hub e switch da 10MB, solo che nel caso degli hub sono 10MB suddivisi in tutta la rete a cui sono collegati, nel caso dello switch invece sono 10MB da ingresso a ingresso (uno switch di 8 porte, quindi con 4 collegamenti porta-porta, deve essere in grado di gestire un massimo di 40MB di traffico). Ovviamente più aumentano il numero di porte in uno switch, più il dispositivo costa, in quanto necessità di un hardware molto più potente per gestire il traffico tra tutte le porte ad una velocità che sia adeguata. Se il processore è troppo lento allora non riesce ad elaborare i dati in tempo, rischiando di perdere pacchetti per strada. Quindi ogni rete, man mano che cambia la complessità, necessita di un dispositivo di complessità differente.

Lo switch permette di separare il traffico, in questo modo aumentano le prestazioni; solo nel momento iniziale abbiamo un traffico broadcast, ma avviene solo una volta. Che accade se sposto lo switch? La tabella ha un tempo di vita? Cosa accade se invece sposto la macchina? Per ogni interfaccia viene inserito l'indirizzo **MAC** ed un tempo, in

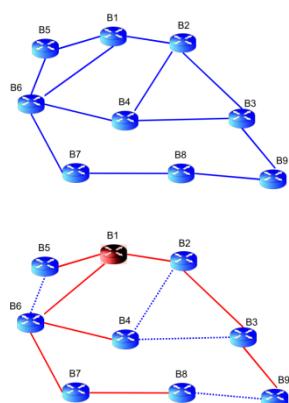
modo che se vedo del traffico da parte di quell'indirizzo aggiorno il tempo; se ad un certo punto non vedo più traffico da quell'indirizzo, allora quel valore lì lo cancello. In questo modo mantengo la tabella sempre giovane; il meccanismo avviene anche se non sposto nulla. Se sposto lo switch ovviamente la ram si cancella, per cui deve ricominciare d'accapo, se sposto il pc invece vale il discorso precedente. Ovviamente se inizio a spostare il pc in continuazione potrei fare impazzire lo switch, rischiando di non comunicare per un bel po'. E se per disgrazia due costruttori hanno lo stesso MAC address? Allora si permette all'utente di cambiare il MAC address alla scheda, con tutti i problemi che esso comporta. La sicurezza basata sul MAC address è una enorme cavolata, in quanto l'indirizzo MAC address è un'auto dichiarazione, ma non da la certezza che corrisponde all'identità.

## Spanning Tree Protocol (IEEE 802.1d)



Supponiamo di avere una rete come in figura (inoltre questo è un esempio molto semplice), ovvero che gli switch sono disposti tra loro in modo da formare un grafo. Ad un certo punto assistiamo ad una proliferazione di copie del pacchetto e in casi di reti più complesse la situazione potrebbe diventare abbastanza intricata. Dobbiamo risolvere questo problema. L'unico modo di risolvere il problema è quello di tagliare il grafo e farlo diventare un albero.

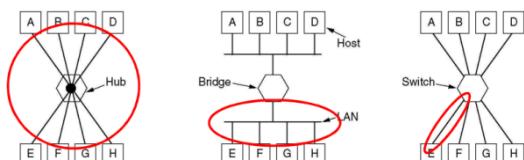
Lo standard **802.1d** ci dice: prendiamo una serie di switch, attacchiamoli sottoforma di grafo e in qualche modo devono capire come tirare fuori una struttura ad albero. Domanda: ma se sappiamo che la struttura a grafo è un



problema, perché non progettare la rete a priori come se fosse un albero? Perché il grafo ci garantisce un'alta resistenza ai guasti, che l'albero non riesce a garantirci. B1 manda un pacchetto ai suoi vicini e si identifica, i suoi vicini rispondono con i loro identificativi e successivamente si fa un confronto per capire qual è l'identificativo più alto e quello più basso, una volta stabilito chi vince lo comunica agli altri. Nel nostro esempio B1 ha l'identificativo più alto, per cui diventa radice, B5, B2 e B6 invece diventano nodi di primo livello e se lo comunicano tra di loro, in modo da localizzarsi a vicenda. B6 poi parla con B4 e B7 dicendogli che è sotto B1, inviando l'identificativo di quest'ultimo a B4 e B7; se l'id di B1 è sempre il più alto non cambia nulla, altrimenti viene sostituito da quello che lo batte. Con un po' di questi passaggi e sapendo quando terminare l'algoritmo, è possibile identificare quali di questi nodi ha l'id più alto, in modo da tirare per prima cosa fuori la radice e in seguito si calcolano gli altri livelli di nodi. Non sono i percorsi migliori, ma è un albero di copertura che ha come radice il nodo con l'id scelto. Tutto ciò avviene sempre in modalità trasparente.

## Riepilogo hubs, bridges e switches

Nel caso di hub il dominio di collisione è tutta la rete, nel caso di bridge (ricordiamo si parla di bridge quando si ha un



dispositivo che interconnecte LAN) parliamo di dominio di collisione identificando tutta la LAN che sta da una parte o dall'altra del bridge.

Nel caso dello switch il dominio è nell'collegamento diretto tra macchina e switch, in pratica non c'è collisione. Nel caso dello switch si perde completamente il significato di LAN. Gli hub praticamente non li produce più nessuno. Alcuni bridge permettono di collegare due porte uplink verso la stessa sorgente, come se fossero in parallelo, in modo da bilanciare e potenziare il traffico. In uno switch metto una quantità di memoria minima ed indispensabile, anche perché devo evitare di stressare il processore.

## Indirizzi MAC (by wikipedia)



L'originale **Indirizzo MAC IEEE 802**, ora chiamato ufficialmente "MAC-48", deriva dalla specifica dell'Ethernet. Poiché chi inizialmente progettò l'Ethernet ebbe la previdenza di usare uno spazio indirizzi a 48-bit, adesso disponiamo potenzialmente di ben  $2^{48}$  (cioè 281.474.976.710.656) possibili indirizzi MAC, un numero che è praticamente impossibile raggiungere prima che le schede ethernet cambino standard.

In tale formato (quello attualmente più diffuso), i 48 bit del codice sono suddivisi in 12 cifre esadecimale: le prime 6 cifre individuano il produttore dell'interfaccia di rete mentre le successive corrispondono al numero di serie della scheda stessa. L'indirizzo MAC si scrive normalmente in 6 ottetti separati da un trattino (es. 00-50-FC-A0-67-2C) ed i primi 3 ottetti sono detti **OUI (Organizationally Unique Identifier)**. Per questo tipo di indirizzi di solito si preferisce la notazione esadecimale anche per differenziarla dagli **indirizzi IP** che usano la notazione decimale.

I tre sistemi di numerazione utilizzano lo stesso formato, differendo soltanto nella lunghezza dell'identificatore. Gli indirizzi si dividono in "indirizzi universally administered" e "indirizzi locally administered".

Gli indirizzi **universally administered** vengono assegnati ai dispositivi dal loro produttore; e vengono talvolta chiamati "indirizzi burned-in". I primi tre **ottetti** (in ordine di trasmissione) identificano l'organizzazione o il produttore che ha emesso l'identificatore e rappresentano l'Organizationally Unique Identifier (OUI). I successivi tre (MAC-48 e EUI-48) o cinque (EUI-64) ottetti sono assegnati dal produttore che deve rispettare il solo vincolo dell'univocità. L'IEEE prevede che lo spazio MAC-48 non sia esaurito prima dell'anno 2100, mentre non ci si aspetta l'esaurimento degli indirizzi EUI-64 in un futuro ragionevolmente vicino.

Gli indirizzi **locally administered** vengono assegnati ad un componente dall'amministratore di rete, annullando l'indirizzo burned-in. Gli indirizzi locally administered non contengono gli ottetti OUI.

Gli indirizzi universally administered e locally administered si distinguono per mezzo del settaggio del secondo **bit meno significativo** del byte più significativo dell'indirizzo. Se il bit è a 0, l'indirizzo è *universally locally administered* o *local scope*; se viceversa è a 1, esso è *locally globally administered* o *global scope*. Tale bit vale 0 in tutti gli OUI. Per esempio, 02-00-00-00-00-01 è un indirizzo locally administered.

Gli indirizzi MAC-48 ed EUI-48 vengono solitamente rappresentati in formato esadecimale, separando ciascun ottetto con un trattino o con i due punti. Un esempio di indirizzo MAC-48 è "00-08-74-4C-7F-1D". Confrontando i primi tre ottetti con le assegnazioni OUI dell'IEEE, si può osservare come esso appartenga alla Dell Corporation, mentre gli ultimi tre ottetti rappresentano il numero seriale assegnato al componente dal produttore.

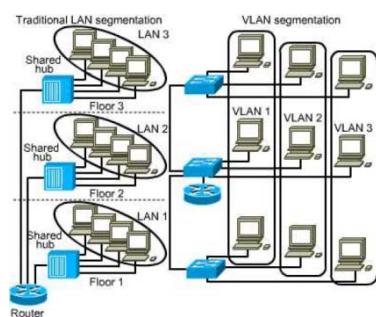
Ogni scheda ha un indirizzo unico perché i primi 24 bit sono identificativi della casa produttrice e i successivi della scheda. In questo modo ogni casa produttrice ha a disposizione  $2^{24}$  indirizzi, quindi può produrre più di 16 milioni di schede; se un produttore ne produce meno, gli indirizzi (a 48 bit) non assegnati vengono persi, non potendo essere utilizzati da altri costruttori.

Si comprende, quindi, come l'indirizzo MAC non cambi se si sposta una scheda di rete da una LAN ad un'altra, mentre può cambiare l'indirizzo IP.

La conversione tra indirizzo MAC e indirizzo IP avviene mediante alcuni protocolli, il più conosciuto è **ARP**.

## VLAN (Virtual LAN - 802.1q)

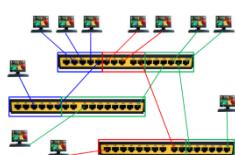
Il traffico non è sempre diretto ad una sola macchina, in quanto ogni tanto c'è un po' di traffico broadcast che disturba, possiamo separare le situazioni? Supponiamo di avere una serie di macchine logicamente distinte, come



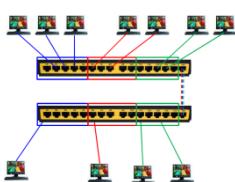
detto in precedenza conviene prendere diversi switch e collegare ad ognuno i gruppi di macchine che devono stare separati. Per risparmiare mi basterebbe prenderne uno bello grosso e collegare ad esso tutte le macchine dicendo al software di trattare ogni gruppo di porte (es. switch 12 porte – 1-2-3 – 4-5-6 – 7-8-9 – 10-11-12) come una LAN a se stante, cioè fisicamente tutte le porte possono comunicare tra di loro, però a livello software si creano delle camere stagna. Avviene cioè quello che si vede in figura (quello in figura però raggruppa macchine di switch differenti) e questo sistema viene detto **VLAN** (Virtual LAN). Il sistema è utile anche per raggruppare macchine appartenenti a switch differenti. La segreteria studenti è presente al primo piano e utilizza un unico server presente in dipartimento, poiché il server è abbastanza prezioso bisogna collocarlo in un'area differente, magari più sicura. Allora che faccio, tiro un unico lungo cavo fino al server? Oppure mi gestisco

Istradamento via software? Mi conviene gestire il tutto via software utilizzando le VLAN. Come si implementa tutto ciò? Inizialmente si interroga il dispositivo e gli si chiede quante porte ha, dopo di che si numerano le porte e si assegnano alle diverse VLAN. Lo scopo delle VLAN:

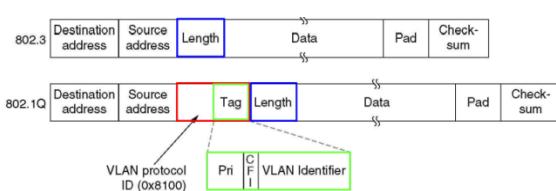
- **Risparmio:** riutilizzo delle linee e degli apparati preesistenti
- **Flessibilità:** facile spostamento fisico degli utenti
- **Aumento prestazioni:** il traffico broadcast viene confinato
- **Sicurezza:** gli utenti di VLAN differenti non vedono i reciproci frame dati



Quando lo switch viene **logicamente partizionato** in più parti, assegnando le singole porte alle varie VLAN abbiamo una **VLAN port based (untagged)**. Per realizzarne una è sufficiente uno switch che supporti il protocollo 802.1q. In questo caso c'è un collegamento diretto tra diverse VLAN (e a mio modo di vedere anche una bella confusione).



Qualcuno ha pensato, invece di creare un casino con i cavi per collegare anche tra di loro gli switch - sprecando così anche porte e aumentando considerevolmente il numero di switch necessari – cerco di realizzare un unico collegamento tra switch che faccia passare tutti i segnali delle varie VLAN, come si vede in figura. Ovviamente questo si può fare (sistema **tagged**), ma si deve fare qualche giochetto con la frame, in quanto al suo interno bisogna



inserire il valore che la lega alla VLAN di destinazione; nella frame standard c'è spazio? No, per cui bisogna allargare la frame. In figura abbiamo la frame standard 802.3 e sotto la frame 802.1q che ha in più, al suo interno, le informazioni aggiuntive per l'utilizzo della VLAN.

- **Pri:** ci indica la priorità
- **CFI:** indica se è una frame normale o VLAN
- **VLAN Identifier:** l'identificatore della VLAN a cui appartiene la frame

A questo punto posso implementare il meccanismo. Le frame che viaggiano tra le macchine e lo switch sono secondo lo standard 802.3, mentre quelle che viaggiano tra switch e switch (ovviamente se implementata una VLAN) seguono lo standard 802.1q.

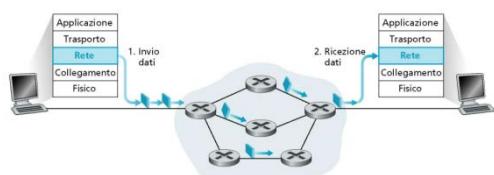
## Parte 4 – Livello di Rete



Il livello di rete presuppone ancora collegamenti a saltare attraverso macchine intermedie, il suo scopo principale è trovare la destinazione. Ovvero saltare attraverso diverse macchine intermedie che utilizzano anche esse un livello di rete. Esistono due tipologie di servizio a livello di rete: **Datagram** e a **circuito virtuale**.



### Servizi Datagram e a circuito virtuale



Il servizio **Datagram** funziona come la figura a lato, ovvero i pacchetti entrano nella rete, ognuno va per i fatti propri e tutti devono arrivare a destinazione in un ordine di arrivo non definito. Ovviamente affinché si possa fare una cosa del genere ogni nodo intermedio deve avere un modo per poter fare routing, ogni pacchetto deve essere identificato con gli indirizzi IP mittente e destinazione, altrimenti gli algoritmi di routing non li possono instradare. La comunicazione dipende da un insieme di fattori quali: traffico, dati e molto altro. Tutti questi fattori vengono definiti con il nome di **metrica**.



Il servizio a **circuito virtuale** invece utilizza un canale virtuale logico attraverso cui viaggiano i pacchetti, la difficoltà iniziale sta nel creare, mediante algoritmi di routing, il canale virtuale, però una volta creato può essere numerato

banalmente e utilizzare questo id per instradare i pacchetti. Questo identificatore viene chiamato **identificatore di flusso**. Il lavoro del router in questo caso è molto semplice, in quanto deve solo verificare l'id e instradarlo nel relativo canale. Metodo molto veloce, ma non è garantito l'arrivo, però garantisce i tempi di arrivo, a meno della prima operazione.

Nella connessione datagram le congestioni non sono molto gestibili, in quanto c'è il rischio di sovraccaricare un povero router che in un momento è inutilizzato e si ritrova sommerso, inoltre non è evitabile in quanto i router decidono in maniera spontanea dove instradare il traffico. Nella connessione virtuale non dovrebbe esserci congestione, in quanto al momento di creare il circuito la macchina chiede a quelle vicine se riescono a gestire il futuro traffico, chi riesce si prende l'incarico e una volta stabilito il percorso tutte le macchine sono d'accordo. Un po' di congestione ci sarà comunque, in quanto la dichiarazione iniziale è sempre al risparmio, per cui bugia qui e bugia là, le macchine potrebbero proprio non essere in grado di gestire il traffico, ma è comunque una situazione abbastanza gestibile.

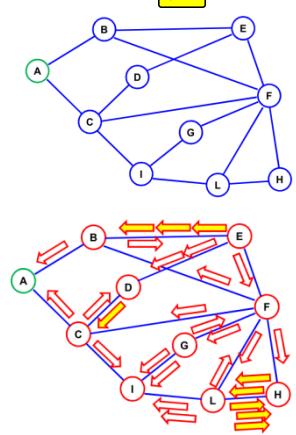
Quindi ricapitolando facciamo una tabella di confronto:

	<b>Datagram</b>	<b>Circuito Virtuale</b>
<b>Creazione circuito</b>	No	Si (porta del ritardo iniziale)
<b>Informazione di Stato</b>	Ogni pacchetto deve avere indirizzo mittente e destinatario	Solo indirizzo circuito (molto veloce e leggero)
<b>Instradamento</b>	Nessuna informazione – pacchetti non fanno mai la stessa strada	Ogni circuito virtuale deve essere identificato
<b>Effetti di guasti nei router</b>	Nessuno, a parte i pacchetti persi	Il percorso legato ai router guasti crollano – recupero complesso
<b>Controllo della congestione</b>	Complesso – demandato ai livelli superiori	Semplice

## Generalità su algoritmi di Routing

- **Statico:** decido una volta per tutte e rimane quello
- **Dinamico:** vedo com'è la situazione dei link e in caso di congestione decido che fare. Cerco di decidere con intelligenza. L'hot potato è un esempio, prendo il pacchetto, cerco il link più scarico e gli sbologno il tutto.
- **Locale:** ogni router vede la situazione di se stesso e dei suoi vicini e in base a questo decide. E' quello più comodo, perché ognuno decide che fare, il problema è che due router vicini possano andare in opposizione.
- **Globale:** devo riuscire a fare una fotografia di tutto il mio sistema e in base a quella eseguire le operazioni di routing. Ovviamente è quello più efficace, solo che non è assolutamente semplice da realizzare.

## Flooding



Ipotizziamo di avere una rete a grafo come in figura, e che la macchina A deve comunicare con la macchina H. A non conosce niente, ovvero è a conoscenza delle sue sole uscite. Il **flooding** funziona che il pacchetto arriva e viene replicato così com'è in tutte le uscite presenti, tranne quella da cui è arrivato. Ad alcuni nodi arriveranno i pacchetti doppi e riusciranno a capire che si trattano di copie solo se noi inseriamo un identificatore nel pacchetto, inoltre questo id deve essere memorizzato dal nodo per un tempo sufficiente. Per un pacchetto singolo è più o meno banale...ma per milioni? Diventa un macello sia dati di memorizzare (si necessita una grossa memoria e un potente processore) sia di lentezza. Il pacchetto finalmente riesce ad arrivare ad H, ma chi è che comunica che il pacchetto è arrivato? Nessuno! Flooding significa proprio questo, inondazione, per cui ad ogni passaggio riempiremo la rete di pacchetti copia, per cui dobbiamo capire quando fermare il tutto. La cosa più semplice sarebbe inserire un id nel pacchetto, ma non è conveniente. Altra soluzione è mettere un contatore di salti, ovvero dopo 10 salti il pacchetto lo butta via, il problema è che il numero deve essere un valore adeguato che permetta comunque al pacchetto di arrivare (se esaurisce i salti prima di arrivare si perde l'informazione); il numero sarebbe il diametro del grafo, solo che calcolare il diametro di internet è praticamente

impossibile. La terza soluzione sarebbe far diventare questo grafo un albero, per cui eliminarremmo completamente il problema; solo che togliendo la struttura a grafo perderemmo i vantaggi dei percorsi multipli. I vantaggi del flooding il pacchetto è sicuro che arriva e troviamo sicuramente il percorso migliore, solo che intaseremo la rete dopo pochissimi passaggi. E' un ottimo algoritmo di attacco. Può essere utilizzato, infine anche come **metrica** di confronto per altri algoritmi di routing: il flooding segue sempre il percorso più breve poiché utilizza ogni possibile percorso in parallelo, ne segue che nessun altro protocollo riesce ad ottenere ritardi inferiori.

**IPv4**

Nacque con grazie a **TCP** che prima si chiamava **Transfer Control Protocol**, perché si occupava di realizzare tutto il livello di trasporto; poi lo si delegò alla sola trasmissione, divenendo **Transmission Control Protocol**, delegando il trasporto al protocollo **IP** (Internet Protocol). Un indirizzo IPv4 (notazione decimale-punteggiata).

172 . 16 . 254 . 1  
 ↓      ↓      ↓      ↓  
 10101100 . 00010000 . 11111110 . 00000001  
 Un byte = otto bit  
 Trenta due bit (4 \* 8), o 4 byte

Le altre versioni di IP sono cadute nel dimenticatoio. IP si basa su uno strato di collegamento i rispettiamo lo standard di IP. Si interfaccia an

ne funzionare anche sui fogli di carta se porto che sono **UDP** e **TCP**.

Sono numeri a 4 byte, nello standard vengono rappresentati con la notazione puntata utilizzando sia i valori binari (scomodi) che i valori decimali (largamente più utilizzati). 4 byte, fanno circa 4 miliardi di combinazioni, ma siamo arrivati lo stesso alla saturazione.



L'indirizzo IP viene suddiviso in due parti, la prima che identifica la **rete** e la seconda che identifica l'**host**. A loro volta quindi otteniamo 5 diversi tipi di gruppi di indirizzi IP.



	8	16	24	32
CLASSE A	0  Ident. rete	Identificatore di host		
CLASSE B	1 0  Identificatore di rete	Identificatore di host		
CLASSE C	1 1 0  Identificatore di rete	Identificatore di host	Ident. di host	
CLASSE D	1 1 1 0	Indirizzo multicast		
CLASSE E	1 1 1 1 0	*Usi futuri*		
<b>Classe</b>	<b>Da</b>	<b>A</b>		
A	1.0.0.0	127.255.255.255		
B	128.0.0.0	191.255.255.255		
C	192.0.0.0	223.255.255.255		
D (multicast)	224.0.0.0	239.255.255.255		
E (riservati)	240.0.0.0			

Il tipo **A** è quello che ha il primo bit dell'indirizzo **0**. Il primo bit identifica massimo 128 reti ed il resto 16 milioni di host. La cosa che risalta un po' all'occhio è che secondo la suddivisione iniziale, il routing si fa solo nella parte di Rete, per cui tutti gli host sono senza routing o almeno lo hanno solo al livello DLL. Il tipo **B** inizia con **10**. I primi due bit rappresentano la rete, il resto gli host; identifica circa 16 mila reti e 65 mila host. Il tipo **C** inizia con **110** ed è molto squilibrato. I primi tre bit identificano massimo 2 milioni di reti e il restante circa 254 host. Il tipo **D** inizia con **1110**, riservato ad uso multicast, ovvero il pacchetto viene spedito a più macchine contemporaneamente (i precedenti sono di tipo unicast, ovvero da macchina a macchina). Il tipo **E** inizia con **11110** e sono riservati per usi futuri, ovvero completamente inutilizzati.

In figura abbiamo i range di indirizzi definiti fai vari tipi.



Alcuni indirizzi sono stati riservati per scopi particolari, ad esempio:

- un indirizzo di tutti 0 dovrebbe essere illegale, però viene utilizzato solitamente per indicare l'host stesso.
- Un indirizzo che ha tutti 0 nella parte riservata alla rete e diverso da 0 nella parte riservata all'host, indica un indirizzo della stessa sotto rete, ovvero il pacchetto non seguirà percorsi di routing ma rimarrà nella sottorete.
- Un indirizzo di tutti 1 (255.255.255.255) indica broadcast totale, ovvero mandando un pacchetto di questo tipo va a tutte le macchine. In teoria si dovrebbe propagare per il mondo, in pratica non supera il router, che appena vede un pacchetto del genere non lo fa passare.
- Il range da 127.0.0.0 a 127.255.255.255, ovvero 16 milioni di possibilità, indica gli indirizzi di Loopback (ovvero l'indirizzo che indica la stessa macchina). Perché mettere 16 milioni di possibilità, se scrivere, ad esempio, 127.0.0.1 e 127.0.0.50 è la stessa cosa? Boh! Volevano metterne tanti (non esiste una spiegazione logica).

- gli indirizzi IP Privati, che servono a mettere su delle reti private che non devono accedere ad internet. Un router che vede un pacchetto con un indirizzo privato, non lo fa passare. Anche in questo caso hanno distinto gli indirizzi privati in A, B, C e D ovvero rispettivamente:
  - 10.0.0.0 – 10.255.255.255
  - 169.254.0.0 – 169.254.255.255
  - 172.16.0.0 – 172.31.255.255
  - 192.168.0.0 – 192.168.255.255

L'organo che si occupa di assegnare gli indirizzi IP è lo **IANA** (*Internet Assigned Numbers Authority*), che a sua volta nel 1992 è stato suddiviso in diversi **RIR** (*Regional Internet Registry*), per gestire gli indirizzi e i nomi di dominio per ciascuna area assegnata. Esistono cinque RIR:

- **APNIC** (*Asia Pacific Network Information Center*)
- **ARIN** (*American Registry for Internet Numbers*)
- **LACNIC** (*Latin American and Caribbean Internet Addresses Registry*)
- **RIPE NCC** (*Réseaux IP Européens Network Coordination Centre*)
- **AFRINIC** (*African Regional Internet Registry*)

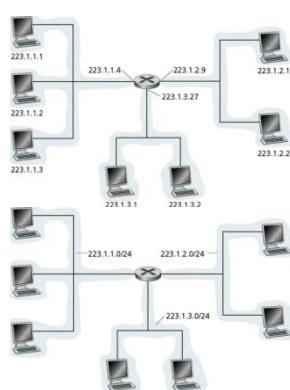
Ai RIR fanno capo i **LIR** (*Local Internet Registries*) per gestire gli indirizzi a livello locale. L'Università di Catania fa parte del **GARR-LIR**. 

