1

Ingranaggi di Internet

-**Host**/Sistemi periferici: dispositivi connessi alla rete. Ospitano le applicazioni di rete, quindi queste ultime risiedono alla periferia della rete (edge).
Lo scopo di Internet è interconnettere gli host. Degli host fanno parte anche i server, su cui sono in esecuzione i programmi che erogano servizi.
-Verso l'interno abbiamo i **Commutatori di pacchetto** che si occupano di inoltrare i pacchetti (date n linee di ingresso e di uscita) da una linea all'altra.
Router se a livello di rete
Switch se a livello di collegamento
-Tra i commutatori e host vi sono **linee di comunicazione** (livello fisico): fibra ottica, rame, segnali radio, satellite… ciò che li differenza è la larghezza di banda (velocità di trasmissione)
.-**Reti**: con il concetto di rete si intende una collezione di host, commutatori e linee di comunicazione gestiti da una singola organizzazione (rete mobile, domestica, aziendale)
Internet rappresenta infatti una rete di reti, non è gestita da un'unica entità.

Le entità che consentono l'accesso ad internet sono gli Internet Service Provider **ISP**. Poiché gli ISP coprono soltanto un certo raggio, anche in base al livello gerarchico di cui fanno parte, per garantire copertura quasi globale gli ISP sono interconnessi tra loro.

Reti di distribuzione di contenuti sono reti private di grandi conduttori di servizi (es. Google) che cercano di aggirare la rete pubblica per arrivare il più vicino possibile ai propri utenti.

Quando due entità in Internet dialogano il **dialogo è regolato da protocolli** (es. protocollo IP per inoltrare pacchetti attraverso la rete). Per far sì che vi sia comprensione reciproca tra dispositivi i protocolli sono standardizzati dalla **IETF** e gli standard prendono il nome di **RFC**.

Non tanto rete di calcolatori quanto **Internet of Things**.

Le applicazioni di rete si appoggiano ad Internet in