## Le maschere di sottorete

Si profila un nuovo problema. Immaginiamo di scegliere una famiglia di indirizzi di tipo D, ovvero la capacità di gestire circa 65 mila macchine, per cui dovremmo fare Routing con questa mole di macchine. Ricordando lo schema di funzionamento dei bridge ci accorgiamo subito che, con una rete di queste dimensioni, il bridge dovrebbe tenere in memoria anche 65 mila entry, che sono un po' troppe. Converrebbe spezzettare in subnet e in questo caso che ne abbiamo 65 mila si potrebbe fare, e se invece ne avessimo 16 milioni? Dobbiamo trovare una soluzione. Viene adottato un nuovo sistema di indirizzamento detto **CIDR** (*Classless Inter-Domain Routing*), che si basa sull'utilizzo delle **maschere**; una maschera sostanzialmente è un qualcosa che si mette davanti e fa passare quello che vogliamo. Come funziona? Vediamolo!

Immaginiamo di avere l'indirizzo in notazione decimale puntata **151.097.252.066** e la subnet mask **255.255.254.000**, trasformiamoli in binario e applichiamo un operazione di AND bit a bit tra i due indirizzi:

<b>10010111.1100001.11111100.01000010</b>	<b>151.097.252.066</b>
<b>11111111.11111111.11111110.00000000</b>	<b>255.255.254.000</b>
<hr/>	
<b>10010111.1100001.11111100.00000000</b>	<b>151.097.252.000</b>



Ovvero questo sistema ci permette di effettuare operazioni di Routing solo ed esclusivamente sulla parte di rete dell'indirizzo; in particolare, una volta individuata la sottorete di appartenenza, il protocollo IP opererà l'istradamento indiretto tramite la parte Net-Id dell'indirizzo per raggiungere quella sottorete, seguito poi dall'istradamento diretto tramite l'Host-Id per raggiungere l'host in quella sottorete mediante i protocolli della sottorete locale (ovvero a livello DLL). La maschera di sottorete permette al dispositivo di rete di ricercare il destinatario all'interno di un range ben definito senza dover ricorrere all'uso di router che funga da gateway con un'altra rete. La notazione **223.1.1.0/24** indica che in quell'indirizzo bisogna prendere esclusivamente i primi 24 bit e portare a 0 i rimanenti, ovvero all'indirizzo originario è stata applicata una maschera di sottorete

avente **24 bit 1 e i restanti a 0**. Per determinare il numero massimo di indirizzi utili in una subnet basta contare applicare la formula  $2^{32-m} - 2$  dove  $m$  rappresenta il numero di bit 1 presenti nella maschera (sottraiamo alla totalità degli indirizzi generati i due indirizzi riservati, uno indica la subnet stessa, l'altro è usato per fare broadcast).

Es.

Supponiamo di avere una macchina con:

- Indirizzo IP: 192.168.32.97
- Subnet mask: 255.255.255.224

E richiediamo di connetterci all'indirizzo IP 192.168.32.130

Innanzitutto applichiamo la subnet mask per ricavare le sottoreti di appartenenza

11000000.10101000.00100000.01100001	192.168.032.097
11111111.11111111.11111111.11100000	255.255.255.224
-----	-----
11000000.10101000.00100000.01100000	192.168.032.096
11000000.10101000.00100000.10000000	192.168.032.130
11111111.11111111.11111111.11100000	255.255.255.224
-----	-----
11000000.10101000.00100000.10000000	192.168.032.128



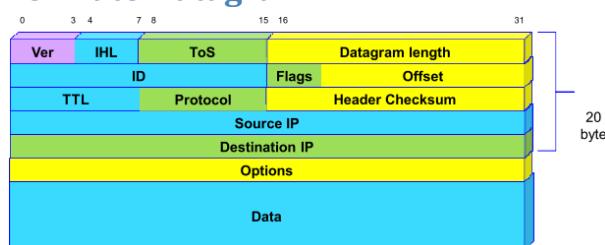
I risultati indicano due sottoreti differenti, per cui le macchine appartengono a sottoreti differenti.

Con la subnet mask 255.255.255.224 è possibile avere un range di 30 indirizzi utili, in quanto  $2^{32-27}-2=2^5-2=30$ , per cui:

- L'indirizzo 192.168.32.97 appartiene al range tra 192.168.32.96 e 192.168.32.127
- L'indirizzo 192.168.32.130 appartiene al range tra 192.168.32.128 e 192.168.32.159

Se l'IP di destinazione fosse stato 192.168.32.100, i due indirizzi avrebbero indicato macchine appartenenti alla medesima subnet.

## Formato Datagram IPv4



Come vediamo in figura, l'header del pacchetto IPv4 consiste in 13 campi di cui 1 opzionale e chiamato con il nome di *Options*. I campi sono inseriti col byte più significativo messo per primo e all'interno dei singoli byte il bit più significativo è il primo (quello di indice 0). Vediamo ora a cosa corrispondono i vari campi:

- **Versione [4 bit]** – indica la versione del datagramma IP: per IPv4 ha valore 4.
- **Internet Header Length (IHL) [4 bit]** – indica la lunghezza (in word da 32 bit) dell'header del datagramma IP, ovvero l'offset del campo dati; tale lunghezza può variare da 5 word (20 byte) a 15 word (60 byte) a seconda della presenza e della lunghezza del campo Options.
- **Type of Service (ToS) [8 bit]** – nelle specifiche iniziali del protocollo, questi bit servivano all'host mittente per specificare il modo e in particolare la precedenza con cui l'host ricevente doveva trattare il datagramma:
  - bit 0-2: precedenza
  - bit 3: Latenza (0 = normale, 1 = bassa)
  - bit 4: Throughput (0 = normale, 1 = alto)
  - bit 5: Affidabilità (0 = normale, 1 = alta)
  - bit 6-7: riservati per usi futuri

Ad esempio un host poteva scegliere una bassa latenza, mentre un altro preferire un'alta affidabilità. Nella pratica questo uso del campo ToS non ha preso largamente piede. Ultimamente sono stati ridefiniti ed

hanno la funzione di **Differentiated Services**, necessari per le nuove tecnologie basate sullo streaming dei dati in tempo reale.

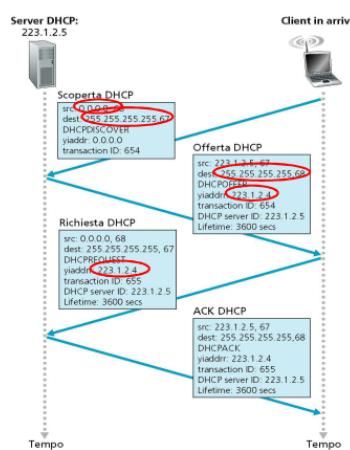
- **Total Length [16 bit]** – Indica la dimensione (in byte) dell'intero datagramma, comprendendo header e dati; tale lunghezza può variare da un minimo di 20 byte(header minimo e campo dati vuoto) ad un massimo di 65535 byte. In ogni momento, ad ogni host è richiesto di essere in grado di gestire datagrammi aventi una dimensione minima di 576 byte mentre sono autorizzati, se necessario, a frammentare datagrammi di dimensioni maggiori.
- **Identification [16 bit]** – Utilizzato, come da specifiche iniziali, per identificare in modo univoco i vari frammenti in cui può essere stato “spezzato” un datagramma IP. Alcune sperimentazioni successive hanno però suggerito di utilizzare questo campo per altri scopi, come aggiungere la funzionalità di tracciamento dei pacchetti.
- **Flags [3 bit]** – bit utilizzati per il controllo del protocollo e della frammentazione dei datagrammi:
  -  ○ **Reserved** – sempre settato a 0.
  -  ○ **DF (Don't Fragment)** – se settato a 1 indica che il datagramma non deve essere frammentato; se tale datagramma non può essere inoltrato da un host senza essere frammentato, viene semplicemente scartato. Utile per ispezionare le capacità di gestione dei vari host del percorso di routing.
  -  ○ **MF (More Fragments)** – se settato a 0 indica che il datagramma è l'ultimo frammento o il solo frammento del datagramma originario, pertanto tutti gli altri suoi frammenti hanno il bit MF settato a 1. Naturalmente, questo bit sarà sempre 0 anche in tutti i datagrammi che non sono stati frammentati.
- **Fragment Offset [13 bit]** – Indica l'offset (misurato in blocchi di 8 byte) di un particolare frammento relativamente all'inizio del datagramma IP originale: il primo frammento ha offset 0. L'offset massimo risulta pertanto pari a  $(2^{13} - 1) * 8 = 65528$  byte che, includendo l'header, potrebbe eccedere la dimensione massima di 65535 di un datagramma IP.
- **Time to Live (TTL) [8 bit]** – Indica il tempo di vita del datagramma, necessario per evitarne la persistenza indefinita sulla rete nel caso in cui non si riesca a recapitarlo al destinatario. Storicamente il TTL misurava i “secondi di vita” del datagramma, mentre ora esso misura il numero di “salti” da nodo a nodo della rete: ogni router che riceve il datagramma prima di inoltrarlo ne decrementa il TTL, quando questo arriva a zero il datagramma non viene più inoltrato ma scartato. Tipicamente, quando un datagramma viene scartato per esaurimento del TTL, viene automaticamente inviato un messaggio ICMP al mittente del datagramma, specificando il codice di *Richiesta scaduta*; la ricezione di questo messaggio ICMP è alla base del meccanismo traceroute.
- **Protocol [8 bit]** – indica il codice associato al protocollo utilizzato nel campo dati del datagramma IP: per esempio al protocollo TCP è associato il codice 6, a UDP il codice 17, mentre a IPv6 è associato il codice 41.
- **Header Checksum [16 bit]** – È un campo usato per il controllo degli errori dell'header. Ad ogni hop, il checksum viene ricalcolato e confrontato con il valore di questo campo: se non c'è corrispondenza il pacchetto viene scartato. E' da notare che non viene effettuato alcun controllo sulla presenza di errori nel campo Data deputandolo ai livelli superiori. Questo campo non viene praticamente utilizzato da nessuno.
- **Source address [32 bit]** - Indica l'indirizzo IP associato all'host del mittente del datagramma.  
Da notare che questo indirizzo potrebbe non essere quello del "vero" mittente nel caso di traduzioni mediante NAT. Infatti, qualora un host intermedio effettui questa traduzione, sostituisce l'indirizzo del mittente con uno proprio, procurandosi poi di ripristinare l'indirizzo originario su tutti i messaggi di risposta che gli arrivano destinati al mittente originario.
- **Destination address [32 bit]** - Indica l'indirizzo IP associato all'host del destinatario del datagramma e segue le medesime regole del campo Source address.
- **Options** - Opzioni (facoltative e non molto usate) per usi più specifici del protocollo, come informazioni sui router. Si ricorda che il valore del campo IHL deve essere sufficientemente grande da includere anche tutte le opzioni e, nel caso queste siano più corte di una word, il padding necessario a completare i 32 bit. Inoltre, nel caso in cui la lista di opzioni non coincida con la fine dell'header, occorre aggiungere in coda ad essa un

marcatore EOL (End of Options List). C'è da notare infine che, potendo causare problemi di sicurezza, l'uso delle opzioni LSSR e SSRR (Loose e Strict Source and Record Route) è scoraggiato e molti router bloccano i datagrammi che contengono queste opzioni.

Normalmente se mando un pacchetto, voglio la risposta, per cui anatomici non ce ne possono essere. Se io voglio attaccare una macchina come faccio? Se mi voglio fare beccare subito allora basta utilizzare il proprio pc senza fare nulla (XD). Se invece non voglio farmi beccare allora conviene utilizzare una terza macchina che faccia da ponte, in modo da poter nascondere la mia identità dietro un ignaro utente. Per cui mai sottovalutare la messa in interne di una macchina per prima cosa, in quanto può essere usata come base di attacco da parte di terzi e soprattutto registrare tutto quello che fa quella macchina e salvarlo su una macchina esterna.

## DHCP (Dynamic Host Configuration Protocol)

Che accade se sposto una macchina da una sottorete all'altra? Devo cambiare necessariamente indirizzo. Una volta non era un problema, in quanto era difficile spostare una macchina, ma nell'era dei portatili questo avviene continuamente. Possiamo avere qualcosa che assegna **dinamicamente** l'indirizzo? Si, questo meccanismo viene chiamato **DHCP (Dynamic Host Configuration Protocol)**.



Questa macchina si identifica in qualche modo nella sottorete, ricevendo dalla stessa un indirizzo, peccato che c'è un'incongruenza di fondo, in quanto questa macchina deve chiedere qualcosa (ovvero l'indirizzo) senza averne ancora uno, per cui abbiamo un'incongruenza logica. Quindi come funziona il meccanismo? Intanto deve esserci un **server dhcp**, ovvero un'entità che assegna l'indirizzo alla macchina. Quando un nuovo client tenta di registrarsi, invia un pacchetto a livello DLL, quindi usando il suo MAC ADDRESS come indirizzo mittente, nella sottorete con indirizzo IP sorgente messo convenzionalmente a 0.0.0.0 e indirizzo destinatario a 255.255.255.255, ovvero l'indirizzo di broadcast; mediante questo pacchetto il client chiede se c'è qualche server dhcp che può fornirgli un indirizzo IP valido. Se il server esiste risponde inviando la sua offerta di indirizzo e una volta che il client accetta il procedimento termina. Cosa accade in particolare?

- Il client invia un pacchetto con indirizzo src 0.0.0.0 e indirizzo mittente 255.255.255.255, ovvero l'indirizzo di broadcast, inviando un **DHCPDISCOVER**
- Un server DHCP raccoglie la richiesta e spedisce, al mac address del client quindi a livello DLL, un **DHCPOFFER**, ovvero un'offerta di indirizzo. Questo pacchetto viene inviato in unicast e può essere ricevuto solo ed esclusivamente da una macchina che si trovi sullo stesso dominio broadcast.
- Il client aspetta un tot di tempo le offerte, dopodiché ne seleziona una, ed invia un **DHCPREQUEST** in broadcast, indicando all'interno di questo pacchetto, con il campo "server identifier", quale server ha selezionato.
- Il server che è stato selezionato conferma l'assegnazione dell'indirizzo con un **DHCPACK**, sempre indirizzato a livello DLL in unicast al client e gli altri server vengono informati che le loro proposte non sono state accettate dal client.

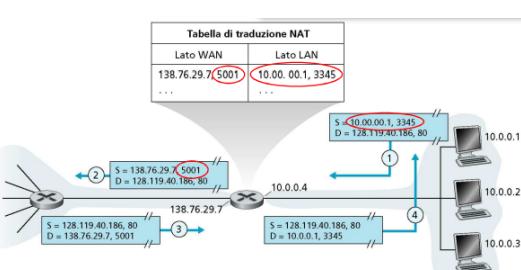
Il server assegna anche un **lifetime**, ovvero l'indirizzo assegnato viene concesso temporaneamente. Entro questo periodo di tempo il client deve confermare che è ancora interessato al servizio, altrimenti non gli verrà rinnovato l'indirizzo. L'operazione di rinnovo è una semplice operazione unicast, se non viene rinnovato l'indirizzo si parte da zero come descritto sopra.

Se sono in molti a chiedere indirizzi il sistema non va in crisi, in quanto il formato del pacchetto di richiesta prevede un **transaction ID** che permette di separare il traffico di diverse macchine.

Se ci sono più server dhcp il client decide in base all'offerta migliore (es. la lifetime più lunga). Un caso famoso in dipartimento è stata la stampante che si spacciava per server DHCP (per cui oltre all'indirizzo inviava anche gateway

e subnetmask errati), in questo caso le macchine windows non funzionavano perché ignoravano il lifetime, per cui sceglievano chi rispondeva prima e in molti casi si affidavano alla stampante; le macchine linux invece funzionavano, in quanto basavano la loro scelta sul lifetime, ed essendo l'offerta del server "ufficiale" quella più conveniente, sceglievano quella.

## NAT (Network Address Translation)



Possiamo fare in modo che un gruppo di macchine si affacci su internet utilizzando un solo indirizzo IP (che è normalmente quello che accade)? Si, è possibile mediante il protocollo **NAT**. Come funziona? Abbiamo una LAN interna chiamata LAN privata, detta in questa maniera perché utilizza una famiglia di indirizzi privati. Se una di queste macchine deve comunicare, nel pacchetto inserisce il suo indirizzo privato come mittente e l'indirizzo pubblico come destinatario. Quando il pacchetto arriva al router, esso sostituisce l'indirizzo mittente privato, con quello pubblico del gruppo a cui appartiene la macchina (vedi figura per capire meglio). L'uscita è banale, ma come fa la risposta a tornare nella macchina di partenza? L'unico modo per riottenere l'indirizzo privato originario della macchina comunicante è quello di salvare, al momento della comunicazione in uscita, l'indirizzo privato e il suo corrispettivo pubblico in una cosiddetta **tavella di traduzione NAT** (come quella in figura). La tabella si suddivide in **Lato WAN (verso l'esterno)** e **Lato LAN (verso l'interno)**. Oltre l'indirizzo viene specificata anche la porta di comunicazione, queste porte sono limitate ad un massimo di 65 mila, per cui non possiamo avere più di 65 mila macchine collegate con quell'indirizzo pubblico, per cui può essere necessario (se vogliamo collegare più macchine) avere più di un indirizzo pubblico da utilizzare per la "traduzione", da qui si capisce che non basta segnare il numero di porta nel lato WAN della tabella, ma l'indirizzo completo. Anche questo è un protocollo molto contestato, perché come DHCP, mischia roba del livello di trasporto con il livello di rete.



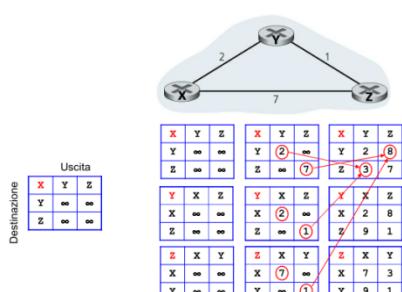
I vantaggi del NAT è che risparmio indirizzi e che le comunicazioni devono essere inizializzate per forza dalle macchine interne, in quanto un pacchetto che parte da fuori non può entrare dentro se la macchina mittente esterna non è presente nella tabella NAT. È uno scudo protettivo, ma è un sistema di sicurezza molto blando.

E se per caso ho un server che non inizializza la connessione, ma la connessione arriva da fuori (es. il server emule)? Ci sono due possibilità, ho do istruzioni al server NAT di inviare i pacchetti diretti ad una porta ben precisa di inviarli ad una macchina ben precisa (quindi scrivo la tabella NAT manualmente) oppure utilizzo il protocollo **UPnP (Universal Plug and Play)**. In questo protocollo la macchina comunica al router che sta mettendo su un servizio che prevede connessioni dall'esterno, che utilizza un determinato numero di porte e che è unico, quindi quando arriva un pacchetto su una di quelle porte deve essere inviata alla macchina che ha messo su il servizio. Ovviamente una porta può essere usata da una macchina alla volta.

## Distance Vectors



Si basa su un algoritmo di ricerca del minimo sui grafici, detto **Bellman-Ford**. La differenza principale è che il DV funziona in parallelo in maniera completamente distribuita, il BF invece è sequenziale. **Funzionamento:**



- Ogni nodo calcola la distanza tra se stesso e gli altri nodi suoi vicini e conserva queste informazioni in una tabella
- Ogni nodo manda la propria tabella ai nodi vicini
- Quando un nodo riceve le tabelle dei propri vicini, calcola il percorso più breve per gli altri nodi e aggiorna la propria tabella con i nuovi dati.

L'algoritmo continua a lavorare finché le tabelle non si stabilizzano, ovvero non vi sono più cambiamenti.

Se per caso uno dei nodi diventa irraggiungibile (es. guasto), accade che gli altri nodi possono impiegare un tempo infinito per aumentare gradatamente la stima della distanza per quel nodo a meno di non adoperare uno scalare.

come soglia, oltre il quale, il nodo viene considerato irraggiungibile; ad esempio supponiamo di avere una rete "lineare" A-B-C-D-E-F e che si interrompa il collegamento con A. Al momento di aggiornare la propria tabella, B noterà che non può più raggiungere A tramite il suo collegamento diretto. Tuttavia C (che è ancora inconsapevole della situazione) sta dichiarando di poter raggiungere A in due passi; B riterrà quindi di poter raggiungere A in tre passi tramite C. Il procedimento potrebbe andare avanti all'infinito.

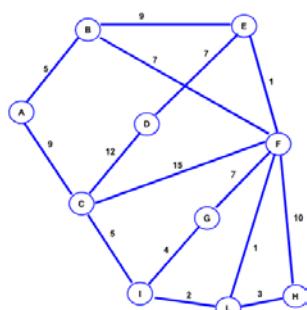
Non funziona con archi negativi, in quanto in caso di numeri negativi il valore minimo sarebbe  $-\infty$ , mentre contemplando solo numeri positivi avremmo 0 come minimo.

L'algoritmo converge molto velocemente.

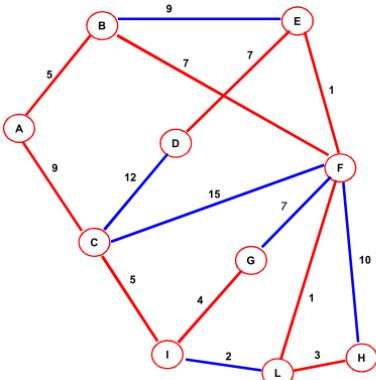
## Link State Routing e Algoritmo di Dijkstra

Sostituisce il distance vectors nel 1979 in quanto il dv non teneva conto delle capacità dei canali, ma solo dei tempi di ritardo di trasmissione. Ogni nodo ottiene una mappa globale della rete mediante particolari pacchetti denominati **Pacchetti Link State**, inviati in broadcast; mediante essi ogni nodo riesce a costruirsi una tabella personale con la situazione di tutti i nodi della rete. 

Una volta ottenuta la mappa globale della rete, i nodi applicano l'algoritmo di Dijkstra per calcolare il cammino minimo per raggiungere ogni nodo della rete ponendosi come radice dell'albero dei cammini minimi. 



L'algoritmo di Dijkstra funziona nella seguente maniera. Ipotiziamo di avere una rete come in figura; prendiamo in analisi il nodo A e scriviamo che  $B=5$  e  $C=9$ , ovvero per raggiungere B sono necessari 5ms, mentre per raggiungere C 9ms. Prendiamo B che è il minimo tra i due e calcoliamo che da A passando per B, abbiamo  $E=5+9=14$  ed  $F=5+7=12$ , inoltre abbiamo ancora che  $C=9$ . Prendiamo il minimo tra questi risultati, ovvero C, e calcoliamo gli altri percorsi:  $D=9+12=21$ ,  $F=9+15=24$  ed  $I=9+5=14$ . Prendiamo F che è il minimo tra questi nuovi risultati e i vecchi e continuiamo il ciclo di ricerca dei percorsi. Alla fine otterremo una cosa uguale a quella presente nella figura successiva con la lista



A	
B = 5	AB
C = 9	AC
F = 12	ABF
L = 13	ABFL
E = 13	ABFE
I = 14	ACI
H = 16	ABFLH
G = 18	ACIG
D = 20	ABFED

dei costi dei cammini e l'albero dei cammini minimi, ponendo A come radice. Ovviamente questo procedimento deve essere ripetuto per ogni nodo della rete.

Lo svantaggio di questi algoritmi centralizzati è che è necessario fare continue fotografie del sistema e passarle a tutti; queste fotografie vengono passate tramite i pacchetti sopra citati, che ovviamente hanno un ritardo di propagazione e in questo tempo in cui si propagano le cose possono cambiare e fare sballare il sistema.

## Confronto Distance Vector e Link State

Entrambi gli algoritmi non funzionano su reti di grosse dimensioni, ma solo su reti limitate.

### Distance Vector

Decentralizzato

I messaggi sono solo per i vicini

Ogni nodo possiede la tabella di tutte le destinazioni

Può avere un problema di conteggio infinito

I messaggi di aggiornamento partono solo se ci sono

### Link State

Globale (centralizzato)

Messaggi inviati in broadcast a tutti

Ogni nodo possiede la tabella dei propri link

A causa del ritardo di sincronizzazione si possono avere delle oscillazioni

I messaggi vengono inviati periodicamente per cui si ha

variazioni sui link

In caso di nodi che fanno i “furbi” (dichiarazione falsa di malfunzionamento di nodi vicini) sistema poco robusto

un maggiore traffico

Visione globale del sistema che rende il sistema molto robusto (tutti sanno di tutti).

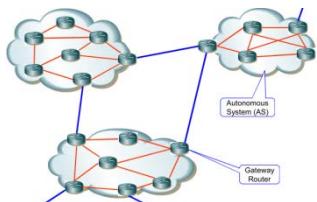
## Internet Routing



E' diviso in **Inter-AS** e **Intra-AS**. Definiamo come AS (*Autonomous Systems*) un gruppo di macchine che sono sotto un'unica gestione amministrativa. L'università di Catania ad esempio è un AS. L'AS potrebbe essere anche molto estesa, ad esempio contenere tutte le università italiane, solo che a quel punto diventerebbe troppo grande e non è detto che sarebbe proprio ottimale, in quanto vengono realizzati due algoritmi particolari: uno specifico per la gestione dell'AS stessa e l'altro per rendere possibile la comunicazione tra AS differenti. I problemi quindi che nascerebbero sarebbero prettamente politici, in quanto in una rete molto grande bisognerebbe mettere d'accordo un'enorme insieme di persone.



Una AS è internamente **fortemente connessa**, quindi non ha una struttura ad albero. Ci sono due serie di protocolli: **IGPs (Interior Gateway Protocols)** e **EGPs (Exterior Gateway Protocols)**; la prima serie riguarda i protocolli interni all'AS (**RIP**, **OSPF**, **HELLO**, **ISIS**) e la seconda invece i protocolli esterni all'AS (**EGP** e **BGP-4**).



In figura abbiamo, con i collegamenti in rosso, un esempio di quella che potrebbe essere una AS. Ci sono una serie di router, chiamati **router di bordo**, che sono quelli che hanno connessioni con altre AS (mediante i collegamenti in blu). Nei router, nei protocolli, viene specificatamente indicato di quale AS fa parte. Non c'è necessità che all'interno di una AS si faccia uso di indirizzi pubblici, in quanto potrebbero essere tutti indirizzi

pubblici. La cosa importante da ricordare è che sono router che fanno parte di un'unica amministrazione.

## RIP (Routing Information Protocol)



RIPv1 e RIPv2 sono i protocolli utilizzati con IPv4; utilizza l'algoritmo di Distance Vector e la metrica utilizzata sono gli **hop**, ovvero il numero di salti effettuati. I protocolli RIPv1 e RIPv2 sono rispettivamente commentati nei documenti **RFC 1058** per v1 e **RFC 1723** e **RFC 2453** per v2. Hanno preso piede sostanzialmente, perché erano di default con BSD.

### RIPv1

Il numero massimo di hop ammessi è 15, che è un numero piccolissimo, per cui l'AS non può essere troppo grande. Ogni nodo ha una **routing Table**, ed ognuna di queste viene scambiata ogni 30 secondi con altre macchine. Ci sono alcune differenze con il Distance Vector: se un percorso per un router **non viene aggiornato** entro 180 secondi, allora la sua distanza è messa a infinito (ovvero a 16, essendo il massimo 15). Dopo questo timer se ne avvia uno di 120 secondi, se entro questi 120 secondi non accade niente allora quel percorso viene completamente cestinato.

RIPv1 utilizza due tipi di messaggi:

- **REQUEST:** per chiedere informazioni ai nodi adiacenti
- **RESPONSE:** per inviare informazioni di routing

Una tabella di routing contiene:

- Indirizzo di destinazione
- Distanza dalla destinazione (in hop)
- Next hop: router adiacente a cui inviare i pacchetti
- Timeout (180)
- Garbage-collection timer (120)

0	7	15	31
Command	Version	Must Be Zero	
Address Family Identifier		Must Be Zero	
IP Address			
Must Be Zero			
Must Be Zero			
Metric			

Il formato del pacchetto è come quello in figura. Una struttura di parola a 32 bit, la dimensione massima è di 512 byte.

- **Command:** 1 = richiesta, 2 = aggiornamento, un tempo erano previsti altri valori.
- **Version:** versione del protocollo
- **Address Family Identifier:** questo campo indica la versione del protocollo IP e deve essere sempre 2 (IPv4)
- **IP Address:** destinazione
- **Metric:** metrica in termini hop count (valore compreso tra 1 e 15)
- **Must be Zero:** sono campi dove deve essere inserito necessariamente ed esclusivamente lo zero.

Per ogni nodo mandiamo il numero di salti e l'indirizzo IP, ma manca la maschera, per cui può usare solo le classi di indirizzi viste in precedenza.

## RIPv2

E' l'evoluzione della versione 1, i principali cambiamenti sono:

- Indirizzamento **CIDR** e **VLSM**. Il CIDR è il sistema delle maschere di sottorete, mentre il VLSM (*Variable Length Subnet Masking*) è una tecnica che permette la suddivisione ricorsiva dello spazio di indirizzi di un'organizzazione, al fine di utilizzarlo in maniera più efficiente. Tramite questa pratica una subnet può essere partizionata in ulteriori subnet, attraverso l'utilizzo di parte dei bit destinati all'host number; si crea così un subnetting a più livelli.
- Autenticazione dei messaggi. Se qualcuno imbroglia spacciandosi per un nodo iniziando a spedire tabelle, nel migliore dei casi incasina solo la rete, nel peggiore dei casi può reindirizzare il traffico verso il nodo che vuole. La soluzione è inserire un'autenticazione nei messaggi.
- Specifica del **next hop**.
- **Split horizon, poison reverse.** Questi problemi vengono in qualche modo risolti, ma non in maniera definitiva.

0	7	15	31
Command	Version	0000	
0xFFFF		Autentication Type	
Autentication			
Address Family Ident.	Route Tag		
IP Address			
Subnet Mask			
Next Hop			
Metric			

Togliendo i campi Subnet Mask e Next hop, il formato è compatibile con RIPv1, ovviamente solo se l'autenticazione è disattivata.

**Route tag:** identificazione di route esterni, ovvero permette di indentificare i percorsi esterni alla nostra AS.

Esistono due modalità di autenticazione: la prima modalità chiamata **light** prevede

la cifratura dei campi mediante una password inviata in chiaro. L'altra modalità è basata su **MD5** e funziona nella seguente maniera: viene preso tutto il pacchetto, aggiunta una password che è scritta in ogni router, calcolato l'MD5 di tutto questo che viene inserito all'interno del campo Autentication. Il router di destinazione prende le informazioni, aggiunge la sua password di identificazione, calcola l'MD5 e verifica che sia uguale a quello presente nel campo Autentication. Sostanzialmente la password non gira, però ogni router aggiunto alla rete deve essere opportunamente configurato.

## RIPng (RIP new generation)

È basato su RIPv2, ma non ne rappresenta un'estensione. E' pensato esclusivamente per IPv6, quindi non può essere utilizzato con IPv4 in quanto gli indirizzi e le tecnologie di routing sono completamente differenti. Stesse caratteristiche di RIPv2 tranne l'autenticazione.

## OSPF (Open Shortest Path First)

Altro protocollo presente nelle AS.

- Pensato per sostituire RIP

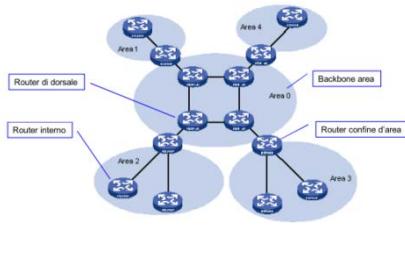
- Si basa sul Link State Routing, per cui la metrica dei salti può essere tranquillamente sostituita.
- Ottimo per reti di dimensioni grandi (non ha il limite dei 15 hop)
- I messaggi hanno autenticazione (poiché i messaggi girano in broadcast, se non fossero autenticati potrebbe essere un grosso problema)

Questo protocollo è formato da tre parti differenti:

- **HELLO:** scoperta e verifica dei vicini. Periodicamente (ogni 30 minuti) un router manda ai suoi link un pacchetto speciale per sapere chi c'è dall'altra parte e calcolarsi il costo dei link.
- **EXCHANGE:** sincronizzazione iniziale del DB. Utilizzato per l'inizializzazione di un nuovo router. Questo funziona a richiesta.
- **FLOODING:** aggiornamento del DB. Funziona a offerta. Periodicamente il router prepara le informazioni sui suoi link e li manda ad i suoi link. Si basa sempre su Dijisktra.

I DB vengono aggiornati mediante particolari messaggi chiamati **LSA (Link State Advertisement)** che vengono emessi quando:

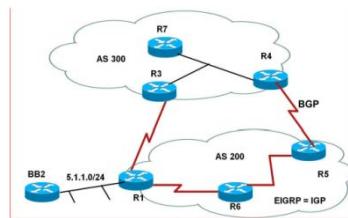
- Quando un router trova un nuovo router adiacente
- Quando un router/link si guasta
- Quando il costo di un link cambia
- Periodicamente, come detto in precedenza, ogni 30 minuti



Una cosa importante è che permette una certa **scalabilità** del sistema, quindi permette una gestione gerarchica di diverse AS, secondo lo schema in figura. Sono necessarie una rete dorsale detta **Backbone** e diverse AS esterne collegate tra di loro mediante questa rete dorsale. Questa suddivisione è ottima per permettere l'utilizzo di metriche differenti nelle diverse aree, a seconda dei contesti. E' sempre un'unica gestione, ma conviene strutturarla in maniera gerarchica.

## BGP-4 (Border Gateway Protocol v.4)

Le AS comunicano mediante i Border Gateway, il protocollo più importante che permette la comunicazione tra queste AS è il protocollo BGP, che attualmente è arrivato alla versione 4.



Data una situazione come in figura, dove due border gateway sono connessi alla stessa AS (una cosa del genere è concepita per motivi di ridondanza e/o per divider il traffico in maniera equa), come si decide da quale parte il pacchetto deve viaggiare? L'amministratore può decidere tenendo in considerazione vari fattori; una politica è quella dell'**Hot Potato**, dove se un AS non è interessato ad un pacchetto, deve fare in modo da spedirlo fuori il prima possibile, per cui sceglie il percorso più scarico in modo da liberarsene il prima possibile. Oppure dividere il traffico in maniera equa o scegliere la route con il costo più basso. Esistono molte altre politiche, infatti BGP è un protocollo molto complicato.

Perché non usare OSPF anche per le exterior AS? Per questioni prettamente politiche ed economiche. Immaginiamo il traffico tra USA e Giappone; lo facciamo passare tramite la Cina? No, magari usiamo un percorso più lungo ma che sia il più politicamente corretto. BGP è stato ideato appositamente in modo che gli amministratori possano scegliere attraverso quali router far passare il traffico. Tecnicamente parliamo di:

- **AS Border Router (ASBR):** ovvero i router di bordo connessi ad altri sistemi autonomi.
- **BGP speaker:** router che supporta il protocollo BGP, ma che non corrisponde necessariamente con un Border Router.

- **BGP Neighbors:** coppia di BGP speaker (connessi mediante TCP semi permanente, punto-punto) che si scambiano informazioni di instradamento inter-AS sono:
  - **Interni:** se appartengono alla stessa AS
  - **Esterne:** se appartengono ad AS differenti

Il router di casa non ha BGP, perché il provider fa tutto praticamente e quindi riceve tutto dall'esterno. E' possibile configurarlo come RIP, solo se dietro abbiamo un altro sistema di routing, ma a quel punto se dietro abbiamo una rete abbastanza complessa l'operazione non è giustificata.

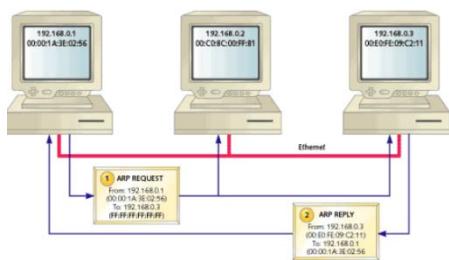
## Indirizzi LAN e IP

Una macchina che vuole mandare un pacchetto verso l'esterno deve disporre del suo indirizzo IP e dell'indirizzo di destinazione e la prima operazione che fa è applicare la subnet mask ai due indirizzi e poi confrontare i due risultati bit a bit. Se la macchina di destinazione è della mia stessa sotto rete, allora applicando la subnet mask (come visto in precedenza) otteniamo la stessa cosa, se non è così allora è necessario fare routing. Come mandiamo il pacchetto al router? Mediante il collegamento DLL, ovvero all'interno di un pacchetto DLL dove specifichiamo l'indirizzo MAC di destinazione. Più in generale, se scopriamo che la macchina di destinazione è locale, dobbiamo scoprire il MAC address della macchina di destinazione; se invece la macchina è remota dobbiamo scoprire il MAC address del router. A livello DLL il router si vede arrivare il pacchetto e una volta ricevuto lo apre e lo manda fuori. Se non arriva a livello DLL il router non fa nulla. Questo implica che un host deve conoscere la net mask della propria sottorete e l'indirizzo MAC del router.



## ARP (Address Request Protocol), RARP (Reverse ARP) e BOOTP (Bootstrap Protocol)

Una macchina per comunicare con una sua controparte ha necessità di conoscerne l'indirizzo sia a livello IP che a livello LAN (DLL), in quanto sul lato software viene creato un pacchetto con l'indirizzo IP, ma questo pacchetto alla fine deve essere inserito su un mezzo fisico, che ragiona in termini di frame e che utilizza l'indirizzo MAC della macchina di destinazione. La macchina se non conosce questo indirizzo lo può chiedere mediante il protocollo **MAC**.



Come funziona? La macchina prepara un pacchetto secondo il protocollo ARP inserendo il suo indirizzo IP e LAN ed inserendo l'indirizzo IP della macchina destinazione e come MAC l'indirizzo broadcast (FF:FF:FF:FF:FF:FF), così che il pacchetto venga propagato per tutta la subnet. Le macchine che lo ricevono controllano l'indirizzo IP e verificano che è il loro indirizzo; se non lo è allora non fanno nulla, altrimenti rispondono con un altro pacchetto ARP inserendo il proprio MAC e spedendolo direttamente alla macchina che aveva avviato la

procedura di ARP request, che ricevendo la risposta si salva l'informazione dandole anche un tempo di vita. Non dovrebbe esserci risposta dalle altre macchine, ma poiché non è un entità certificata che risponde, potrebbe esserci un tentativo d'attacco nascosto dietro la risposta, in quanto la macchina rispondente potrebbe spacciarsi per chi non è, in quanto il tutto si basa su un'auto certificazione. Se il pacchetto deve andare fuori, come detto in precedenza, basta semplicemente conoscere l'indirizzo IP del router.

**RARP** invece serve a ricavare un indirizzo IP dato un MAC address, ovvero consente ad un host di conoscere il proprio indirizzo IP all'accensione chiedendolo, in modalità broadcast agli altri host connessi alla rete. In genere la richiesta arriva ad un server RARP che contiene l'indirizzo di risposta nei propri file di configurazione. C'è una certa somiglianza con il DHCP, però a differenza di quest'ultimo, il protocollo RARP fornisce sempre lo stesso indirizzo dato il MAC Address.

L'unico motivo per cui si utilizza RARP è in unione con il protocollo **BOOTP**. Inventato per risolvere il problema delle stazioni DiskLess (senza Hard Disk), problema che oramai non esiste più. Queste macchine non avevano Sistema Operativo, per cui non potevano fare un Boot completo, ma bensì solo la fase del controllo della memoria per cui un minimo di protocollo di Bootstrap è possibile posizionarlo nella memoria EPROM messa nella scheda di rete. Il pc esegue il Post e poi passa il controllo alla scheda di rete, che ha un minimo di Kernel (una sorta di sistema operativo di rete) per poter accedere alla rete e poter chiedere a qualcuno tutto il sistema operativo, caricarlo nella RAM e poi

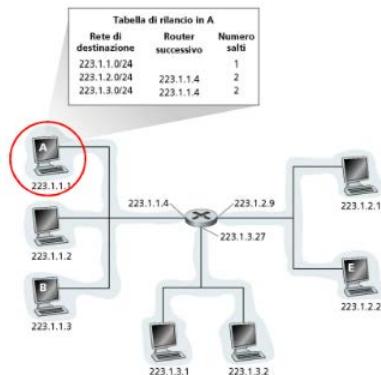
fare tutto il BootStrap. Serve un Kernel minimo per configurare la scheda di rete, configurare il minimo di hardware e quelle poche regole per comunicare con il server. Il protocollo RARP serve per chiedere al server, con la macchina appena "sveglia" qual è il proprio indirizzo IP. Oggi un protocollo di questo tipo può avere senso in un ambiente particolare: in un'azienda possono esserci centinaia di macchine da configurare, che fare a manina potrebbe essere un po' lungo. L'alternativa di configurarle manualmente, sarebbe creare le immagini e mediante BOOTP caricarle da remoto. La stessa filosofia si può utilizzare per effettuare gli aggiornamenti di sistema, senza tenere le macchine bloccate.

## Tabelle di routing per host

Una macchina possiede una tabellina di routing dove sono presenti almeno tre righe minime che contengono:

- **Indirizzo di Loopback**
- **Indirizzi che non devono passare dal router**
- **Indirizzi che devono passare dal router**

Normalmente una macchina deve conoscere la LAN a cui appartiene, se stessa e il mondo esterno (dati fondamentali per navigare). Queste tre righe quindi sono quelle che sono sempre presenti in una tabella di routing di un macchina, scritta in maniera praticamente automatica conoscendo il proprio indirizzo, l'indirizzo del router e la net mask.

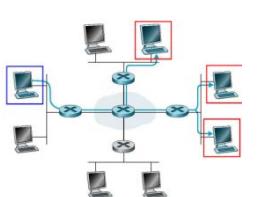


Un esempio lo vediamo nella figura, dove è rappresentata la tabella della macchina evidenziata. Nella prima riga la rete di destinazione è la 223.1.1.0/24, vediamo che il router successivo è vuoto in quanto si tratta della stessa sottorete, per cui il numero di salti è 1. Nelle righe successive invece bisogna raggiungere altre sottoreti, per cui è presente anche l'indirizzo del router successivo con i salti da effettuare.

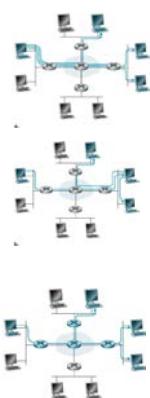
Il mondo esterno come si rappresenta? Dobbiamo far sì che il risultato dell'AND bit a bit della net mask e dell'indirizzo di destinazione sia diverso da 1 (in modo da essere sicuri che appartengano a due sottoreti differenti). Per cui è necessario inserire un indirizzo destinazione IP 0.0.0.0/0, che è proprio l'indirizzo che viene usato per uscire su internet e si trova nell'ultima riga della tabella di routing dell'host. Queste informazioni fortunatamente vengono scritte in automatico.

## Multicast

IPv4 ha riservato una bella fetta di indirizzi per il multicast; non è mai stato usato seriamente in quanto il multicast dovrebbe essere gestito dai router e la maggior parte dei router esistenti ignora questo metodo.



Il **broadcast** vuol dire a **tutti**, mentre **multicast** solamente ad **alcuni**. Non è stato iniziato inizialmente perché non c'era necessità, in seguito ha preso piede con l'avvento dei servizi di video on demand. Ad esempio la macchina blu può decidere di inviare il servizio solo alle macchine rosse, in quanto hanno pagato l'abbonamento. Come si può implementare una cosa del genere? Ci sono tre possibilità:



a) Una connessione per ogni destinazione, solo che questo comporta uno spreco enorme di banda. Ad esempio in figura abbiamo 3 connessioni identiche in un tratto, 2 in un altro e infine una soltanto. I canali di comunicazione doppioni sono praticamente uguali, in quanto dai destinatari non c'è risposta.

b) Oppure si può sviluppare una sorta di catena, dove il server invia ad un client che a sua volta lo dirama verso gli altri clienti, e così via. Questo sistema porta in dote un ritardo, che su sistemi multicast seri può essere un grosso problema. Il vantaggio è che il traffico viene spostato verso la destinazione, non è detto che la macchina che splitta il segnale sia un utente finale, potrebbe essere anche una macchina di buffer. Anche in questo caso abbiamo

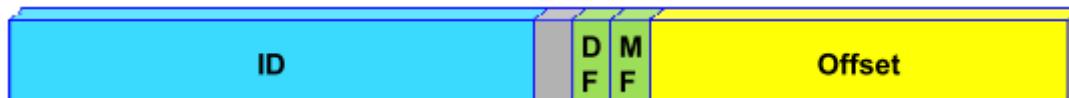
uno spreco di banda.

- c) La soluzione ottimale è che i router di transito supportino il multicast, in modo da prendere loro il segnale e diramarlo per la rete.

Come possiamo implementare un ragionamento del genere? Dato un insieme di macchine collegate come se fossero un grafo dobbiamo trovare un albero di copertura, dove le foglie sono i client finali che devono ricevere il segnale. In particolare è uno dei tanti alberi di copertura minima, partendo dalla sorgente. Nell'esempio A può conoscere tranquillamente il proprio albero di copertura minimo e quindi sa dove instradare il pacchetto, ma C, a meno che non si calcoli l'albero di A, non può conoscerlo. In una rete di grandi dimensioni, permettere ad un nodo di calcolarsi il percorso minimo di tutti gli altri nodi sarebbe una cosa impensabile. Serve quindi un algoritmo che riduca queste informazioni al minimo indispensabile; quindi iniziamo facendo in modo che i nodi conoscano **il proprio albero di copertura minima**. Se basandoci su quest'ultima premessa riusciamo a tirare fuori qualcosa che ci avvicina all'ottimo, abbiamo risolto il problema. Questo algoritmo esiste e si chiama **RPF (Reverse Path Forwarding)**, l'idea è la seguente: Il nodo C riceve un pacchetto da A, sicuramente non lo rimanda indietro e può scegliere 3 strade. Il nodo E a sua volta riceve 3 copie e se le facesse viaggiare tutte e tre avremmo il flooding (con tutte le problematiche legate al caso); E non può conoscere l'albero di copertura di A, ma può conoscere **il proprio albero di copertura minima che lo porta ad A**. Se il collegamento non è un grafo orientato ed i pesi sono **simmetrici** (dati due nodi A e B il peso del link è lo stesso sia in andata che ritorno), allora il minimo percorso calcolato da E tra se stesso ed A, sarà lo stesso di quello calcolato da A fino ad E, per cui se E si vede arrivare un pacchetto da quel cammino, allora sa che ha viaggiato per l'albero di copertura minima. Questo pacchetto lo propaga dovunque, tranne dal ramo da cui l'ha ricevuto; cosa succede? Può inoltrarlo su di un ramo inutile oppure su rami che fanno parte dell'albero di copertura minima tra A ed il nodo che riceve. Il nodo che riceve, se vede che il pacchetto non ha seguito il cammino minimo di copertura con sorgente A, ignora il pacchetto. Seguendo queste regole, non c'è bisogno di inserire un contatore di salti per eliminare le copie, in quanto, se un pacchetto non segue l'albero di copertura minima, fa un salto e viene eliminato. I nodi necessitano quindi di conoscere solo il cammino minimo tra se stessi e la sorgente. C'è un eccesso di comunicazione, ma è solo di un salto, in quanto poi viene buttato.

## Frammentazione

Il pacchetto IP è grande al massimo 64Kbyte, una frame Ethernet è grande 1500byte come payload, per cui un pacchetto grande IP non può stare dentro una frame Ethernet. È necessaria una frammentazione, che non è una cosa complicata, è necessario però un meccanismo di numerazione. Inoltre non è detto che un pacchetto già frammentato possa avere vita facile, in quanto potrebbe incontrare un mezzo trasmissivo che permette una capienza inferiore e quindi debba essere frammentato ulteriormente. IPv4 prevede nell'header tre campi che permettono la frammentazione.



**DF** e **MF** stanno per **don't fragment** e **more fragment** e come abbiamo visto nel capitolo dedicato alla frame IPv4 servono ad indicare se quella frame può essere suddivisa in più frammenti e se fa parte di una serie di frammenti. Vediamo un esempio di funzionamento del sistema:



In cima abbiamo il pacchetto completo con il bit DF settato a 0, per cui abbiamo il permesso di frammentare; L'ID è l'identificativo del pacchetto è sarà lo stesso per tutti i frammenti (in modo da poterlo ricomporre a destinazione anche se arriva intervallato con frammenti di altri pacchetti). Il bit MF è settato a 0, in quanto si tratta del pacchetto completo, così come l'Offset. Questa frame arriva al router che la blocca perché troppo grande e deve frammentarla. La divide quindi nei pacchetti successivi: il primo pacchetto ha il bit MF settato a 1, per indicare che fa parte di una serie di pacchetti frammentati e che oltre a lui ce ne sono altri e l'offset è

impostato a 0 in quanto si tratta del primo frammento. Il secondo pacchetto ha sempre il bit MF settato a 1 e l'offset settato a 4, in quanto al suo interno presenta i dati di quell'ID pacchetto a partire dal 4bit in poi. L'ultimo pacchetto ha il bit MF settato a 0, in quanto dopo di lui non ci sono più frammenti e l'offset settato a 10. Grazie a queste informazioni anche se arrivano in ordine sparso, i pacchetti possono essere ricostruiti tranquillamente, basandosi sull'Offset.

Un pacchetto che ha MF a 0 e offset a 0 è un unico pacchetto e non fa parte di nessun pacchetto frammentato. Invece un pacchetto che ha MF a 0 e offset diverso da 0 è l'ultimo frammento di una serie. Se arriva prima l'ultimo frammento allora possiamo conoscere subito quanto saranno grandi i dati che riceveremo, se arriva invece il primo non possiamo sapere nulla, così come se arriva un pezzo intermedio.

## ICMP (Internet Control Message Protocol)

E' un sotto protocollo di IP che si occupa di trasmettere informazioni riguardanti malfunzionamenti, informazioni di controllo o messaggi tra i vari componenti di una rete di calcolatori. Un esempio ad esempio è **ping**, che ci dice se a livello IP l'applicazione funziona oppure no, ovvero se la macchina ha l'accesso alla rete. Per ragioni di sicurezza a volte il demon che risponde a ping è disabilitato, in quanto la prima cosa che fanno gli attaccanti è controllare se la macchina c'è, inoltre è qualcosa di così banale che non lascia traccia neanche nei file di log. La macchina che manda questo pacchetto mette il proprio indirizzo IP e vogliamo scoprire che percorso fa il pacchetto, però sappiamo che il router appena riceve il pacchetto lo trasmette semplicemente senza fare nient'altro; per cui per sapere che percorso

```
Traccia instradamento verso www.google.com [173.194.35.179]
su un massimo di 30 punti di passaggio:
 1  491 ms  508 ms  529 ms  172.16.4.1
 2  354 ms  498 ms  539 ms  172.16.4.0
 3  1546 ms  528 ms  498 ms  151.6.116.68
 4  549 ms  538 ms  537 ms  80.85.162.155-PMC-B02-Ge10-1.wind.it [151.6.5.37]
 5  538 ms  1508 ms  529 ms  151.6.0.58
 6  1664 ms  558 ms  558 ms  151.6.0.26
 7  205 ms  248 ms  358 ms  209.85.249.54
 8  1363 ms  538 ms  488 ms  216.239.48.146
 9  585 ms  608 ms  1408 ms  209.85.250.39
10  605 ms  588 ms  578 ms  nuc03s02-in-f19.1e100.net [173.194.35.179]
```

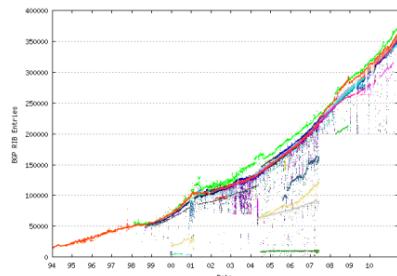
fa è necessario che il router ci comunichi che è stato lui a far uscire il pacchetto. Questo meccanismo si può implementare mediante il TTL (Time to Live), vediamo come: prendiamo una destinazione (es. www.google.com) prendiamo il pacchetto e mettiamo l'indirizzo della destinazione e poniamo TTL a 1, il

primo router che riceve il pacchetto pone il TTL a 0 e ci manda messaggio indietro mediante ICMP con TTL Expired e ci comunica chi è e ce lo scriviamo. Poi mettiamo TTL 2 e ci risponde il secondo router sempre con un TTL Expired ed il suo nome, scriviamo questa info e continuiamo; il procedimento continua fino a destinazione. Nessuno però ci assicura che questa sequenza di pacchetti segua sempre la stessa strada, per cui l'unico caso dove il funzionamento certo è la rete a circuito virtuale, ma su una rete datagram normale no. Infatti solitamente si fanno 3 tentativi per avere una minima certezza. Il comando **traceroute** fa proprio questo, ci comunica l'indirizzo del router, il nome (se riesce a convertirlo) ed il tempo. Se non abbiamo risposta riceviamo un \*. Traceroute ci fa vedere, in caso di malfunzionamento, dove si ferma il pacchetto.

## IPv6

La necessità di un nuovo spazio di indirizzamento è dovuta al fatto ad alcuni limiti di IPv4, ovvero:

- Esaurimento dello spazio indirizzi
- Scalabilità del routing
- Mancanza di possibilità di aggiungere nuovi servizi



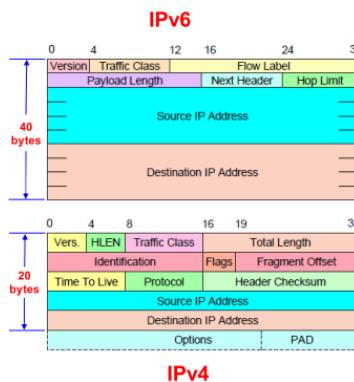
Come si vede dal grafico, le tabelle BGP con il passare degli anni stanno diventando troppo grandi (portando ai problemi di routing di IPv4). Perché? Se avessero avuto più lungimiranza avrebbero fatto in modo che località vicine (es. Francia e Italia) avessero avuto indirizzi molto simili tra loro (es. l'ultimo bit differente). Invece si hanno indirizzi totalmente differenti anche a distanza di 10 metri, con tabelle di routing enormi, in quanto il routing va fatto su quasi tutto l'indirizzo.

I nuovi servizi richiesti sono:

- Sicurezza, che IPv4 non ne ha

- Autoconfigurazione. In IPv4 abbiamo DHCP e NAT, ma vanno a violare in maniera brutale (come abbiamo già visto) lo schema a livelli.
- Gestione della Qualità del Servizio (QoS). Le televisioni multimediali, ad esempio, non devono avere brutalmente priorità, ma risorse ben allocate predefinite. Hanno requisiti ben particolari.
- Indirizzamento Multicast. In IPv4 abbiamo gli indirizzi riservati, ma i router non lo supportano.
- Indirizzamento di host mobili. IPv4 è molto legato alla fisicalità della zona; dare un indirizzo IP al cellulare sarebbe una buona cosa, così come diamo un numero di telefono, ma con IPv4 invece si deve collegare e riceve un indirizzo IP di quel posto. Con IPv6 invece alcuni indirizzi sono riservati agli host mobili ed il problema del routing viene risolto da un'altra parte.

**IPv5** era stato pensato per risolvere i problemi legati al routing per il multicast, nel frattempo stavano realizzando **IPng**. Sono state messe un po' di cose assieme creando **IPv6**.



Il formato del pacchetto datagram di IPv6 è quello in figura, messo a confronto con quello di IPv4. Facciamo una piccola riflessione sul campo Source IP e Destinazion IP: per evitare di rimanere a secco di indirizzi si è deciso di utilizzare uno spazio di circa 16byte, ovvero  $2^{128} \approx 3 * 10^{38}$  indirizzi, ovvero un numero spropositato di indirizzi. Però non è assolutamente un problema.

- **Traffic Class:** serve per gestire la qualità del servizio, specificando la classe di servizio legata a quel pacchetto.
- **Flow Label:** è il campo tipico per i sistemi a circuito virtuale. Se voglio gestire applicazioni multimediali sarebbe ottimale un circuito virtuale che non utilizza l'indirizzamento mediante indirizzo sorgente e destinazione, ma mediante un identificativo di circuito, che è proprio questo campo qui.
- **Payload Length:** il nome già dice tutto
- **Next Header:** in IPv6 hanno pensato di ottimizzare l'istradamento realizzando degli header aggiuntivi, che seguono il blocco degli indirizzi, specificandone la tipologia (ad esempio potrebbe essere presente un header TCP) oppure altro.
- **Hop Limit:** il campo è sempre di 8 bit, in quanto se si effettuano più di 255 hop, sicuramente c'è qualcosa che non va nel routing. Il grafo non va aumentato come diametro, ma come connessioni; ovvero aumento il numero di nodi, ma parallelamente aumento anche le connessioni. Altrimenti avremmo sempre lo stesso un aumento esponenziale dei tempi di routing.

Facendo un confronto con IPv4, sono stati rimossi:

- ID, Flags e Offset che servivano per la frammentazione
- ToS, HLEN. Ora l'header è di dimensione fissa, mentre in IPv4 era di dimensione variabile.
- Header Checksum. Non lo guardava nessuno.

Sono cambiati invece:

- Total lenght → payload
- Protocol → next header
- TTL → hop limit

Sono stati aggiunti:

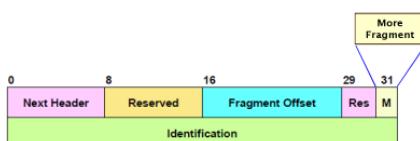
- Traffic class
- Flow label



L'header di dimensione fissa da la possibilità ai produttori di router di tarare l'hardware per controllare quei 40 byte e basta; mentre con IPv4 non era possibile in quanto avevamo 20byte minimo, che poteva aumentare se era presente il campo Options. Un Hardware specifico permette un più veloce istradamento e ovviamente un abbattimento dei costi di produzione.

Come detto in precedenza, la frame IPv6 permette la presenza di un Header Opzionale, dopo l'header classico. Questo meccanismo ha un funzionamento come quello in figura, ovvero alla fine dell'Header classico è presente un campo che specifica di che tipo sarà l'Header che seguirà (nell'esempio è TCP). E' possibile avere più header opzionali, utilizzando uno schema a catena, ovvero: alla fine dell'header classico verranno inserite le informazioni sull'header opzionale 1, che a sua volta conterrà, alla fine, le info che indicano un header opzionale 2...e così via. Come se fosse una lista linkata.

La frammentazione in IPv6 non è contemplata, il minimo MTU è di 1280 byte, ma sono raccomandati di raggiungere



i 1500, ovvero la dimensione della frame Ethernet. Se è proprio necessario, allora il mittente può frammentare, ma non i router di transito, mediante uno schema ereditato da IPv4 (visibile in figura) chiamato **Fragment Header**. Inoltre il pacchetto IPv6 può essere frammentato solo dopo il Fragment Header, tutto quello che viene prima non deve essere assolutamente toccato.

La rappresentazione dell'indirizzo in IPv6 cambia radicalmente, abbiamo una cosa del tipo:

**8000:0000:0000:0123:4567:89AB:CDEF** che si può scrivere anche come **8000::123:4567:89AB:CDEF**

Ovvero una serie di 4 caratteri esadecimali, intervallati da : tra loro. Considerato che c'è una maniera di indirizzi, c'è la possibilità che molti siano pieni di zero, per cui si può utilizzare una notazione ristretta sostituendo la serie continua di zero con caratteri :: come visto nell'esempio superiore. Il numero di zeri mancanti è uguale al numero di bit mancanti per arrivare a 16byte, ovviamente in uno stesso indirizzo non possono esserci più contrazioni in quanto non sapremo più dove andare a piazzare gli zeri mancanti. Poiché il nuovo spazio di indirizzamento è enorme, hanno pensato di prendere una piccola parte di questo spazio e di dedicarlo agli indirizzi IPv4, in modo da facilitare la transizione da IPv4 a IPv6. Un indirizzo IPv4 mappato su IPv6 è del tipo ::151.97.1.1, per cui i primi 12 byte sono tutti a zero.

La suddivisione degli indirizzi è stata fatta in maniera un po' più diligente. Sono state riservate già alcune classi di indirizzo, però il 72% dello spazio è ancora inutilizzato:

<b>0000 0000</b>	Riservati (includono IPv4)
<b>0000 001</b>	Indirizzi per il sistema OSI (se mai entrerà in funzione)
<b>0000 010</b>	Indirizzi Novell IPX
<b>010</b>	Indirizzi per i service providers
<b>100</b>	Indirizzi geografici. E' stato fatto quello che mancava in IPv4, ovvero assegnare alle località geografiche una serie di indirizzi simili in modo da semplificare il routing
<b>1111 1111</b>	Indirizzi Multicast
<b>1111 1110 10</b>	Indirizzi locali di canale. Quando ho un collegamento punto-punto, le due macchine devono comunicare l'una con l'altra e basta; anche se la cosa è abbastanza banale è necessario assegnare un indirizzo IP (sprecato) alle due estremità. Per cui utilizziamo il MAC Address per generare un indirizzo IP univoco che mi serve ad uno scopo limitato.
<b>1111 1110 11</b>	Indirizzi locali di sito. Sono equivalenti agli indirizzi privati di IPv4, però di numero molto più ampio. Sono sempre indirizzi che non permettono la navigazione su internet.

DHCPv6 non è la stessa cosa di DHCPv4, sono due cose completamente differenti. DHCPv6 non ha il problema dei conflitti, in quanto ogni richiesta viene già marchiata con il suo possessore, inoltre vista la grande mole di indirizzi presenti il significato di questo sistema quasi si perde.

Le tipologie di indirizzamento di IPv6 sono:

- **Unicast:** da una macchina ad un'altra macchina, che è quello classico di IPv4.
- **Multicast:** che in IPv4 era previsto, ma mai utilizzato.
- **Anycast:** l'idea è, ci sono tante macchine più o meno equivalenti ovvero che forniscono quasi le stesse informazioni (ad es. Google ha diverse macchine che gestiscono il suo sistema), per cui io mando il pacchetto

al servizio e la risposta mi arriva da una delle tante macchine (magari quella più scarica o la più vicina) che gestiscono quel servizio).

IPv4 e IPv6 sono incompatibili, per cui i rispettivi pacchetti non possono viaggiare sulla rete dell'altro, ci sono due modalità pensate per effettuare la migrazione:

- **Tunneling:** se non vogliamo fare lo switch-off, allora è necessario creare reti con protocolli diversi; se sono dorsali di comunicazione in IPv6 allora il problema non sussiste, perché gli utenti finali sono ancora IPv4. Se cambiamo il mondo esterno e lo mettiamo in IPv6, isoliamo l'utente finale che usa ancora il router in IPv4, per cui dobbiamo garantire il funzionamento finché non verrà cambiato il modem. La soluzione può essere: l'utente parte con IPv4, il modem si collega ad una macchina (es. Alice) IPv4 che esce poi in IPv6, viaggia IPv6 fino a vicino alla destinazione e poi nei pressi della destinazione viene decapsulato e ritrasformato in IPv4. Lo switch-off generale potrà avvenire, solo quando tutte le destinazioni saranno IPv6.
- **Dual-Stack:** si ha un modem che è capace di generare pacchetti sia IPv4 che IPv6, in modo da scegliere quale dei due sia necessario al momento della comunicazione. Questo sarebbe il metodo più adatto per effettuare una migrazione.

## Firewall

Esistono due tipi di firewall, quello meno serio è fatto via software e quello serio è fatto via hardware. Quello software è meno serio in quanto, se devo proteggere una macchina il modo migliore è quello di non fare arrivare proprio i pacchetti che non sono necessari ed il firewall software il massimo che fa è controllare il tipo di pacchetto e buttarlo via se rientra nella blacklist; il problema è che in questo lasso di tempo, la macchina può venire bucata. Diverso è il discorso se mettiamo di mezzo una macchina che fisicamente "filtra" i pacchetti prima di farli arrivare alla macchina desiderata.

Un Firewall hardware è composto da diverse parti hardware: un router verso la zona interna da proteggere, un **Application Gateway** (Gateway di livello Applicativo) e un router sempre verso la zona interna, che comunica però con il mondo esterno. Questo sistema funziona così: entra un pacchetto di posta, il router verso l'esterno lo manda all'application Gateway il quale ne verifica la provenienza (se fa parte della blacklist o meno) e se è formattato correttamente per la sua natura (ad esempio potrebbe essere un pacchetto, che viene da una fonte autorevole, ma essere maligno e spacciarsi per un pacchetto di posta); superati questi controlli, il pacchetto viene inviato al router verso l'interno, il quale controlla se effettivamente la macchina indicata come destinatario è realmente il vero destinatario (ad es. il nostro pacchetto di posta può essere mandato ad una macchina che non è un server di posta, in tal caso c'è qualcosa di sospetto), se tutti i controlli vengono superati il pacchetto passa senza problemi.

Un attaccante per riuscire ad entrare deve eludere un router esterno, che non possiede sistema operativo, ma principalmente firmware, per cui molto più difficile da bucare, in quanto, secondo la legge per cui i bug sono direttamente proporzionali alle linee di codice, essendo software molto piccolo dovrebbe contenerne pochi. La macchina di mezzo invece è più vulnerabile, in quanto possiede già un sistema operativo dovendo effettuare operazioni più complesse. Per l'altro router valgono le stesse considerazioni di sopra.

Soltanamente nei firewall professionali, l'accesso alle configurazioni è permesso solo mediante una porta (parallela, o altro) posta dietro la macchina, questo perché prevedere la configurazione in rete della macchina già prevede un difetto, in quanto il firewall può essere bucato. Invece l'essere fisicamente presente accanto al firewall per poterlo configurare aggiunge un livello di sicurezza in più.

I firewall al livello software non hanno niente di tutto questo. Il firewall non riesce a reagire agli attacchi **dos** o **ddos**, in quel caso serve un hardware particolare.



## Parte 5 – Livello di Trasporto

E' il primo livello end to end della pila protocollare, cioè i due host comunicano direttamente, tramite un unico canale; in parole più semplici fornisce una comunicazione logica tra gli host, saltando tutte le problematiche della connessione di rete. E' il livello che separa la zona protocollare bassa, dalla zona protocollare alta. In questo livello introduciamo le porte.



### Indirizzamento nel TL (mux - demux)

Se io voglio comunicare con un'altra macchina, il punto di contatto con il livello di Rete è il **NSAP (Network Service**

**Access Protocol**), mentre il punto di contatto con il livello Applicativo è **TSAP (Transport Service Access Protocol)**. Questo TSPA ha necessità di un indirizzo che è un numero di 2byte (0 – 64k) dove alcuni sono riservati; io ho due processi da mettere in comunicazione che sono identificati dal rispettivo PID e purtroppo non posso usare questo PID per mettere in comunicazione i due processi, in quanto ogni macchina nel sistema operativo gestisce i PID per i fatti propri. Entrano in gioco allora i **numeri di porta** che sono quegli indirizzi che permettono al processo di affacciarsi sul mondo

internet. Per cui la macchina che deve comunicare, prepara un pacchetto particolare (socket) selezionando il numero di porta del processo che deve ricevere, come fa a selezionare questo numero di porta? La cosa più semplice che si può fare è creare una tabella con tutta una lista di servizi a cui vengono assegnati i numeri di porta; questa tabella può essere creata e messa a disposizione comunicandola, oppure crearne una standard per tutti (che è la soluzione che è stata adottata). Nella prima comunicazione la porta sorgente è a scelta da parte del servizio in uscita, la porta di destinazione invece è obbligata a seconda del tipo di servizio richiesto. In generale nella risposta, la porta sorgente diventa la porta di destinazione e viceversa.

Posso quindi selezionare un servizio e assegnargli un numero di porta, però c'è il rischio che questo processo mi blocchi il sistema finché non viene portato a termine. Una soluzione sarebbe tenere tutti i processi attivi, in modo da tenerli pronti a rispondere, però una soluzione del genere mi occuperebbe troppa memoria; la soluzione pensata è allora creare una sorta di centralino, che rimane sempre in ascolto e appena riceve una chiamata, chiede per quale processo è il pacchetto in entrata. Chiama in seguito il processo corretto e lo mette in comunicazione con quello entrante. Il vantaggio è che il centralino è molto piccolo, per cui può rimanere in attesa tranquillamente ed ha pochi bug. Su linux si chiama **xinetd**.

Il socket rappresenta un canale di comunicazione tra le due entità ed è caratterizzato da 5 parametri. Due sono gli indirizzi IP, due sono gli indirizzi di porta e l'ultimo è il protocollo che può essere **TCP o UDP**. Questo mi permette di avere 65k porte sia per TCP che per UDP, infatti a volte se guardiamo il sistema il servizio è disponibile sia per TCP che per UDP, perché il sistema riesce a differenziare i due protocolli.

### UDP (User Datagram Protocol)

UDP è la minima aggiunta che bisogna fare ad IP per fare viaggiare un pacchetto utile. Serve un numero di porta (la sorgente e la destinazione), quanto è grande il datagramma e un checksum; questo è il pacchetto più piccolo

0	15 – 16	31
Source Port Number(16 bits)	Destination Port Number(16 bits)	possible, in quanto l'informazione principale sono il numero delle porte.
Length(UDP Header + Data)16 bits	UDP Checksum(16 bits)	Non mi da alcuna garanzia: non c'è connessione, non c'è stato e non c'è connessione di flusso, ovvero che un trasmettitore UDP inizia a spedire pacchetti e non sa che fine fanno (arrivano o non arrivano). La porta mittente ci serve solo se necessitiamo di una risposta, altrimenti non serve
Application Data (Message)		icon

a niente.

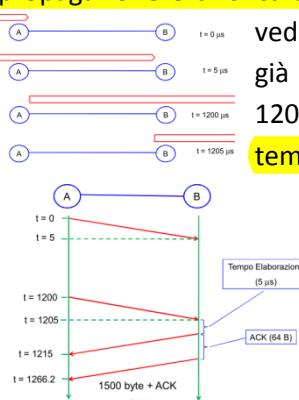
### Reliable data transfer

La situazione attuale è che lo stato di rete fornisce un canale non affidabile, ovvero può far arrivare frame errate e perdere le frame. A livello di trasporto vogliamo un canale affidabile e l'unico modo è mediante la ritrasmissione, che deve essere effettuata a regola d'arte, altrimenti rischiamo di aggiungere pezzi che non ci sono o di far arrivare doppioni. Facciamo una serie di passi per costruire un canale di trasferimento affidabile:

- **RdT 1.0** - Supponiamo di avere un canale affidabile e che il ricevente è in grado di andare alla velocità della luce e di smaltire subito il lavoro. Chi invia prepara il packet, chiama una funzione send e lo invia; il ricevente deve rimanere in attesa di un evento ovvero ricezione di un pacchetto, lo riceve, estrae i dati e lo manda a livello superiore. Questo è il funzionamento del nostro canale ideale.
- **RdT 2.0** - inseriamo un problema reale, ovvero il canale può inserire un errore nel pacchetto. Una soluzione può essere che il receiver invia un **ACK** se il pacchetto ricevuto è corretto oppure un **NACK** (Not ACK) se invece è errato. Il sender in questo caso rimane in attesa di ACK o NACK, a seconda di cosa riceve o rispedisce l'ultimo o si mette di nuovo in attesa.
- **RdT 2.1** - nuovo problema: se l'errore è nella fase di ritorno allora il sender rimane bloccato. Se ricevo qualcosa che non ci capisco nulla e voglio sbloccare il receiver allora gli rimando il mio pacchetto corretto, sperando che arrivi qualcosa al receiver (con errore o non) e la risposta di quest'ultimo mi arrivi corretta, in modo da sbloccare il tutto. Sembra un ciclo infinito, ma non lo è in quanto prima o poi il procedimento andrà a buon fine. Mi rimane solo riuscire a far capire alla destinazione in qualche maniera se è una copia che sto mandando o se è un nuovo pacchetto, perché se non ho la certezza che il mio NACK o ACK arrivi a destinazione, ad esempio, non so se il nuovo pacchetto arrivato è una copia di quello vecchio perché l'ACK arriva sbagliato oppure è un pacchetto nuovo perché l'ACK è arrivato corretto. Devo distinguere solo queste due situazioni e per farlo mi serve solo un bit. Il sender imposta il bit a 0 e invia il pacchetto e si mette in attesa dell'ACK o NACK, se riceve un NACK rimane rimanda il pacchetto e rimane in attesa, se riceve un ACK imposta il bit a 1 e manda il nuovo pacchetto con il bit a 1, sempre così il ciclo. Il receiver rimane in attesa di un pacchetto con lo 0 e di un pacchetto con l'1. Se ho ricevuto un pacchetto corretto da 0 passo a 1, se invece ricevo un pacchetto con ancora lo 0, allora è un pacchetto vecchio per cui mando lo stesso la conferma per sbloccare il sender, ma rimango in attesa con il bit invariato.
- **RdT 2.2** – posso evitare i NACK per ridurre le complicazioni, rispedendo al posto del NACK, l'ACK relativo all'ultimo pacchetto ricevuto correttamente. Ovviamente continuo ad utilizzare il sistema 0/1 per numerare le sequenze.
- **RdT 3.0** – i pacchetti possono anche perdere. Per gestire la perdita di pacchetti utilizziamo un timeout; se spediamo qualcosa e non abbiamo più risposta attiviamo un timer che allo scadere re-invia il pacchetto. Il receiver invece non cambia. Questo è il così detto protocollo **Stop&Wait**.

## Protocolli Stop&Wait, Pipeline, Go back N e Ripetizione Selettiva

Facciamo un po' di conti: Supponiamo di avere un canale a 10Mbps, ovvero in 1s passano 10.000.000 bit, cioè 1 bit dura 0.1 μs. Supponiamo inoltre di avere una frame su questo canale di 1500byte che dura 1200μs. Supponiamo di avere un canale lungo 1km e che la velocità di propagazione del segnale è all'incirca 200.000 km/s, il tempo di propagazione è di circa 5μs; in 5μs ci stanno 50bit. 1500byte quindi si trasferiscono in 1200μs + 5μs. Nella figura



vediamo perché: al tempo 0 la frame è all'inizio del canale, al tempo 5μs l'inizio della frame ha già raggiunto la fine del canale, al tempo 1200μs ha finito di generare tutti i bit e al tempo 1205μs la frame è passata per intero. Per cui per far passare tutta la frame devo considerare il tempo di propagazione più la durata della frame. Nella figura successiva vediamo come

funziona il sistema a livello di intervalli: al tempo 5 la il primo pezzo di frame arriva a destinazione, al tempo 1200 inizia la propagazione del rimanente della frame e il tutto arriva al tempo 1205. Successivamente viene messo un tempo di elaborazione di 5μs, per capire se la frame è corretta o no, dopo di chè parte l'ACK e dopo 5μs arriva. Il totale è 1266.2μs per spedire 1500byte più l'ACK per capire che è arrivata correttamente. Meglio di tanto non si può fare, in quanto prima di spedire un altro pezzo dobbiamo

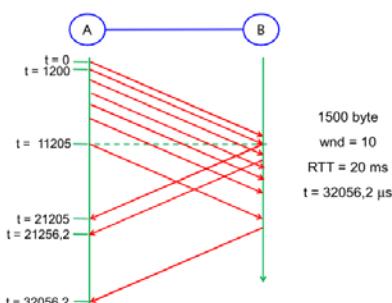
avere la conferma che la frame è arrivata ed è corretta. Sapendo che spediamo 1500byte in 1266.2μs, la velocità effettiva del nostro canale è  $1500 * 8 / 1266.2 \mu s = 9,477 Mbps$ , ovvero il protocollo Stop&Wait ha rallentato il nostro canale, ma meglio di così non può fare. Potrebbe fare meglio riducendo l'ACK, ma non risparmiamo proprio tantissimo, però non sono risultati cattivi.

Supponiamo che le macchine siano più lontane, le cose cambiano non poco. Chiamiamo **Round Trip Time (RTT)** il tempo impiegato dal segnale, per andare, arrivare a destinazione e tornare indietro. Facciamo un confronto tra valori:

RTT 10µs       $t = 1266,2\mu s$       9,477Mbps

RTT 20ms       $t = 21256,2\mu s$  0,564Mbps ----> 564kbps

Aumentando le distanze la velocità effettiva del canale è scesa di parecchio e meglio di così con questo protocollo non posso fare, in quanto devo aspettare necessariamente l'ACK per poter andare avanti. Se il canale è molto lungo,



la frame è molto più piccola del canale, per cui lo occupa per un tempo molto piccolo lasciando il canale per la maggior parte del tempo inutilizzato. La soluzione quindi è prendere una serie di pacchetti e farli andare uno dietro l'altro in **pipeline**. Se invio i pacchetti in questa maniera, ovviamente inizio a riempire il canale, utilizzandolo appieno. Facciamo qualche calcolo per verificare che la situazione effettivamente migliori: Abbiamo sempre un RTT di 20ms e inviamo 10 pacchetti prima di inviare l'ACK (**finestra di invio**). In totale abbiamo un tempo di invio e risposta di 32056,2µs ma la velocità effettiva del canale è

arrivata a 3,743Mbps. Posso fare di meglio aumentando il numero di pacchetti da inviare prima di rispondere con l'ACK, per cui se riesco a coprire quasi interamente tutto il canale riesco quasi ad arrivare ai 10Mbps della velocità ideale del canale. Ovviamente il controllo è molto più pesante, in quanto ci sono in sospeso x frame. Se le distanze sono brevi non ha senso usare la pipeline.

Esistono due tipi di protocolli che seguono la filosofia di pipeline: uno è **Go back N** e l'altro è **Ripetizione selettiva**. Si basano sul concetto di finestra scorrevole; questa finestra è scorrevole in quanto, mano mano che i dati vengono confermati la faccio andare avanti.

**Go back N**: Nel Go back N la finestra si divide in: dati già confermati, quelli che sono stati inviati ma ancora nessun riscontro, spazio ancora disponibile dove si possono inviare dati. Per ogni cella in attesa devo tenere un timer di controllo per verificare che arrivino. Mando i pacchetti 1, 2, 3, 4 e 5, ma il 2 si è perso; il protocollo avvia il timer per il pacchetto 1, all'interno del timer riceve l'ACK1 per cui si rende conto che è arrivato. Il 2 non viene confermato, per cui il protocollo torna indietro a rispedire 3, 4 e 5, che erano già stati spediti e arrivati, solo che il mittente non lo può sapere, in quanto ogni volta torno indietro l'ACK1; per non sapere cosa fare torna semplicemente indietro. Il nome infatti dice tutto, Go back N (torno indietro di N steps) e ricomincia. Il destinatario, che non ha ricevuto mail il pacchetto 2 e continua ad inviare l'ACK1, appena riceve il pacchetti 3,4 e 5 non li memorizza.

Oppure si può fare in un'altra maniera. Si perde sempre il 2, ma appena scade il timeout rispedisce il 2 ed il destinatario invece di buttare 3, 4 e 5 se li tiene in memoria; il destinatario una volta ricevuto il pacchetto 2, conferma questo pacchetto. Il mittente dopo aver re-inviato il pacchetto 2, spedisce il 6, in quanto gli altri erano già arrivati. Fa una Ripetizione selettiva. Sembra un'evoluzione, però c'è un problema, in quanto se si dovessero perdere più pacchetti consecutivi, può passare molto tempo prima di riprendere il sincronismo; invece Go back N tornando indietro e rispedendo, mi permette di recuperarlo immediatamente.

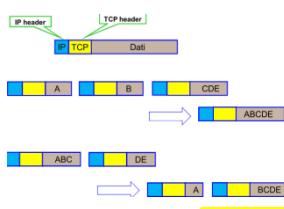
Se io ho una serie di perdite che coinvolge molti pacchetti consecutivi, è meglio usare Go back N; se invece ho diversi pacchetti persi distribuiti, allora conviene Ripetizione selettiva. Tutto dipende da come sono distribuite le perdite.

## Introduzione a TCP (Transmission Control Protocol)

Le funzionalità di TCP sono:

- Fornisce funzionalità di connessione a livello di trasporto
- Connection Oriented
- Possiede un meccanismo di gestione e packaging dei dati
- Si preoccupa di trasferire questi dati

- Si preoccupa di fornire un servizio affidabile e di qualità
- Fornisce controllo di flusso e gestione della congestione
- E' bidirezionale



Il processo si interfaccia con una socket che scrive i dati da inviare, che vengono messi in un buffer, il TCP prende questi dati dal buffer, li inserisce in un pacchetto e li spedisce; il TCP del ricevente prende i dati dal buffer di ricezione e l'applicazione svuota quando vuole e come vuole il buffer per prelevare i dati. E' stream oriented, ovvero che i dati da socket a socket viene visto come un flusso continuo e non partizionato, la parte TCP spezzetta in realtà i dati, ma i processi non percepiscono questo spezzettamento; il controllo di flusso è fatto da buffer di invio e di ricezione e non tra le due applicazioni. Facendo riferimento alla figura, l'applicazione vede i dati ABCDE solo quando sono completi, ma TCP per poterli inviare potrebbe anche averli spezzati.

La comunicazione funziona secondo il **Modello Client-Server**, che funziona nella seguente maniera: ci deve essere un client ed un server, ovvero una macchina spenta che si sveglia solo per spedire dati ed una macchina che rimane sempre in ascolto per ricevere dati. Una volta che il server riceve i dati, manda una risposta al client. Il funzionamento è il seguente: il server rimane in attesa di connessione, il client chiede di connettersi ed il server, se non è oberato di lavoro, accetta la connessione effettuando una fork del suo processo, in quanto una metà rimarrà in ascolto di nuove richieste, l'altra metà invece gestirà i dati entranti dal client. Perché questo? Se nella realtà noi inviamo una richiesta al servizio in ascolto sulla porta 80, se non si sdoppiasse, il servizio rimarrebbe bloccato finché non termina di processare i dati inviati dal client. Invece il server, effettuando una fork, copia i processi che gestiscono la connessione in ingresso e nel frattempo apre una nuova porta, il processo copia chiude la porta 80, spostando così la comunicazione sulla nuova coppia di porte, il processo padre invece chiude definitivamente la connessione ritornando in ascolto sulla porta 80, che è nuovamente libera.

## Formato dei pacchetti TCP

Sopra l'header TCP, si trova quello IP. TCP ha bisogno di:

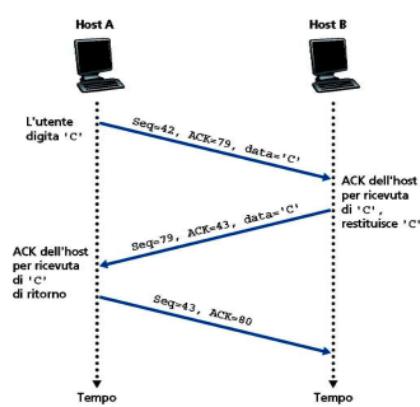
Bit offset	Bits 0–3	4–7	8–15	16–31
0		Source port		Destination port
32			Sequence number	
64			Acknowledgment number	
96	Data offset	Reserved	CWR ECE URG ACK PSH RST SYN FIN	Window Size
128			Checksum	Urgent pointer
160			Options (optional)	
160/192+			Data	

- Un numero di porta sorgente e uno di destinazione [16bit X 2]
- Numero di sequenza [32bit], indica lo scostamento (espresso in byte) dell'inizio del segmento TCP all'interno del flusso completo, a partire dall'*Initial Sequence Number*, negoziato all'apertura della connessione
- Numero di riscontro [32bit], ha significato solo se il flag ACK è impostato a 1, conferma la ricezione di una parte del flusso di dati nella direzione opposta (detto in parole semplici, riporta indietro l'ACK della connessione precedente), indicando il valore del prossimo Sequence number che l'host mittente del segmento TCP si aspetta di ricevere. Per 32bit possiamo permetterci di inserire un numero per ogni byte della comunicazione.
- Lunghezza intestazione [4bit], indica la lunghezza dell'header del segmento TCP; tale lunghezza può variare da 5 word (20 byte) a 15 word (60byte) a seconda della presenza e della lunghezza del campo facoltativo Options.
- Non usato [4bit], riservati a usi futuri, tutti settati a 0.
- Una serie di Flags [8bit], utilizzati per il controllo del protocollo:
  - CWR (Congestion Window Reduced), se settato a 1 indica che l'host sorgente ha ricevuto un segmento TCP con il flag ECE settato a 1 (aggiungo all'header in RFC 3168)

- **ECE** (ECN-Echo), se settato a 1 indica che l'host supporta ECN (Explicit Congestion Notification) durante il 3-way handshake (aggiunto all'header in RFC 3168)
- **URG**, se settato a 1 indica che nel flusso sono presenti dati urgenti alla posizione (offset) indicata dal campo Urgent pointer
- **ACK**, se settato a 1 indica che il campo Acknowledgment number è valido
- **PSH**, se settato a 1 indica che i dati in arrivo non devono essere bufferizzati, ma passati subito ai livelli superiori dell'applicazione
- **RST**, se settato a 1 indica che la connessione non è valida; viene usato in caso di grave errore, a volte utilizzato insieme al flag ACK per la chiusura della connessione
- **SYN**, se settato a 1 indica che l'host mittente del segmento vuole aprire una connessione TCP, con l'host destinatario e specifica nel campo Sequence number il valore dell'Initial Sequence Number (ISN); ha lo scopo di sincronizzare i numeri di sequenza dei due host. L'host che ha inviato il SYN deve attendere dall'host remoto un pacchetto SYN/ACK
- **FIN**, se settato a 1 indica che l'host mittente del segmento vuole chiudere la connessione TCP aperta con l'host destinatario. Il mittente attende la conferma dal ricevente (con un FIN-ACK). A questo punto la connessione è ritenuta chiusa per metà: l'host che ha inviato FIN non potrà più inviare dati, mentre l'altro host ha il canale di comunicazione ancora disponibile. Quando anche l'altro host invierà il pacchetto con FIN impostato la connessione, dopo il relativo FIN-ACK, sarà considerata completamente chiusa.
- **Finestra di ricezione** [16bit], indica la dimensione della finestra di ricezione dell'host mittente, cioè il numero di byte che il mittente è in grado di accettare a partire da quello specificato dall'acknowledgment number. Il buffer è di dimensione limitata ed il compito di TCP è di regolare l'afflusso dei dati, se il client è lento a consumare i dati deve rallentare i flusso. In caso di buffer pieno il client scrive in questo campo uno 0 quando invierà l'ACK, in modo da comunicare che non può essere inviato più nulla. La situazione può essere sbloccata solo dal client, ovvero da chi possiede il buffer di ricezione.
- **Checksum** [16bit], Campo di controllo utilizzato per la verifica della validità del segmento.
- **Urgent pointer** [16bit], puntatore a dato urgente, ha significato solo se il flag URG è settato a 1 ed indica lo scostamento in byte a partire dal Sequence number del byte di dati urgenti all'interno del flusso.
- **Options**, Opzioni facoltative per usi del protocollo avanzati. Un possibile utilizzo potrebbe essere mettere d'accordo le due parti sulla dimensione massima dei pacchetti da trasferire.
- **Data**, rappresenta il carico utile o payload da trasmettere, cioè la PDU (Protocol Data Unit) proveniente dal livello superiore.

## Finestra di ricezione e gestione ACK in TCP

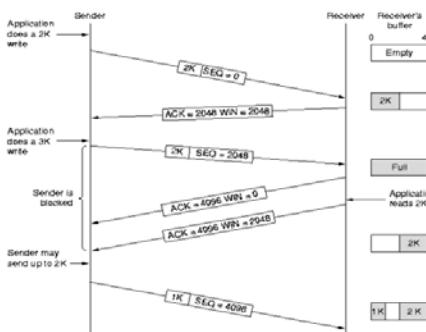
Come funziona il meccanismo? Supponiamo di avere un file bello grande da dover spedire. Innanzi tutto sorgente e destinazione si mettono d'accordo sul **MSS** (Maximum Segment Size), che nel nostro esempio abbiamo settato a 1000, questo valore lo ricavano dal MTU a livello DLL, se sotto c'è Ethernet sappiamo che più di 1500byte non



possiamo trasferire. C'è la possibilità che i router attraversati possano frammentare i segmenti replicando però i preamboli, ciò porta ad uno spreco nella banda utilizzata. Quello che fa TCP solitamente è di considerare i pacchetti, solo se sono interi, per cui in questo caso la frammentazione è un grosso danno; TCP utilizza un protocollo di tipo **sliding window** per la gestione del flusso dati. Come funziona? Prendiamo un esempio considerando TELNET (servizio che gestisce una shell in remoto); premo un tasto, il codice viene inserito in un pacchetto e spedito alla controparte, questo codice viene inserito nel buffer del destinatario e nello stesso momento viene rispedito indietro al mittente con l'ACK e il carattere digitato in modo da poterlo visualizzare nel client. Questo doppio passaggio, che sembra inutile, serve a fornire una sorta di conferma visiva e della velocità di comunicazione a chi scrive. Il numero di sequenza identifica la posizione del primo carattere nello stream dei dati, l'ACK invece identifica il prossimo carattere disponibile per la ricezione, ovvero

segnalà che fino al quel carattere la trasmissione è andata a buon fine. Nella figura vediamo che spedendo solo il carattere "C", il numero di sequenza è 42, mentre l'ACK è 43, se avessimo invece spedito la sequenza "CI", avremmo avuto sempre numero di sequenza 42, ma ACK a 44. Questo è il meccanismo del **Piggybacking**, usato da TCP per rispedire indietro l'ACK per evitare di sprecare pacchetti.

Vediamo ora come funziona il meccanismo delle **finestra di ricezione**. Supponiamo di avere una comunicazione come in figura, dove il buffer del ricevente è grande 4k e il sender deve inviare un po' di dati.



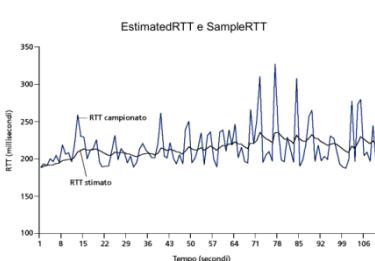
L'applicazione inserisce 2K di dati nel buffer del sender e spedisce questi dati con numero di sequenza 0, il ricevente ottiene tutti i dati senza problemi, per cui spedisce al sender un ACK a 2048, per segnalare che la comunicazione è andata a buon fine e che la finestra (WIN) è di 2048, ovvero lo spazio rimasto nel buffer. Il sender riceve questo ACK e spedisce altri 2K dati (2048), il receiver ottiene tutti i dati, ma nel frattempo non ha svuotato il buffer, per cui spedisce un ACK a 4096, ma WIN a 0, in quanto il buffer è pieno. Il sender si blocca finché il receiver, liberato 2K di spazio sul buffer, spedisce al sender un ACK con lo stesso numero 4096 (molto importante ciò) e WIN a 2048. Il sender spedisce gli ultimi 2K di dati con SEQ a 4096.

Considerando sempre l'esempio di TELNET, immaginiamo di avere due macchine poste ad una grande distanza tra loro (quindi con grande latenza di comunicazione), il receiver con un buffer molto piccolo e molto lento a macinare dati. Spedire pacchetti con un carico dati di 1byte è pura follia, in quanto sulle lunghe distanze sarebbe uno spreco di banda e di tempo allucinante. Sulla LAN Ethernet a corte distanze, possiamo permetterci di fare una cosa del genere in quanto possiamo spedire pacchetti a tempesta. Quindi, se la comunicazione è su lunghe distanze, è conveniente accumulare dati e poi inviare; nel caso della shell remota, l'effetto visivo non è più vedere restituito un carattere alla volta, ma una serie di righe. Se per caso i tempi di risposta sono troppo lunghi, perché magari il receiver è lento, il sender può ipotizzare che il destinatario sia defunto oppure che la connessione sia crollata, per evitare inutili allarmismi, il receiver manda ogni tot di tempo dei pacchetti ACK, con l'ultima sequenza ricevuta corretta e la finestra a 0 per segnalare che è ancora operativo e che sta lavorando.

## I tempi di Round Trip, Gestione dei Timer e Fast Retransmit

Una volta che mandiamo un pacchetto avviamo un timer, se è troppo breve c'è il rischio di congestionare la rete con una miriade di pacchetti duplicati, d'altro canto se è troppo lungo, rischiamo su reti troppo grandi di accorgerci di aver perso un pacchetto e quindi di rispedirlo, dopo un'enormità di tempo. Come faccio a valutare come fissare il timer, che è una cosa molto importante? In teoria dovrei conoscere dove si trova la controparte, ma non posso, però posso stimare il tempo di **round trip** di ogni singolo pacchetto, calcolando quanto tempo passa dalla spedizione del pacchetto all'ACK di risposta. Per cui conoscendo le stime delle varie comunicazioni, posso ricavare un valore unico applicando la seguente formula:

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT} \text{ dove tipicamente } \alpha = 0.125$$



Ovvero si utilizza una **EWMA** (Exponential weighted moving average), ovvero una media esponenziale; se utilizzassimo una semplice media aritmetica potremmo ottenere dei valori altamente falsati, in quanto basta semplicemente avere una lunga serie di valori molto alti e vecchi, ed una nuova serie di valori bassi ma nuovi, che i valori più alti hanno più peso di quelli bassi, non tenendo in conto che magari si trattano di vecchi campioni. La logica vuole invece, che sia più significativo controllare cosa è accaduto nel passato recente, che nel passato remoto, per cui

potrei memorizzarmi tutti i valori e assegnare ad ogni singolo valore un peso decrescente, man mano che ci allontaniamo nel tempo: viene fuori una cosa enorme. Invece questo sistema, tiene conto della storia passata, ma in maniera esponenzialmente più limitata, rispetto ai nuovi valori; EstimatedRTT è la media stimata allo step

precedente, SampleRTT è l'ultimo campione misurato, insieme vengono pesati con quel valore  $\alpha$  e si ottiene così il nuovo valore EstimatedRTT. Il valore  $\alpha$  fa in modo che, il valore ottenuto in precedenza, pesi sempre di meno. Come vediamo in figura, la media esponenziale segue l'andamento del campione preso in esame. Per il primo pacchetto, non avendo valori "storici", assegnamo il valore massimo, in quanto non abbiamo la più pallida idea di dove si trovi la controparte, sarà il successivo passaggio, per il secondo pacchetto, ad aggiustare sensibilmente la media, in quanto con questo sistema di  $(1-\alpha)$  e  $\alpha$ , assestiamo immediatamente la situazione (un valore molto alto, moltiplicato per  $1-\alpha$ , peserà pochissimo, rispetto al valore campionato in quel momento, dando così precedenza al valore reale).

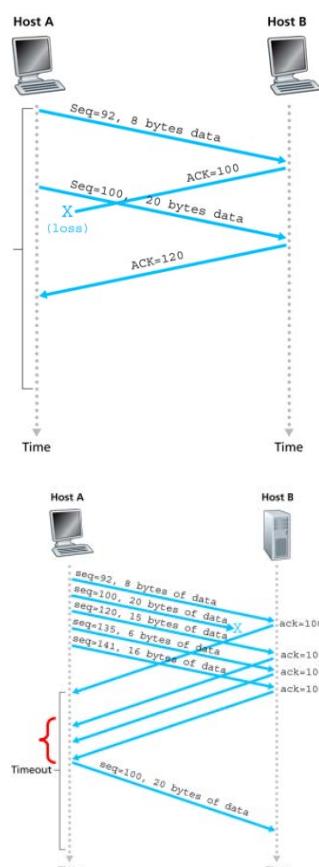
TCP effettua anche una stima sulla variabilità dei tempi di Round Trip, in quanto devo tenere conto del fatto che in precedenza si è avuto un valore altissimo e successivamente uno molto basso. La formula che tiene conto di questo è:

$$\text{DevRTT} = (1 - \beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}| \text{ dove } \beta = 0.25$$

Dove la variazione viene calcolata, tenendo in conto la vecchia variazione è il valore assoluto del campione misurato reale e del tempo di round trip stimato. A questo punto posso ottenere il **Timeout** da applicare alla comunicazione, mediante la formula:

$$\text{Timeout} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

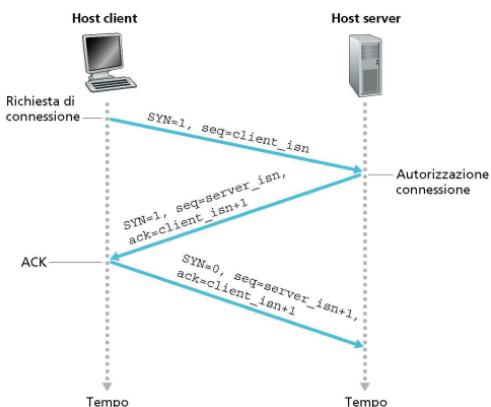
Si moltiplica per 4 volte il DevRTT per dare tutto il tempo che serve al pacchetto di arrivare; se nonostante ciò un pacchetto fa comunque scattare il timeout, perché si perde, allora il prossimo timeout viene settato al doppio del valore precedente. Il timer, in caso di timeout, aumenta in maniera esponenziale; raggiungo un valore di timer troppo grande, mi fermo e comunico che c'è un problema sulla linea.



TCP abbiamo visto che utilizza un protocollo sliding window, per cui invia più segmenti contemporaneamente, però utilizza un solo timer; invece di settare un timer per ogni pacchetto, ne utilizza solo uno per il primo. Se riceve un ACK per il primo, il timer si azzerà e passerà al secondo; abbiamo una grossa imprecisione, però abbiamo già abbondato abbastanza per la determinazione del timer, per cui la speranza è quella di perdere il meno possibile, se comunque avviene una perdita il timer c'è per cui scatterà. In figura viene rappresentato cosa significa in pratica. Il sender invia un pacchetto con sequenza 92 di 8bytes, il receiver lo riceve e spedisce al sender un ACK con 100, però questo ACK si perde, nel frattempo il sender ha spedito un altro pacchetto con sequenza 100 di 20bytes, il receiver lo riceve e spedisce l'ACK con 120 che viene ricevuto dal sender. Quest'ultimo ACK segnala al sender che tutti i dati, fino a 119 sono correttamente arrivati al receiver, perché il fatto di comunicare di aver ricevuto fino a 119 di sequenza, implica che ho ricevuto anche i primi 100 di cui si era smarrito l'ACK. E se invece perdiamo un pacchetto? Analizziamo la situazione in figura. Il sender invia una serie di pacchetti con sequenze rispettive 92,100, 120, 135 e 141, il primo pacchetto arriva e viene confermato con un ACK 100, ma il secondo non arriva. Il destinatario, che si vede arrivare questi pacchetti fuori sequenza, allora invia un duplicato dell'ACK dell'ultimo pacchetto arrivato in ordine, per cui per i restanti pacchetti arrivati restituisce al mittente una serie di ACK 100. Il mittente, che si vede arrivare questa serie di ACK 100, si rende conto che il primo è arrivato sicuro, poi salta una sequenza di non so quanto e poi successivamente c'è una sequenza che è arrivata, lui non sa con esattezza cosa si è perso. Una volta che mi arrivano tutti questi ACK duplicati e non aspetto che il timer del pacchetto perso scatti (nel nostro caso manca il pacchetto due, ma può capitare che se ne perdano altri in sequenza), allora rispedisco immediatamente il pacchetto con sequenza 100, nel frattempo il timer di questo pacchetto si raddoppia e appena questo pacchetto arriva al destinatario, allora confermerà tutta la sequenza in ordine ricevuta (nel nostro caso invierà un ACK = 157). Tutto questo meccanismo di recupero del pacchetto perso si chiama **Fast Retransmit**, in quanto il pacchetto perso viene rispedito

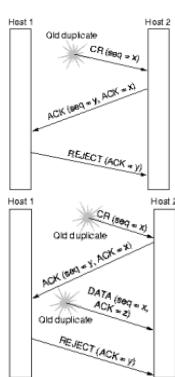
immediatamente appena il mittente si vede arrivare una serie di ACK duplicati (in media 3) e non aspetto che scatti il timer.

## Apertura e chiusura delle connessioni



Il client ed il server si devono mettere d'accordo sull'instaurazione del canale di comunicazione. Il client inizia a vivere dalla richiesta di connessione, mentre il server deve essere, ovviamente, operativo da molto prima; il client invia un primo pacchetto con  $SYN = 1$  ed il flag  $ACK = 0$ , perché essendo il primo pacchetto della sequenza, l' $ACK$  non può riferirsi ad un precedente pacchetto, per cui quel flag indica che il campo  $ACK$  deve essere ignorato, per cui  $SYN = 1$  e  $ACK = 0$  identifica sempre il primo pacchetto, per cui non può arrivare un pacchetto di una qualsiasi sequenza se prima non c'è stato questo pacchetto. La risposta dal server è sempre  $SYN = 1$  e  $ACK = client\_isn + 1$ , ovvero il valore di sequenza del client + 1. Entrambi client e server creeranno dei numeri di sequenza distinti, che andranno a caratterizzare i rispettivi ACK. Questa risposta è un'autorizzazione alla connessione. Il client risponde a questa autorizzazione con un ACK. Questo meccanismo viene detto **Three Way Handshake**, ed è un meccanismo di sicurezza per istaurare la connessione, che si chiama **sfida**; come funziona? Un server che si vede arrivare una richiesta di connessione deve verificare che sia di una macchina che è ancora presente in rete, perché potrebbe capitare che si tratti di una vecchia richiesta che si era persa e che vagando alla fine è arrivata; per cui il server accetta la proposta del client di connettersi, però prima di aprire il canale chiede al client di spedirgli il suo  $client\_isn + 1$ , dopo che il server gli ha inviato  $client\_isn$ . Ovvero, se il client è online dopo aver ricevuto 100, può calcolare 101, ma se invece non è online in quel momento, se il server invia 100, il client risponde con un numero a caso. Questi tre pacchetti sono **fortemente** legati tra loro, in quanto il secondo si può generare solo a partire dal primo ed il terzo solo a partire dal secondo. Questa sfida assicura che due macchine sono online contemporaneamente e che devono calcolare i valori quando la sfida viene lanciata.

L'attacco **DoS** (Denial of Services) si basa su questo meccanismo, in quanto chiede una serie di connessioni al server per poi sparire impedendo al server di completare il 3-way handshake, bloccando di fatto le risorse del server, che non può più accettare connessioni in ingresso. Il server in realtà è dotato di un timeout, ma se le richieste sono a valanga il sistema si intasa. La prima soluzione per contrastare quest'attacco è limitare il numero di richieste da parte di uno stesso host; l'evoluzione dell'attacco per scavalcare questo sistema è il **DDoS** (Distributed DoS), dove un hacker buca più macchine e utilizza quelle per sommersere il target di richieste. Per sconfiggere quest'attacco doto la mia connessione di un accessorio: se ricevo un pacchetto  $SYN = 1$  e  $ACK = 0$ , non lo spedisco subito a destinazione ma lo memorizzo. L'accessorio verifica che quei pacchetti sono tutti confermati e se lo sono li manda avanti. E' una specie di firewall che protegge il server da pacchetti "strani".

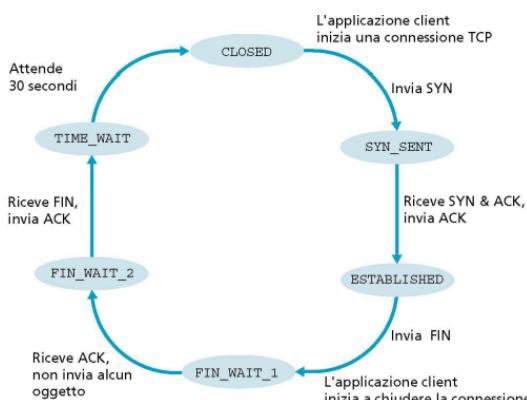
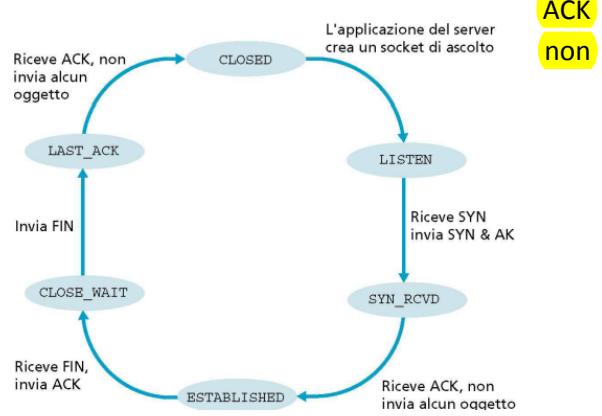


Può accadere che il server riceva una richiesta di connessione che si era persa in passato e manda la sfida al client che aveva fatto richiesta. Il client non più interessato alla connessione manda un REJECT al server, per rifiutare la connessione, oppure fa silenzio, in quanto è consapevole che il server passato un tot di tempo eliminerà la richiesta.

Altra cosa che può accadere è che il server riceva una richiesta vecchia e lancia la sfida e una volta lanciata questa sfida riceva dalla rete una vecchia risposta alla sfida, che ovviamente viene ignorata in quanto presenta valori non conformi (bisogna veramente essere sfortunati per ricevere una risposta ad una sfida vagante con i valori della sfida nuova). Il server quindi ignora la vecchia risposta alla sfida e il client risponde sempre con il REJECT oppure fa silenzio.

Il problema dell'apertura è risolto, nel senso che esiste un algoritmo che porta ad una situazione in cui la connessione è correttamente stabilita. Se fallisce qualche cosa non si fa la connessione.

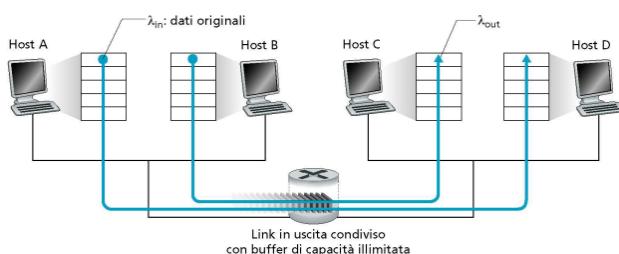
La chiusura della connessione non permette di avere un protocollo affidabile. In quanto quello che accade è sostanzialmente il dilemma dei fidanzatini al telefono (chiudi tu...no chiudi tu...no dai chiudi tu...etc), ovvero il problema dell'ultimo pacchetto, che non può essere confermato, perché se venisse confermato non sarebbe l'ultimo. Se non viene confermato allora viene rimosso ed il problema viene spostato al penultimo, e così via fino ad eliminarli tutti. Una soluzione pratica comunque è necessaria: il client manda un pacchetto con  $FIN = 1$  e riceve l' $ACK$  dal server, dopodiché il server invia il suo pacchetto con  $FIN = 1$  e il client risponde con il suo  $ACK$ , fatto questo si attende un tot di tempo e si chiude la connessione. Se l'ultimo si perde, la macchina che lo attendeva rimane in attesa, ma se ricevo nulla attivo un timer e scaduto il tempo chiudo la connessione di default. Oppure se si perde il secondo  $ACK$ , la macchina che inizializza la chiusura aspetta un tot di tempo e rimanda la richiesta di chiusura. Il caso più disperato è quando si perde il secondo  $ACK$  e in successione si perdono tutti i tentativi di richiesta di chiusura, a quel punto scaduto sempre un tot di tempo, la connessione viene brutalmente chiusa, in quanto comunque c'è sicuramente un problema di connessione che non permette comunque comunicazione. Di default se non ricevo più connessioni chiudo tutto.



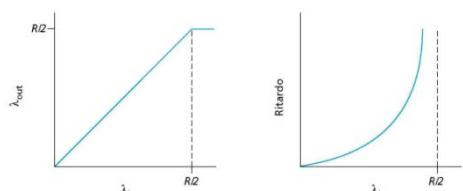
## La congestione: aspetti teorici

Possono esistere due casi: nel primo è il ricevente che rallenta la connessione, in quanto presenta un buffer di ricezione troppo piccolo che impone al sender di rallentare la trasmissione, nel secondo invece potrebbe essere presente nella rete una strozzatura, che rallenta la trasmissione, anche se il receiver ha un buffer molto capiente. Questa comunque non è vera e propria congestione in quanto i dati prima o poi arriveranno, la vera congestione è quando i dati non arrivano completamente a destinazione.

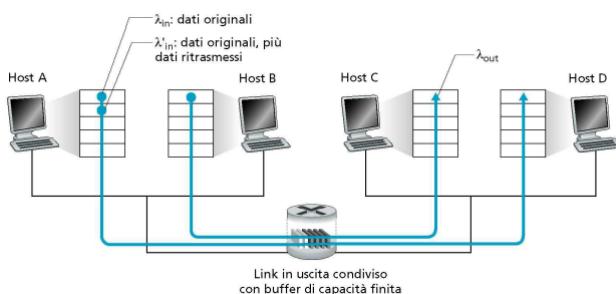
### Primo Scenario



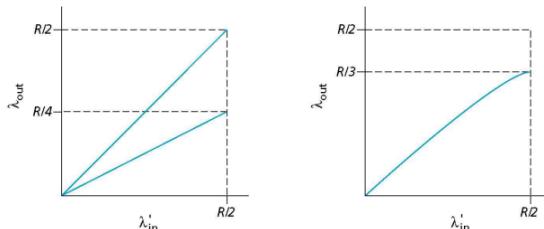
Abbiamo due macchine come in figura, con un link in uscita condiviso con un buffer di capacità illimitata. Quello che succede è che man mano che  $\lambda_{out}$  e  $\lambda_{in}$  occupano tutta l'ampiezza di banda il ritardo cresce potenzialmente, in quanto, anche se abbiamo un buffer infinito, per cui i pacchetti non si perdono, questi pacchetti arriveranno a destinazione dopo un tempo infinito.



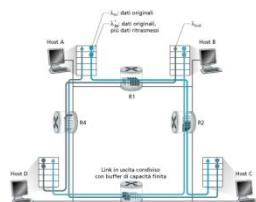
### Secondo Scenario



dalla macchina mittente finché non riesce a passare. Nel frattempo il ricevente non riceve nulla e sta fermo. In questo caso lo scompenso dei dati  $\lambda'_{in}$  (ovvero i dati originali, più le copie da ritrasmissione) è molto più grande, in quanto dei primi ne arrivano solo alcuni.



### Terzo Scenario



Questo è uno degli scenari più terribili, in quanto, visto i cammini dei pacchetti, potrebbero accadere delle collisioni che fanno perdere ACK e quindi generare copie, intasando così la rete.

In generale è questa la **congestione**, dove copie di pacchetti non confermati intasano il sistema fino a saturarlo. A questo punto quindi serve un controllo della congestione specifico; diversi protocolli prevedono un controllo diretto della rete attraverso l'utilizzo di token. Il mittente ha una serie di token ed è autorizzato a spedire pacchetti solo per quel tot token, appena li finisco mi fermo; quando il destinatario mi autorizza posso ottenere nuovi token per poter trasmettere altri pacchetti. In questo modo i timer scattano, ma il mittente, se non ha più token non può trasmettere più; il numero di token è proporzionale alla capacità della rete. ATM in particolare prevede, nel pacchetto, un flag che comunica se c'è confusione o no; quando quella cella arriva a destinazione, chi vede la cella e si accorge che c'è confusione, zittisce il sistema per un tot di tempo in modo da smaltire il traffico in eccedenza, finché qualcun altro non comunica che il traffico è finito.

### Controllo della congestione in TCP

Il TCP implementa il controllo della congestione, congiuntamente al controllo di flusso, solo agli estremi della comunicazione (end-to-end) e non richiede alcun supporto da parte dei router intermedi per realizzare questa funzione. Questo è coerente con il modello progettuale di IP, che prevede di aggiungere "intelligenza", ovvero funzioni di elaborazione ai nodi terminali, lasciando ai router mansioni meno complesse e proprie del livello di rete. Tale controllo è realizzato in modo *reattivo* anziché *preventivo* ovvero valutando istante per istante lo stato di congestione di rete e agendo di conseguenza. In particolare, il suo funzionamento si basa sull'uso di una variabile TCP detta **CongWin** (Congestion Window). Tale variabile impone un vincolo alla quantità di dati trasmessi e non ancora riscontrati dal mittente, ovvero i dati che sono stati consegnati per la trasmissione al livello di rete, che sono in viaggio sulla rete o in fase di elaborazione da parte di TCP sul nodo destinazione, o i cui ACK sono a loro volta in viaggio sulla rete stessa. Se il mittente rileva che sul percorso di invio dei pacchetti si verificano condizioni di scarso traffico, incrementa il tasso a cui trmette i pacchetti, mentre al contrario, se il traffico rilevato è alto, lo riduce. Si fissa un valore massimo per tale finestra di trasmissione, si assume uno stato di congestione quando si ha perdita di pacchetti (ovvero mancanza di ACK alla scadenza del time-out relativo al segmento da ricevere) ed infine si regola costantemente la finestra di trasmissione a seconda del livello di congestione di rete rilevato modulando il tasso di trasmissione dei pacchetti inviati e apportando così la necessaria stabilità: maggiore è il livello di congestione più

Ma un buffer di dimensione infinita non esiste, per cui arriviamo al secondo scenario dove, abbiamo la medesima situazione di prima, ma utilizzando un buffer finito. Se il buffer è di dimensione finita allora dobbiamo mettere in conto le ritrasmissioni a causa delle perdite, per cui vanno creati dei duplicati continui che fanno ingolfare il sistema, in quanto se non riesce a passare viene buttato via e ricreato un'altra volta

dalla macchina mittente finché non riesce a passare. Nel frattempo il ricevente non riceve nulla e sta fermo. In questo caso lo scompenso dei dati  $\lambda'_{in}$  (ovvero i dati originali, più le copie da ritrasmissione) è molto più grande, in quanto dei primi ne arrivano solo alcuni.

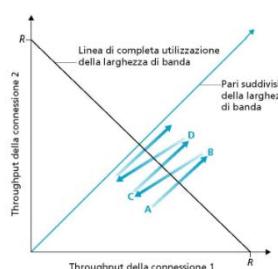
piccola sarà la finestra di trasmissione, viceversa minore sarà il livello di congestione maggiore sarà la finestra di trasmissione.

L'algoritmo di controllo di congestione presenta due fasi:

- Partenza Lenta – Slow Start:** Il mittente TCP inizia trasmettendo il primo segmento dati ed attende un riscontro. Ogni volta che un segmento trasmesso viene riscontrato il mittente incrementa la finestra di congestione **CongWin** di una quantità pari al MSS (Maximum Segment Size). Quindi ad ogni RTT la CongWin raddoppia di dimensioni fino a raggiungere la metà della dimensione massimo. Da questo punto in avanti l'aumento procede in maniera lineare fino a raggiungere il massimo. Questo valore viene mantenuto finché non si incorre in un evento di congestione.
- Aumento Additivo – diminuzione moltiplicativa:** al verificarsi di ogni perdita di pacchetto, il valore della CongWin viene dimezzato. Se si verifica un altro smarrimento la CongWin viene dimezzata ancora. Il valore della CongWin può continuare a diminuire ma non scende sotto 1MSS ovvero la dimensione in byte massima di un segmento TCP. Nel caso non si riscontri congestione lungo il canale, probabilmente significa che sarà disponibile una larghezza di banda non utilizzata e quindi il mittente TCP incrementerà la propria CongWin di 1 MSS ad ogni tempo di andata e ritorno. Riassumendo quindi il mittente TCP incrementa la propria frequenza in modo additivo quando il percorso è libero da traffico, invece la decremente in modo moltiplicativo quando rileva che il percorso è congestionato.

Le due versioni del protocollo principali (TCP Reno e TCP Tahoe) sono compatibili tra loro; ne esistono molte altre ma non tutte sono compatibili tra loro.

## Fairness tra connessioni concorrenti



Se ho più connessioni parallele TCP, litigano per chi deve prevalere sull'altro oppure si distribuiscono la banda amichevolmente? Ovvero TCP è **Fair** oppure no?

Ipotizziamo di avere due connessioni TCP (Connessione 1 e 2), che viaggiano sullo stesso canale e che presentano le medesime caratteristiche. Nel diagramma vediamo il grafico del Throughput delle due connessioni: la linea blu è il Troughput ideale per cui si ha un'equa suddivisione di banda, le varie frecce sono i vari aggiustamenti che le due reti fanno per dividersi la banda. Cosa succede? Iniziamo con il punto A e crescono in maniera lineare, l'aumento è parallelo alla retta ideale, per cui non andiamo ne meglio ne peggio. Arrivati al punto B abbiamo una perdita, per cui il protocollo prevede di diminuire il ConWin, perdendo un po' qualcosa, arrivando al punto C; dopo il punto C si cresce linearmente fino al punto D. Si va sempre così fino ad avvicinarsi sempre di più alla linea ideale di suddivisione di banda; il sistema man mano si auto assesta. Stiamo però sempre parlando di un sistema ideale, in quanto non è detto che converga così velocemente; può capitare che uno dei due abbia una perdita e l'altro no (le perdite però sono sempre bilanciate), oppure che i mezzi di comunicazione o le impostazioni non siano le stesse.

Ogni tanto un picco fa sballare il sistema, però alla fine il tutto si auto ottimizza, giungendo alla suddivisione perfetta del segnale.

## Parte 6 – Livello Applicativo



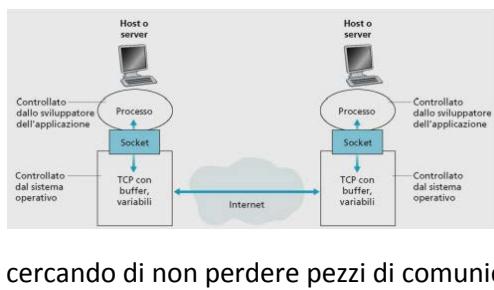
I protocolli a livello applicativo sono innumerevoli; non sempre fornisce un'interfaccia verso l'utente. Alcuni protocolli realizzano qualcosa di intermedio tra il livello di trasporto e l'applicazione vera e propria; ad esempio il web è composto da una serie di elementi distinti, tra cui HTTP che è il protocollo vero e proprio, pur non fornendo una vera e propria interfaccia verso l'utente; si tratta di qualcosa che viene accostato ad un protocollo di trasporto ma questo non va confuso con il livello. In particolare HTTP fornisce delle primitive, che poi possono essere utilizzate mediante i browser. Alcuni protocolli P2P utilizzano il protocollo HTTP come livello di comunicazione al posto di TCP/IP; GNU-tella è uno di questi protocolli P2P.

## Comunicazione tra i processi

I processi a livello applicativo desiderano semplicemente comunicare nella maniera migliore possibile, necessitano di uno stream di comunicazione continuo, veloce, affidabile e con un basso ritardo di comunicazione.

Le tipologie di comunicazione sono **Client-Server**: ovvero che da una parte c'è un client che fa richiesta di un dato servizio e dall'altra parte un server sempre pronto (o quasi) a soddisfare la richiesta del client. Il client mentre il server elabora, potrebbe andare anche offline e risvegliarsi quando deve ricevere la risposta, al contrario del server che deve rimanere sempre online. A livello di trasporto tutte le connessioni sono Client-Server, anche le comunicazioni P2P, che da una parte possono ricevere traffico da parte dell'esterno e dall'altro possono generare traffico sulla rete. Le comunicazioni IPv4 sono tutte di tipo Client-Server.

Rimanendo nell'ambito p2p, supponiamo di aver bisogno di una data risorsa, all'utente non interessa chi possiede la risorsa, per cui comunica all'OverLay Network che necessita di quella risorsa e chiede di indicargli chi la può fornire. IPv4 però non permette di implementare un Overlay network vero e proprio, in quanto in teoria dovrei inviare la comunicazione al sistema di macchine che compongono la rete p2p in generale, in pratica devo contattare una macchina ben precisa; in IPv6 invece la situazione è nettamente differente, in quanto con l'indirizzamento Anycast la cosa è fattibile. Questa tipologia di reti sta diventando sempre più importante, in quanto molte società si stanno interessando alle reti p2p, per gli enormi vantaggi che possiede; supponiamo che Microsoft in una certa data rilascia le nuove patch di sicurezza, il problema è che in quella data i loro server saranno stracarichi di richieste, per cui potrebbero anche piantarsi. Con l'utilizzo di p2p si può pensare un meccanismo di distribuzione differente: una macchina si è già connessa al server Microsoft per scaricare l'aggiornamento e un'altra macchina si deve ancora connettere, prima di inviare la richiesta al server, verifica che qualche macchina presente nella rete abbia già l'aggiornamento e lo possa condividere; andando avanti così riusciamo a decentralizzare la distribuzione e non sovraccaricare i server centrali. Ci sono diversi problemi legati a questo meccanismo di aggiornamento, in quanto io potrei spacciare una patch malevola per quella originale. Altro scenario dell'utilizzo di reti p2p è il video-on-demand; questa tipologia di servizio in genere richiede dei server molto potenti, per gestire il flusso di dati in uscita.



L'applicativo è un processo, che si interfaccia con una socket aperta localmente che utilizza TCP/IP o UDP sul livello di trasporto, tramite il livello di trasporto si affaccia su internet e ci si sposta sulla controparte che svolte il lavoro inverso. Essenzialmente tutto questo passaggio, permette di implementare un sistema di comunicazione tra due processi che vanno avanti contemporaneamente, cosa molto importante, cercando di non perdere pezzi di comunicazione.

## Indirizzamento nelle comunicazioni tra Processi: Well known Ports, Registered Ports e User Ports

Le porte sono suddivise in 3 gruppi:

- **Well Known Ports** (0 – 1023) Sono le porte dei servizi di sistema.
- **Registered Ports** (1024 – 49151) Sono assegnate da *Internet Corporation for Assigned Names and Numbers* per qualche uso.
- **User Ports (Dynamic and/or Private Ports)** (49152 – 65535) non sono legate ad usi ben precisi.

La differenza tra il primo ed il secondo gruppo è molto importante, vediamolo con un esempio: sappiamo che la porta web sono la 80 e la 8080. Se dovessimo connetterci ad un sito di una banca e la connessione avvenisse mediante la porta 8080, allora non avremmo una connessione sicura, perché? I sistemi protetti non permettono ad utenti che non possiedono le credenziali di root, di creare (o accedere ad) un servizio utilizzando le porte da 0 a 1023, ma lo permettono mediante porta da 1023 in per cui se io necessito di connettermi ad un servizio web sicuro presso un server e quel server mi comunica che si trova presso la porta 8080, capisco subito che **non** si tratta di un servizio sicuro, in quanto quel servizio può essere stato creato da chiunque; il vero servizio sicuro si trova

invece alla porta 80 che può essere inizializzata solo da un processo con credenziali root. Esistono infatti protocolli quali TELNET, FTP, SSH, che richiedono credenziali, proprio per accedere al primo gruppo di porte, che necessita credenziali di accesso stabilite dall'amministratore di sistema. La differenza quindi tra il primo ed il secondo gruppo di porte sta proprio in questo: il primo è il gruppo di porte di cui si ci può ciecamente fidare (se comunque la provenienza del servizio è lecito), in quanto il servizio a cui si collega quella porta è stato sicuramente creato da un amministratore di sistema (a meno di intrusioni ben più gravi all'interno del sistema), il secondo gruppo di porte invece ci presenta dei servizi che non necessitano credenziali particolari per essere creati, per cui non presentano una grande sicurezza e se vengono chieste credenziali su queste porte, allora c'è sicuramente qualcosa che non quadra. Ovviamente un servizio sulle prime porte non è sempre affidabile, in quanto, se voglio rubare delle credenziali di un sito, mi copio la pagina di accesso a quel sito e lo carico sul mio server, uso magari il primo gruppo di porte e frego l'utente medio. Un modo per difendersi da uno scenario del genere è verificare l'esatta provenienza della pagina web, in quanto, ad esempio, se mi ritrovo la pagina di login del sito delle poste italiane in Indocina, allora c'è qualcosa che non quadra. Un modo per stabilire se la fonte è affidabile o no, ma ne parleremo più avanti nella sezione di sicurezza, è il certificato. Alcune aziende offrono come servizio la firma autorevole di certificati da concedere ai siti web; l'università di Catania non si è affidata ad una azienda che fa questo servizio, firmandoselo da sola il certificato. Nel caso in cui un browser si vede arrivare un certificato con una firma non autorevole (ovvero non proveniente da grosse aziende del settore) avverte che potrebbe esserci qualche problema, sta all'utente in quel caso stabilire se la fonte è autorevole o no.

## Requisiti delle applicazioni

Non sempre il massimo dei requisiti è quello richiesto, alcuni applicativi richiedono cose particolari, sotto alcuni esempi:

Applicazione	Perdita dei dati	Larghezza di banda	Sensibile al tempo
Trasferimento di file	NO	Elastica	NO
E-mail	NO	Elastica	NO
Documenti Web	NO	Elastica(pochi kbit/s)	NO
Audio/Video in tempo reale	Tollerabile	Audio: pochi kbit/s – 1Mbit Video: 10kb – 5Mbit	Si, centinaia di ms
Audio/Video memorizzati	Tollerabile	Come sopra	Si, pochi secondi
Giochi interattivi	Tollerabile	Pochi kbit/s – 10kbit	Si, centinaia di ms
Messaggi istantanei	NO	Elastica	Si e No

- **Perdita dei dati**

Una connessione affidabile è necessaria per il trasferimento file e l'e-mail, ad esempio, in quanto tutte le informazioni scambiate devono arrivare, anche se ci mettono molto tempo. I giochi online invece non necessitano di una connessione affidabile, in quanto si predilige la velocità di arrivo dei pacchetti; ad esempio, se utilizzassi una connessione affidabile nel gioco, rischierai di ritrovarmi l'avversario in punti diversi (es. un fps tengo sotto mira il nemico), perché le informazioni arrivano tutte, ma con tempistiche troppo lunghe; con una connessione affidabile, magari in caso di errore vedo l'avversario muoversi a scatti, ma riesco comunque a seguirlo. Per le applicazioni Audio/Video si può fare anche utilizzare un protocollo non affidabile, in quanto anche qui è molto più importante ottenere i dati velocemente, che non tutti i dati corretti, in quanto i dati persi possono sempre essere ricavati da quelli presenti (es. mediante interpolazione). Su questi sistemi dove è "tollerata" la perdita di dati, si va a definire un **QoS (Quality of Service)**; su un trasferimento file la QoS non ha senso, in quanto in casi di dati errati quel file posso prenderlo e buttarlo per intero, ma su un sistema di giochi o video-on-demand, la QoS è definibile come **quanti pacchetti perdo, in funzione al numero totale di pacchetti persi**.

- **Larghezza di banda**

Un trasferimento file, email o documenti web necessitano di una banda Elastica, in quanto a seconda dell'esigenza si deve aumentare o no la potenza di trasmissione. Per quanto riguarda l'audio/video quelli in tabella sono i valori medi che si riscontrano, che oltre a indicarci i valori entro cui dovremmo essere per poter usufruire del servizio, ci indica anche quanto tempo potremmo metterci a vedere un video o sentire un

brano. I giochi interattivi richiedono una larghezza di banda che faccia passare pochi kbit/s, in quanto chi progetta giochi di questo genere, deve a tutti i costi ridurre all'osso la fase di comunicazione e concentrare tutta la parte computazionale nelle varie macchine.

### • Sensibile al tempo

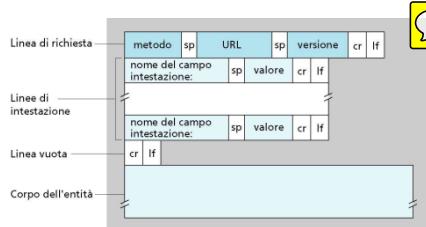
Trasferimento di file, e-mail e documenti web, non dovrebbero essere sensibili al tempo, anche se quando apriamo una pagina web o trasferiamo file vorremmo che le operazioni si eseguissero in tempi ragionevoli. Audio/video in tempo reale e memorizzati invece hanno politiche differenti, in quanto nel primo non sono tollerati ritardi, nel secondo invece un ritardo iniziale (buffering), è tranquillamente tollerato. Nei video memorizzati possono anche permettermi variazioni nei tempi di trasmissione, dilatando il tempo di riproduzione del video. Per i giochi interattivi non sono tollerati ritardi, per i messaggi istantanei dipende.

## Protocollo HTTP



Nata dall'idea di due ingegneri del CERN, per rendere le ricerche del centro di Ginevra, accessibili in tutta la struttura in maniera completamente telematica. L'idea base è una macchina che fa una richiesta ad un server che risponde. Serve un browser che ha un duplice scopo: il primo è quello di interpretare e leggere la pagina scritta in HTML, che è il linguaggio di markup con cui vengono sviluppate le pagine web, dall'altra parte realizza l'interfaccia con il protocollo http verso il mondo esterno. http semplicemente va a cercare la risorsa su internet, specificando il nome della macchina ed il file (la risorsa) all'interno della macchina; in principio questo file era fisicamente presente all'interno della macchina, con l'avvento dei linguaggi di Scripting (es. PHP), si preferisce generarlo al momento della richiesta. I due ingegneri stabilirono anche la modalità di collegamento tra le varie macchine mediante l'**HyperLink**.

Sono presenti due versioni: la 1.0 e la 1.1. Inizialmente nella 1.0 la connessione TCP veniva chiusa dopo il trasferimento di un singolo oggetto e questo era fattibile quando le pagine erano composte da pochi oggetti; con l'avvento del web moderno e quindi di pagine sempre più grandi, gli oggetti aumentarono, per cui aprire e chiudere una connessione per un oggetto soltanto divenne qualcosa di troppo dispendioso, per cui nella versione 1.1 venne aggiunta la possibilità di aprire una connessione TCP permanente.

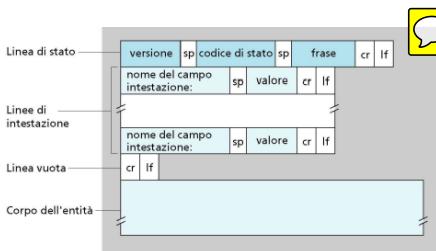


Un pacchetto TCP di richiesta è formato da una serie di righe di testo, che segue una struttura come in figura. Dove inseriamo, il metodo richiesto, l'URL e la versione del protocollo (intervallati da spazi), a capo inseriamo il nome del campo intestazione e il valore (intervallati da spazi), andando a capo per ogni nuovo valore, ed infine una linea vuota e subito dopo il corpo dell'entità.

### Es. Richiesta HTTP

```
GET /home/home.html http/1.1
Host: www.dmi.unict.it
Connection: close
User-agent: Mozilla/4.0
Accept-language: it
...
```

Ogni browser comunica al server di cosa si tratta (Mozilla, Explorer, etc..) in modo che il server possa decidere che tipo di formattazione utilizzare (se previsto in fase di progettazione); questo è sia un pregio che un difetto, in quanto può essere utilizzato per scopi malevoli. La guerra dei browser che scoppio un po' di anni fa diede vita a questa cosa, in quanto i browser si differenziano tra loro per il linguaggio di scripting utilizzato, per cui una pagina che si vede bene in Explorer, non è detto che si veda bene su Mozilla. L'accept-language è una chicca di programmazione, in quanto un sito web fatto bene, sfrutta questo campo per selezionare la lingua in automatico, permettendoti l'eccezione di cambiare lingua; un sito fatto male invece lascerà all'utente il tedioso rito di selezionare la propria lingua, da una lista. Il campo connection indica cosa si dovrà fare della connessione una volta esaudita questa richiesta, in questo caso verrà chiusa. Il campo Host, permette di realizzare un servizio di Hosting di domini virtuali, più o meno quello che fa il server galileo.



Un pacchetto TCP di risposta è simile a quello di richiesta, ma non lo stesso.  
Nella prima riga presenta la versione del protocollo, il codice di stato (es. 404) e la frase (NOT FOUND) di ritorno.

Es. Risposta http

HTTP/1.1 200 OK

Connection: close

Date: Thu, 06 Aug 2003 12:00:00 GMT

Server: Apache/1.3.0 (Unix)

Last-Modified: Mon, 22 Jun 2002 11:23:00 GMT

Content-Length: 2653

Content-Type: text/html

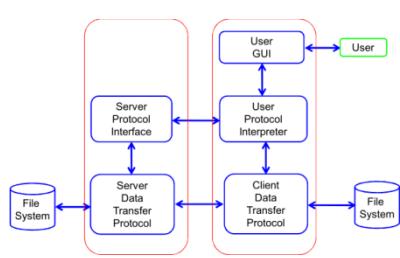
...

Il DHTML (Dynamic HTML) è un insieme di tecnologie che permettono di cambiare in modo dinamico la rappresentazione e i contenuti di un documento ed aumentare l'interattività dell'utente sulla pagina. Non è un vero e proprio linguaggio ma è una sorta di contenitore di script a cavallo tra il Javascript, l'HTML ed il CSS. In generale un User, mediante il Browser, interagisce con il server che a sua volta avvia uno script dinamico, che basandosi magari su un database, costruisce la pagina, restituendola al server che la farà visualizzare al browser. In principio questi script erano dei veri e propri programmi in C, che man mano vennero sostituiti da linguaggi di scripting per problemi di sicurezza, in quanto il linguaggio di scripting, se ben configurato, evita che qualche mal intenzionato possa distruggere il server.

L'utente richiede un accesso ad un database, che se contiene informazioni private, può essere accessibile solo dopo un'autenticazione, ma il protocollo HTTP è del tipo invio richiesta, risposta e chiusura, ovvero ogni comunicazione HTTP a livello di trasporto sono connessioni separate tra di loro. Nel livello di OSI c'è un livello di sessione, che raccoglie tutte queste connessioni separate, in un'unica sessione, ma in TCP questa cosa non esiste, quindi ogni sessione applicativa corrisponde ad una sessione di trasporto, a meno di non legare queste sessioni mediante un "biscottino", alias un Cookie. L'idea è, faccio una richiesta, mi viene chiesto di autenticarmi e invio queste informazioni, in risposta alle credenziali di autenticazione ricevo un Cookie, ovvero un numero random generato dal server, dopo che ha vagliato la mia richiesta, questo numero viene utilizzato in tutte le mie richieste e le successive risposte, che sono legate, dal punto di vista logico, alla precedente fase di autenticazione. Questo numero viene memorizzato nell'User-agent; finito il lavoro il cookie viene buttato. Ci sono però siti web che il biscottino se lo conservano, lasciando di fatto la sessione di autenticazione aperta; per cui se io riesco a prendere quel biscottino, posso accedere al sito come utente autenticato.

Le informazioni passate mediante POST, vengono passate in chiaro, per cui se vado a fare un bookmark della pagina, riesco ad ottenere quelle variabili e se contengono login e password, è una grossa falla di sicurezza. Questa cosa si può evitare cifrando l'header mediante il protocollo HTTPS.

## Protocollo FTP (File Transfert Protocol)



File Tranfer Protocol, è uno dei protocolli più vecchi in quanto era la cosa più richiesta all'epoca e presenta alcune differenze rispetto a HTTP: la prima tra tutte è che utilizza due connessioni differenti, entrambe TCP, su due porte distinte, 20 e 21, una per gestire i dati e l'altra per i comandi. Questo sistema si chiama, Sistema di connessione con dati fuori banda, questa diversificazione serve per avere due canali distinti su cui inviare i comandi e i dati; ad esempio se ho un trasferimento in corso e voglio bloccarlo, invece di inviare il comando tramite la connessione sulla porta 20 (trasferimento dati), che potrebbe rimanere bloccato, lo mando tramite la porta 21 che

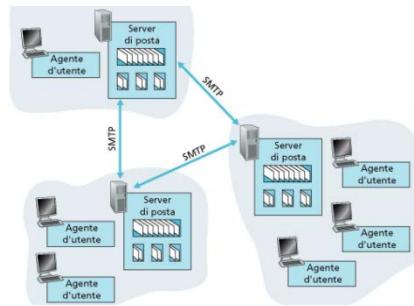
serve esclusivamente per inviare i comandi. Lo schema a blocchi è come quello in figura: il trasferimento file è gestito dal livello più basso, a cui è collegato direttamente il file system delle macchine, che comunicano (scambiano dati) mediante il canale dati; questo livello più basso è gestito da un livello di controllo, uno per il client e l'altro per il server, collegati tra loro dal canale di controllo. A livello del client poi troviamo l'interfaccia grafica ed infine l'utente. Per utilizzare il protocollo è necessaria un'autenticazione e viene stabilita una connessione persistente (al contrario di HTTP), ovvero che vive finché l'utente non la chiude oppure finché non scatta un timeout. Un'altra differenza con HTTP è che il trasferimento è di tipo bidirezionale.



## Protocolli SMTP (Simple Message Transfer Protocol), POP e IMAP



SMTP è un protocollo per il trasferimento di file, in quanto la posta in origine funzionava nella seguente maniera: la posta era un unico file di testo e il trasferimento di messaggi consisteva nel prendere un pezzo di questo file e accodarlo alla fine del file di un'altra macchina. È un sistema **push**, dalla mia macchina invio il pezzo di file fino alla



macchina di destinazione; invece con HTTP, prendo dalla macchina remota e porto nella mia macchina. Per cui il protocollo base di comunicazione, che permette ai server di scambiarsi queste informazioni è smtp. Come funziona globalmente il sistema? Sono presenti diversi server di posta che comunicano tra di loro mediante il server smtp, l'utente ha un proprio User Agent per l'e-mail e comunica al proprio server di posta, mediante dei protocolli che devono semplicemente garantire l'accesso alla propria casella di posta. **POP e IMAP**. La mail viene inviata riga per riga. La posta è un file di testo, per cui sono necessari

in totale 127 caratteri, ovvero quelli leggibili, inoltre in qualunque lingua del mondo, non si inizia mai la mail con un punto (.), per cui questo carattere, a inizio comunicazione, può essere utilizzato per inviare delle direttive, come il mittente o il destinatario. Con il passare degli anni, nacque l'esigenza di inviare degli allegati, che possono anche non essere del testo, ma oggetti, mediante la codifica **MIME**, che permette di trasformare gli oggetti in una sequenza di caratteri e quindi renderli inviabili.



POP3 invia login e password al server di posta e chiede tutta la posta, che viene interamente trasferita nella macchina di destinazione; questo protocollo è comodo solo se abbiamo sempre la stessa macchina con cui ci connettiamo al provider di posta, in quanto cambiando computer non si ricorda se abbiamo già letto la posta oppure no, in quanto potrebbe copiare la posta nel pc di destinazione e lasciare l'originale intatto sul server. Un protocollo più intelligente è IMAP4, che permette proprio l'aggiornamento della situazione dei messaggi (letti, cancellati, in arrivo, etc...). Il problema di entrambi i protocolli è che inviano login e password in chiaro, infatti vengono appoggiati da **SSL**, che permette la cifratura dei dati in arrivo e in uscita, soprattutto le credenziali di accesso.



## Il protocollo DNS (Domain Name System)

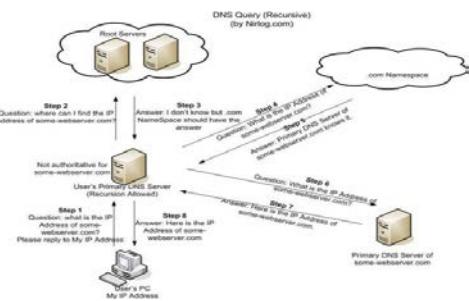


E' un servizio base che fornisce, una serie di primitive a quasi tutti i servizi a livello applicativo. Sostanzialmente esegue una traduzione, da un nome simbolico ad un indirizzo IP che identifica la macchina. Il nome di una macchina IP viene identificato da una combinazione massima di 255 caratteri nel formato *nome.nome.nome.nome*, non c'è un massimo di letterali seguiti dal punto, possiamo scegliere quanti ne vogliamo; il minimo è di due letterali separati da un punto. Per quanto riguarda l'università di Catania, all'interno dello stesso dominio, ci sono macchine identificate da 3 letterali (es. [www.dmi.unict.it](http://www.dmi.unict.it)) e macchine identificate da 4 (es. galileo.dmi.unict.it). L'indirizzo IP ha una strutturazione gerarchica, come abbiamo visto in precedenza; per quanto riguarda il nome DNS ha anche una struttura gerarchica, è la parte più importante è la prima a partire da destra, ovvero il **dominio di primo livello**, che dovrebbe più o meno indicare dove si trova la macchina (.it -> italia, .eu -> europa, etc...). Viene utilizzato questo sistema con i nomi, per una questione prettamente mnemonica, il problema è che bisogna tenere traccia di questi nomi associati agli indirizzi IP; il problema è facilmente risolvibile utilizzando una tabella che, in una colonna possiede gli indirizzi IP, nell'altra i corrispettivi nomi. Il problema è dove tenere questa tabella, per cui si è pensato di distribuirla in maniera intelligente tra varie macchine, manovra agevolata dal sistema di suddivisione dei nomi radice. In quanto, tenere una tabella unica dei nomi sarebbe troppo dispendioso, per cui decido di suddividerla a fette a seconda del dominio di primo livello. Ad esempio, la macchina galileo.dmi.unict.it sarà localizzata nella seguente

maniera: l'host richiedente contatta il server DNS globale, che reindirizza la macchina verso il server che contiene i domini .it, a sua volta quel server reindirizza la macchina ad un altro server DNS, che contiene tutti i domini unict.it (quindi una macchina all'interno dell'università di Catania), che a sua volta reindirizza l'host richiedente al server DNS del dipartimento, che contiene tutti i domini dmi.unict.it, che riuscirà finalmente a indicargli l'indirizzo IP della macchina galileo. Dal nome possiamo andare a rintracciare la macchina che gestisce la singola partizione della tabella.

Quando si invia una richiesta verso l'esterno, deve essere comunque inserito un indirizzo IP per spedire il pacchetto, se l'Host non sa a chi inviare, perché ancora non conosce l'indirizzo IP relativo a quell'URL, allora si rivolge al server DNS suo diretto superiore, ovvero a quello che noi settiamo in fase di configurazione, quando connettiamo il pc alla rete internet. Ovviamente il sistema a cascata è nel caso peggiore in cui nessuna macchina riesce a risolvere completamente il nome al primo colpo, se invece possiede memorizzato l'indirizzo IP dell'host completo, allora reindirizza direttamente la macchina all'indirizzo IP corrispondente. Una volta che l'host richiedente ottiene l'indirizzo, se lo salva in una cache temporanea DNS, in modo da poter risolvere le connessioni successive immediatamente; ovviamente questa cache del DNS non rimane fissa per sempre, ma, come tutte le cache, viene aggiornata ogni tot di tempo.

E' raro che navigando su internet si utilizzino direttamente gli indirizzi IP, per cui per ogni sessione di navigazione, a meno di non avere già salvato in cache i nomi, avvengono continue chiamate ai server DNS, che se non sono abbastanza potenti da riuscire a risolvere milioni di nomi, allora bloccano gli host. Questo sistema è molto pesante e critico per internet, per cui è necessario rendere più agile tutto questo sistema. Se dobbiamo mettere su un sistema molto grosso cosa facciamo? Per prima cosa creiamo un server DNS primario, dove inseriamo tutti gli indirizzi da risolvere, poi creiamo diversi server DNS secondari, che ogni tot di tempo si scaricano la tabella del primario, diventando delle esatte copie e rimanendo sempre coerenti con l'originale; in seguito pubblico verso l'esterno il DNS primario, ma evito che all'interno della mia rete sia oggetto di richieste, in modo che sia più scarico possibile. I DNS secondari invece sono quelli che si beccano tutto il traffico e se dovesse crollare non abbiamo grossi danni.



Come si vede in figura, fare una richiesta al server DNS è come fare una serie di domande; a volte può capitare di ricevere una risposta del tipo **"Not authoritative for ...."** che sta ad indicare che la risposta è arrivata da un server DNS secondario, ciò però non significa che la risposta sia falsa.



Ad ogni nome DNS possono corrispondere diversi tipi di informazioni. Per questo motivo, esistono diversi tipi di **record DNS**. Ogni voce del database DNS deve essere caratterizzata da un tipo. I principali sono:



- **A:** corrispondenza tra un nome ed un indirizzo IPv4
- **AAAA:** corrispondenza tra un nome ed un indirizzo IPv6
- **CNAME:** alias dell'host. Questo vuol dire che la macchina ha più di un nome legato allo stesso indirizzo IP, uno degli utilizzi di questo tipo di record consiste nell'attribuire ad un host che offre più servizi un nome per ciascun servizio. In questo modo, i servizi possono poi essere spostati su altri host senza dover riconfigurare i client, ma modificando solo il DNS.
- **MX:** (Mail eXchange) server e-mail per il dominio. L'indirizzo come sappiamo è del tipo nome@dominio.estensione; un tempo la posta era su una macchina ben precisa, per cui l'indirizzo era del tipo nome@nomemacchina.dominio.estensione, ovvero è saltata il nome della macchina. Sostanzialmente collega un nome di dominio ad una lista di server di posta autorevoli per quel dominio, in quanto solitamente il servizio di posta elettronica è gestito da molte macchine coerenti tra loro.
- **SRV:** server per un dato servizio
- **NS:** (Name Server) server DNS autoritativo
- **SOA:** (Start of Authority) gestione delle zone DNS
- **TXT:** Testo



Identificazione	Etichette	
Numero di domande	Numero RR di risposta	12 byte
Numero di RR assoluti	Numero di RR aggiuntivi	
Domande (numero di domande variabile)		Nomi, tipi di campi per una richiesta
Risposte (numero variabile di record di risorse)		RR in risposta a una richiesta
Assoluti (numero variabile di record di risorse)		Record di server assoluti
Informazioni aggiuntive (numero variabile di record di risorse)		Informazioni aggiuntive "utili" che possono essere usate

## Parte 7 – La sicurezza nelle reti

Può accadere una cosa abbastanza spiacevole, se qualcuno riesce a spacciarsi per server DNS, può risolvere i nomi URL inseriti dagli utenti, per macchine fasulle. Facciamo un esempio: un tizio lascia la sua connessione wifi aperta apposta e crea al suo interno un server DNS, un tizio B si accorge della rete aperta e decide di connettersi, ovviamente la prima cosa che risolve i suoi nomi URL è il server DNS creato ad hoc, chi ha creato il server DNS può reindirizzare l'utente che si connette a pagine fasulle, per poter facilmente carpire informazioni confidenziali.

Se vogliamo navigare in maniera sicura su una rete wifi aperta, conviene utilizzare una macchina sicura d'appoggio; immaginiamo di avere una macchina in dipartimento da shell, con quattro righe di codice si può mettere su un programmino che riceve i pacchetti e li gira ad un indirizzo ben preciso; questa prima connessione si può proteggere come vuole. SSH si può utilizzare come un banale tunnel, che poi a sua volta sbuca su internet.

Le comunicazioni su internet sono in generale in chiaro; TELNET manda la password in chiaro, perché? Il file delle password contiene al suo interno le password di tutti gli utenti, codificate con un algoritmo HASH, inoltre questo file non è visibile da tutti, ma solo dal super user. Quando mi vado a loggare da remoto le cose sono due, ho mando la password già “hashata”, ma in questo caso un probabile ascoltatore potrebbe catturare la password “hashata” e andare direttamente ad inserire nei pacchetti, per cui l'alternativa è inviarla in chiaro e poi hasharla nella macchina dove bisogna loggarsi, che andrà poi a confrontarla con quella presente nel file delle password. Il massimo della insicurezza.

Quando si parla di sicurezza bisogna parlare di quattro proprietà:

1. **Riservatezza** delle informazioni, nei confronti di utenti non autorizzati
2. **Autenticazione** della controparte
3. **Non-ripudiabilità** delle comunicazioni (transazioni) effettuate, in modo da non potersele rimangiare
4. **Integrità** dei dati, se mando a dire una cosa non ci deve essere nessuno che la deve modificare

La sicurezza la dobbiamo piazzare dappertutto e quando progettiamo un sistema, dobbiamo già pensarla con la sicurezza già inclusa. Abbiamo diversi livelli in cui possiamo piazzarla:

- **Utente:** ovvero prendere l'utente e certificarlo, per cui è necessario autenticare l'utente sulla macchina
- **Processi:** possono essere sicuri solo se l'hardware supporta la sicurezza, per cui vedi sotto
- **Hardware:** intel ha pensato alla sicurezza a partire dal processore 386, ovvero i processori precedenti non avevano alcun riguardo per la sicurezza, in quanto i processi potevano andare in tutta la memoria. A partire dal 386 hanno inserito dei flag da assegnare ai processi, in modo da suddividere delle varie “aree d'accesso”. Il meglio di tutti è **TPM (Trusted Platform Module)**, ovvero un chip che ha memorizzato dentro di sé alcune chiavi uniche per ogni chip, in modo da effettuare delle cifrature uniche.
- **Mezzo Fisico**

Sicurezza non significa cifratura, in quanto non è l'unica cosa che serve e può anche essere omessa. Ci sono vari metodi utilizzabili quali:

- **Steganografia:** consiste nel nascondere informazioni all'interno di un'altra.
- **Chaffing and Winnowing:** Il destinatario, conoscendo la chiave k, non deve far altro che calcolare il MAC di ogni elemento e scoprire l'unico corretto, tra la miriade di dati inutili che il mittente invia. Gli altri, di fronte ad una miriade di informazioni, tra cui quella corretta, non sanno che fare se non provarle tutte finché non trovano quella corretta. Mittente e destinatario si accordano su due cose:
  - Un **MAC** (Message Authentication Code) simile ad una funzione hash
  - Una chiave **K**
- **Crittografia:** l'arte di codificare e decodificare l'informazione, senza perdita di parti di essa.

Qualsiasi cosa non può fare l'avversario bisogna dimostrarlo in maniera rigorosa.

## Cifrari a sostituzione e a trasposizione

Consiste nel sostituire una lettera con un'altra (o un gruppo con un altro gruppo). Quello più semplice di tutti consiste nel far corrispondere le lettere dell'alfabeto, con una serie di lettere dell'alfabeto translitterato (il metodo che usava Cesare nel suo De Bello Gallico); la sicurezza si deve basare sulla **chiave** e non sull'algoritmo, in quanto è probabile che l'avversario conosca l'algoritmo ed i suoi punti deboli. Se alle lettere dell'alfabeto, faccio corrispondere una serie di lettere messe a casaccio, sono necessari 26! (un numero enorme) di tentativi per trovare la chiave. Perché comunque un documento scritto con questo metodo si buca in pochissimo tempo? Perché se scritto in Italiano, basta andare a controllare quante volte si ripete una data lettera (ad esempio le vocali si ripeteranno molte volte) e seguire le regole della lingua italiana, che con una analisi statistica si può arrivare a decodifica.

Cosa potrei fare per rendere il cifrario ancora più resistente? Potrei assegnare alla prima lettera il valore della prima lettera data da una traslitterazione, poi cambio traslitterazione e alla seconda lettera assegno il valore della seconda di quest'altra traslitterazione e così via; in questo modo perdo completamente la frequenza. Oppure potrei prendere il testo e zipparlo, perché quando io vado a comprimere qualcosa, in pratica vado ad assegnare una frequenza ad ogni simbolo e se cambio il valore di questa frequenza, rendendoli tutti equi frequenti, l'attacco non si può più effettuare. Il primo metodo è più sicuro, solo che devo avere un meccanismo che mi dice come la traslitterazione cambia da una lettera all'altra, ovvero la traslitterazione di n deve essere legata alla traslitterazione di n-1 e così via. Questo era il meccanismo che utilizzava la macchina Enigma dei Tedeschi.

I cifrari a **trasposizione** operano invece effettuando le modifiche in base alla posizione dei caratteri, e non in base al loro valore. La parola chiave viene utilizzata per determinare le modalità del riordino.

## Metodo one-time pad

In realtà un metodo sicuro esiste, ed è questo. Prendo una sequenza random e la utilizzo come chiave per effettuare la traslitterazione o più banalmente faccio una operazione di XOR, tra il carattere del testo e quello della chiave. Funziona solamente, quando il testo da cifrare è uguale alla dimensione della chiave e quest'ultima non deve più riutilizzata; se la chiave di cifratura è più corta, posso effettuare attacchi sulle frequenze. Questo era il metodo utilizzato dalle spie negli anni '60. Di fatto questo sistema è inutilizzabile.

## Principi per la crittografia

Per utilizzare la crittografia per scopi di sicurezza è necessario soddisfare due principi:

- 1) Un messaggio cifrato deve contenere della **ridondanza**, in modo che quel pacchetto abbia una validità solo dopo una certa azione. Ad esempio se dobbiamo prelevare dei soldi dal bancomat, prima di completare l'operazione ci chiede una conferma che sarà unica, se qualcuno subito dopo si aggancia e rimanda quell'informazione e il sistema telematico si vede arrivare lo stesso codice precedente, blocca tutto. Un esempio nella vita reale: le aziende usavano un trucchetto quando assumevano le donne, insieme al contratto facevano firmare un foglio in bianco, che poi le stesse aziende, utilizzavano per presentare le dimissioni della lavoratrice, se andava in maternità. Per evitare questo scherzetto è stata introdotta una normativa, per cui, quando si presentano le dimissioni, bisogna applicare un particolare bollo emesso lo stesso giorno in cui si consegnano i documenti.
- 2) Un messaggio cifrato **vecchio** non deve essere confuso con uno **nuovo**

## Crittografia simmetrica e asimmetrica

Abbiamo un messaggio in chiave, un algoritmo di cifratura con una chiave, il testo cifrato, una chiave di decodifica e un algoritmo di decodifica. Un attaccante, con il testo cifrato può fare quello che vuole, anche farlo sparire (e questa



è una cosa che deve essere gestita). Quando si parla di crittografia si utilizza un certo formalismo matematico, che elencheremo a seguire:

- $E_k(m)$  = funzione di codifica su m con la chiave k
- $D_k(m)$  = funzione di decodifica su m con la chiave k
- $k_k$  = chiave di codifica dell'utente A
- $k_k^{-1}$  = chiave di decodifica dell'utente A

Parliamo di:

- Crittografia **simmetrica** se la chiave di cifratura e quella di decifratura coincidono  

$$m = D_k(E_k(m))$$
- Crittografia **asimmetrica** se la chiave di cifrature e di decifratura non coincidono  

$$m = D_k^{-1}(E_k(m))$$

Storicamente la prima ad essere usata è stata la crittografia simmetrica. Però normalmente ai giorni nostri si utilizzano sistemi asimmetrici, in quanto anche conoscendo una delle due chiavi è impossibile decifrare il messaggio.

Ovviamente qualunque sistema crittografico può essere violato con tentativi a **forza bruta**, ovvero tentando tutte le possibili chiavi. Se però la dimensione della chiave è elevata, tale tipo di attacco richiederebbe hardware troppo potente, per ottenere risultati in tempi ragionevoli, in quanto i tempi che si impiegano potrebbero essere anche infiniti. Immaginiamo un sistema di cifratura dove la chiave dura una settimana, l'attaccante ha quel tempo per bucare il sistema, ma solitamente un algoritmo di cifratura che si buca in un mese è molto scarso. Quello che rende un sistema di sicurezza, più "sicuro" di un altro è il tempo che si impiega per forzarlo, se un attaccante vede una macchina dove l'ordine di tempo per forzare la chiave è dell'ordine di mille anni, cambia obiettivo e magari si concentra su una macchina dove il tempo è di un anno.

## DES (Data Encryption Standard)

E' stato uno dei primi algoritmi di cifratura, scelto come standard dal Federal Information Processing Standard per il governo degli Stati Uniti d'America nel 1976 e in seguito diventato di utilizzo internazionale, l'idea originale era comunque della IBM.

Si basa su una chiave simmetrica a 56bit, che come dimensione è molto piccola; proprio la dimensione della chiave fece nascere numerose discussioni su questo algoritmo, in quanto in principio era stato pensato con una chiave a 64bit. Si supponeva che dietro queste scelte vi fosse la NSA (National Security Agency) e l'inserimento di una backdoor. Comunque ai tempi nostri viene considerata una schifezza, in quanto i tempi di bucaggio si aggirano sull'ordine delle ore; l'algoritmo si ritiene sicuro solo se applicato reiterandolo 3 volte nel **Triple DES** e utilizzando due chiavi separate, anche se è sempre esposto agli attacchi.

Il DES è l'archetipo della **cifratura a blocchi**. In crittologia un algoritmo di cifratura a blocchi è un algoritmo a chiave simmetrica operante su un gruppo di bit di lunghezza finita organizzati in un blocco. A differenza degli algoritmi a flusso che cifrano un singolo elemento alla volta, gli algoritmi a blocco cifrano un blocco di elementi contemporaneamente. L'algoritmo DES prende in ingresso una stringa di lunghezza fissa di testo in chiaro e la trasforma con una serie di operazioni complesse in un'altra stringa di testo cifrato della stessa lunghezza; in questo caso la dimensione del blocco è di 64bit, inoltre usa una chiave per modificare la trasformazione in modo che l'operazione di decifratura possa essere effettuata solo conoscendo la chiave stessa. La chiave è lunga 64bit, ma solo 56 di questi sono effettivamente utilizzati dall'algoritmo. Otto bit sono utilizzati solo per i controllo di parità e poi scartati, per questo la lunghezza della chiave effettiva è riportata come di 56bit.

Il DES non si buca solo con gli attacchi a forza bruta, in quanto può venire implementato in hardware, come nei chip delle carte di credito. In questi casi si può agire, in una maniera un po' macchinosa: poiché è tutto realizzato in hardware, che alla fine è un sistema di circuiteria, si può attaccare tagliando fisicamente un pin di quel circuito, dopodiché vedo che cosa accade, analizzando il crittotesto. Facendo un numero limitato di tentativi, si può ridurre la complessità dei tentativi ad ogni attacco, l'unico inconveniente è che il chip lo rovina.

Negli ultimi anni DES è stato sostituito da un altro algoritmo, molto più sicuro, chiamato **AES** (*Advanced Encryption Standard*), usato normalmente nella cifratura della cifra wifi. Molto diverso dallo schema DES e soprattutto usa chiavi da 128bit.

## RSA

Il sistema di crittografia si basa sull'esistenza di due chiavi distinte, che vengono usate per cifrare e decifrare. Se la prima chiave viene usata per la cifratura, la seconda deve necessariamente essere utilizzata per la decifratura e viceversa. La questione fondamentale è che nonostante le due chiavi siano fra loro dipendenti, non sia possibile risalire dall'una all'altra, in modo che se anche si è a conoscenza di una delle due chiavi, non si possa risalire all'altra, garantendo in questo modo l'integrità della crittografia.

Per poter realizzare con il cifrario asimmetrico un sistema crittografico pubblico è importante che un utente si crei autonomamente entrambe le chiavi, denominate **diretta** e **inversa**, e ne renda pubblica una soltanto. Così facendo si viene a creare una sorta di "elenco telefonico" a disposizione di tutti gli utenti, che raggruppa tutte le chiavi dirette, mentre quelle inverse saranno tenute segrete dagli utenti che le hanno create e da questi utilizzate solo quando ricevono un messaggio cifrato con la rispettiva chiave pubblica dell'elenco da parte di un certo mittente, ottenendo in questo modo i presupposti necessari alla sicurezza del sistema.

Nel 1978 questo sistema trova applicazione reale; tre ricercatori del MIT (Ronald Rivest, Adi Shamir e Leonard Adleman) hanno implementato tale logica utilizzando particolari proprietà formali dei numeri primi (fattorizzazione) con alcune centinaia di cifre. L'algoritmo da loro inventato, denominato RSA per via delle iniziali dei loro cognomi, non è sicuro da un punto di vista matematico teorico, in quanto esiste la possibilità che tramite la conoscenza della chiave pubblica si possa decrittare un messaggio, ma l'enorme mole di calcoli e l'enorme dispendio in termini di tempo necessario per trovare la soluzione, fa di questo algoritmo un sistema di affidabilità pressoché assoluta.

L'algoritmo ha le seguenti basi:

- Scegliere due numeri primi (molto grandi)  $p$  e  $q$
- Siano  $n = p \cdot q$  e  $z = (p-1) \cdot (q-1)$
- Scegliere un numero  $d$  che sia primo rispetto a  $z$
- Trovare  $e$  tale che  $e \cdot d = 1 \text{ mod } z$

Dividiamo il testo da cifrare in blocchi di dimensione  $m$ , tali che  $0 \leq m \leq n$ , ovvero raggruppiamo il testo in blocchi da  $k$  bit con  $2^k < n$ .

Per cifrare calcoliamo:  $C = m^e \text{ mod } n$

Per decifrare calcoliamo:  $D = C^d \text{ mod } n$

Inoltre si può dimostrare che  $m = C^d \text{ mod } n$  per qualunque  $m$

Facciamo un esempio:

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>• due numeri primi <math>p</math> e <math>q</math></li> <li>• <math>n = p \cdot q</math></li> <li>• <math>z = (p-1) \cdot (q-1)</math></li> <li>• <math>d</math> primo rispetto a <math>z</math></li> <li>• <math>e \cdot d = 1 \text{ mod } z</math></li> </ul> | $p=3, q=11$<br>$n = 33$<br>$z = 20$<br>$d = 7$<br>$e = 3$ |
|---|---|

Le due chiavi di cifratura sono 7 e 3, ora dobbiamo verificare che la proprietà prima citata sia verificata; per farlo possiamo utilizzare Excel facendo una tabella come la seguente:

Messaggio orig. $x$	$x^e$	$E = y = x^e \text{ mod } n$	$y^d$	$D = y^d \text{ mod } n$
9	729	3	2187	9

Se la proprietà è verificata, allora la prima e l'ultima colonna sono uguali.

La complessità dell'algoritmo è notevole, per cui la cifratura per messaggi molto grandi potrebbe essere molto dispendioso, ma otteniamo un mezzo di comunicazione abbastanza sicuro.

## Protocolli di Autenticazione

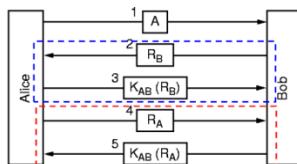
Un protocollo di autenticazione serve per avere la certezza sull'identità della controparte; basare l'autenticazione sull'indirizzo IP è da folli, invece basarlo su login e password è poco sicuro, in quanto l'avversario potrebbe aver registrato le credenziali e usarle per spacciare per qualcun altro.

Una soluzione potrebbe essere basata sulla firma, vediamo qualche cosa: l'utente A invia la richiesta di connessione all'utente B, che risponde ad A inviando un numero R random, che A dovrà restituire cifrato (utilizzando una chiave che A e B hanno in comune) a B. Il meccanismo funziona, ma si può fare di meglio, in quanto se l'avversario conosce la chiave, è un grosso danno e anche se non conosce la chiave potrebbe fare la seguente: l'utente anomalo T, chiede di connettersi a B spacciandosi per A, come prima B invia il numero R; T ovviamente non conosce la chiave tra A e B, ma magari conosce una chiave tra se e B, per cui se T riesce in qualche modo a dire a B di cifrare la R, B manda la R cifrata a T, fregando così l'intero sistema. Serve quindi un meccanismo più complicato, possiamo quindi basarci su un meccanismo di chiave pubblica e chiave privata; supponiamo che A e B debbano comunicare, A apre la comunicazione e B invia il solito numero R, A cifra questo numero R con la sua chiave privata e lo rispedisce a B, quest'ultimo invia una richiesta ad A, ricevuto il messaggio cifrato, in cui chiede la sua chiave pubblica, A la spedisce e B riesce a decifrare il messaggio correttamente, confermando che A è chi dice di essere. Il meccanismo di frode di questo sistema è semplicissimo, in quanto B vede solo messaggi cifrati e messaggi in chiaro e quello che interessa a lui è che la sua R venga cifrata e decifrata correttamente, per cui se T si vuole spacciare per A che deve fare? T apre la comunicazione spacciandosi per A, B invia il numero R che T cifra con la propria chiave privata e lo rispedisce a B, quest'ultimo a sua volta si fa spedire la chiave pubblica di T e decifra il messaggio, una volta che si rende conto che il risultato è corretto, autentica T come A. Il protocollo SSH, se non configurato correttamente, funziona proprio in questa maniera. Per configurare correttamente SSH si deve: generare la coppia di chiavi sul server, prendere la chiave pubblica fisicamente e me la porto dietro, ovvero non la faccio domandare da remoto. Per cui quando mi connetto con il server, la parte in cui domando la chiave pubblica la salto completamente; in questo modo sono assolutamente sicuro dell'identità della macchina con cui mi sto connettendo, in quanto la chiave l'ho generata e presa io fisicamente dal server.

Il problema principale delle chiavi pubbliche e segrete è che funziona solo se si ha la certezza dell'autenticità della chiave pubblica; per tale motivo di solito esiste un'autorità centrale, riconosciuta da tutti come fidata, che provvede alla distribuzione delle chiavi pubbliche, un cosiddetto **KDC** (*key distribution center*) che crea le coppie di chiavi delle entità che devono comunicare. Per fare un esempio: io ho necessità di entrare all'interno di un circuito di comunicazione gestito da un KDC, per cui mi faccio generare una coppia di chiavi dal KDC e me le faccio consegnare offline e la stessa entità pubblicherà all'interno del circuito di comunicazione, la mia chiave pubblica. Poiché avrò necessità di autenticarmi presso il KDC, per poter ricevere le chiavi pubbliche degli altri appartenenti al circuito di comunicazione, oltre alla mia coppia di chiavi, mi faccio consegnare anche la chiave pubblica dell'entità. Posso comunicare anche la mia coppia di chiavi, generata nella mia macchina, al KDC, però devo inserirle fisicamente nel KDC. La cosa fondamentale è che l'inserimento della coppia di chiavi sia fatto offline.

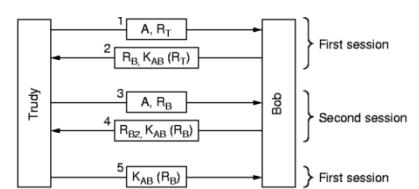
## Autenticazione con chiave segreta

Ipotizziamo che le due controparti condividano una chiave segreta  $K_{AB}$ , l'autenticazione avviene con un meccanismo



di sfida e risposta che funziona nella seguente maniera: A invia la sua richiesta di comunicazione, B risponde inviando il suo numero  $R_B$  che A dovrà cifrare con la chiave che hanno in comune e restituirlo a B, nello stesso momento A invia a B un suo numero  $R_A$  che B dovrà restituire ad A dopo averlo cifrato con la chiave in comune. Anche in questo caso un attaccante può utilizzare l'attacco riflesso per poter bucare il sistema, 5840facendo i seguenti passaggi. T si connette a B in una prima

questo caso un attaccante può utilizzare l'attacco riflesso per poter bucare il sistema, 5840facendo i seguenti passaggi. T si connette a B in una prima

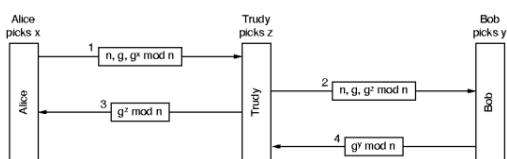


sessione, comunicando che si tratta di A e gli spedisce il suo numero  $R_T$ , B dal canto suo accetta questa connessione e risponde inviando il numero di T cifrato con la chiave  $K_{AB}$  (che T non sa interpretare) e il numero  $R_B$  che T dovrà cifrare (sempre con la chiave  $K_{AB}$ ) e restituire a B. T fatto questo apre una seconda comunicazione con B, spacciandosi sempre per A, ma questa volta gli spedisce il numero  $R_B$ , B cifra questo valore e lo restituisce a T, che lo utilizzerà nella prima sessione, restituendolo a B e spacciandolo per il numero appena cifrato da lei, con la chiave  $K_{AB}$ . B è contento e fornisce i privilegi di accesso a T. Questo è un esempio di crittografia senza sicurezza.

## Scambio di chiavi Diffie-Hellman

E' un protocollo crittografico che consente a due entità di stabilire una chiave condivisa segreta utilizzando un canale di comunicazione insicuro senza la necessità che le due parti si siano scambiate informazioni o si siano incontrate in precedenza. La chiave ottenuta mediante questo protocollo può essere successivamente impiegata per cifrare le comunicazioni successive tramite uno schema di crittografia simmetrica.

Si considera inizialmente due numeri  $g$  e  $p$  dove quest'ultimo è un numero primo. Uno dei due interlocutori, ad esempio Alice, sceglie un numero casuale  $a$  e calcola il valore  $A = g^a \text{ mod } p$  e lo invia attraverso il canale pubblico a Bob, assieme ai valori  $g$  e  $p$ . Bob da parte sua sceglie un numero casuale  $b$ , calcola  $B = g^b \text{ mod } p$  e lo invia ad Alice. A questo punto Alice calcola  $K_A = B^a \text{ mod } p$ , mentre Bob calcola  $K_B = A^b \text{ mod } p$ . I valori calcolati sono gli stessi, in quanto  $B^a = g^{ba}$  e  $A^b = g^{ab}$ . A questo punto i due interlocutori sono entrambi in possesso della chiave segreta e possono cominciare ad usarla per cifrare le comunicazioni successive. Un attaccante può benissimo ascoltare tutto lo scambio, ma per calcolare i valori  $a$  e  $b$  avrebbe bisogno di risolvere l'operazione del logaritmo discreto, che è computazionalmente onerosa e richiede parecchio tempo.



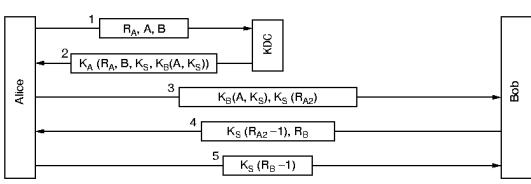
Purtroppo l'algoritmo è vulnerabile all'attacco **Man in the middle**, durante il quale un agente terzo può falsificare le chiavi pubbliche di Alice e Bob ed ingannare le due parti (ovvero si mette in mezzo, facendo da intermediario malevolo). L'algoritmo infatti attua lo scambio delle chiavi simmetriche segrete, ma con il presupposto di avere delle informazioni pubbliche condivise e si dimostra resistente nei confronti dell'intercettamento di queste ultime. Ma nulla può impedire che le informazioni pubbliche siano state modificate o falsificate; in questo caso Alice e Bob non avrebbero modo di accorgersi della frode appoggiandosi al solo algoritmo Diffie-Hellman. Ecco perché occorre che le chiavi pubbliche, possano essere autenticate tramite un algoritmo di autenticazione o da un Certification Authority.

## KDC (Key Distribution Center)

Per evitare di memorizzare  $n$  chiavi segrete, è possibile utilizzare un **Centro di Distribuzioni di Chiavi**, dove vengono memorizzate tutte le chiavi segrete degli utenti.

Alice crea un messaggio per KDC cifrato con la chiave segreta che condivide con KDC, inoltre manda anche l'identificativo della controparte (Bob) e una chiave di sessione ( $K_S$ ). KDC poi contatta B. Questa procedura di autenticazione può fallire se l'intruso duplica il messaggio 2 e i successivi messaggi spediti da Alice a Bob. Bob infatti non può sapere con certezza se il messaggio che riceve viene da KDC o è una copia vecchia. Tali attacchi vengono eliminati con l'uso di **nonce** (number used once).

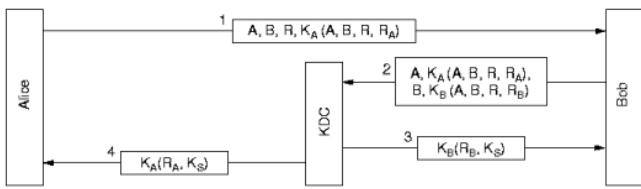
Le nonce vengono usate per protocolli sfida-risposta. Un protocollo è il **Needham-Schroeder**, che funziona nella seguente maniera:



Alice spedisce al KDC un messaggio contenente la sua identità, quella di Bob ed un numero  $R_A$  casuale. Il server genera la chiave di sessione  $K_S$  e risponde ad Alice inviandole: la chiave di sessione appena generata, la coppia  $(A, K_S)$  criptata con la chiave  $K_B$  (ovvero la chiave conosciuta tra KDC e B) in modo che possa essere inoltrata a Bob per essere reso partecipe, il nonce  $R_A$  e l'identificativo di B, il tutto criptato con la chiave  $K_A$ . Alice comunica a

Bob la chiave di sessione e il proprio identificativo, criptati con la chiave  $K_B$ , come comunicata dal server con il precedente messaggio, inoltre invia il suo nonce criptato con la chiave di sessione. Bob può decriptare il messaggio e il fatto che sia stato cifrato da un entità fidata (KDC) lo rende autentico e rispedisce ad Alice il nonce di alice – 1 criptato con la chiave di sessione ed il proprio nonce  $R_B$ . Alice rispedisce a Bob  $R_B$  – 1 criptato con la chiave di sessione. La restituzione di  $R_{A2} - 1$  e  $R_B - 1$  serve ad evitare che  $K_S(R_{A2})$  venga rubata da un intruso e rispedita come vera.

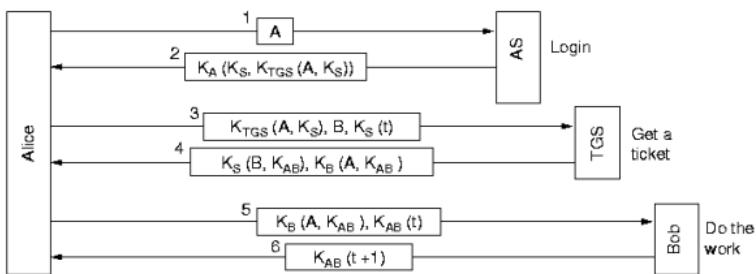
Tuttavia se un intruso ottiene una vecchia chiave di sessione col testo in chiaro, può violare il protocollo, sempre con il solito attacco men in the middle. Una variante del protocollo precedente che risolve il problema è dovuta a **Otway e Rees**:



Dove R serve affinché KDC possa essere sicuro che le due parti del messaggio di B siano accoppiate e non costruite artificialmente da un intruso.  $K_S$  viene usata solo per quella sessione ed una volta terminata quella comunicazione viene eliminata, inoltre  $K_A$  e  $K_B$  vengono usate il meno possibile.

## Autenticazione in Kerberos

Kerberos è un protocollo di rete per l'autenticazione tramite crittografia che permette a diversi terminali di comunicare su una rete informatica insicura provando la propria identità e cifrando i dati. Kerberos previene attacchi quali l'intercettazione e i replay attack ed assicura l'integrità dei dati. I suoi progettisti mirarono soprattutto ad un modello client-server, e fornisce una **mutua autenticazione** cioè sia l'utente che il fornitore del servizio possono verificare l'identità dell'altro.



Kerberos si basa sul protocollo di Needham-Schroeder. Utilizza una terza parte affidabile per centralizzare la distribuzione delle chiavi detta Key Distribution Center (KDC), che consiste di due parti separate logicamente: **l'Authentication Server (AS)** e il **Ticket Granting Server (TGS)**. Kerberos funziona utilizzando dei "biglietti" (detti ticket) che servono per provare l'identità degli utenti. L'AS mantiene un database delle chiavi segrete; ogni entità sulla rete — che sia un client o un server — condivide la chiave segreta solo con l'AS. La conoscenza di questa chiave serve per provare l'identità di un'entità. Per comunicazioni tra due entità, Kerberos genera una chiave di sessione, che può essere utilizzata dai due terminali per comunicare.

Kerberos funziona su questa filosofia: autenticazione centrale e dopo che mi sono autenticato, posso accedere ad una famiglia di risorse legate a quei permessi che ho ottenuto. C'è una prima fase dove l'utente si autentica, nella seconda l'utente riceve un diritto di accesso e, mediante il sistema a Ticket, la macchina che ha richiesto l'accesso ha diritto all'utilizzo di una data risorsa solo per un intervallo di tempo limitato; una volta che viene fornito il ticket, viene passato a Bob, che si rende conto che Alice vuole utilizzarlo e che ne ha il diritto.

## La firma digitale

Il documento viene cifrato con la chiave privata e tramite la chiave pubblica tutti lo possono leggere, la chiave se voglio posso anche appenderla alla fine del messaggio. Un possibile attaccante non può modificare il documento e farlo passare per mio, in quanto anche se riesce a leggerlo, non può ri-firmarlo con la mia chiave privata. Se il documento è molto lungo, invece di andare a firmare tutto il documento, tramite una funzione hash tiro fuori un pezzo più piccolo e firmo quello. Funziona in quanto, se un attaccante modifica il documento senza modificare la firma, ma l'hash risultante del documento modificato sarà completamente differente da quello originale.

Anche qui comunque è necessaria un'entità che certifichi che quella firma sia proprio la mia. Mentre facevo il responsabile di Galileo, utilizzavo gpg per firmare la posta elettronica che mi scambiavo con l'altro responsabile, le chiavi ovviamente ce le siamo scambiate di presenza.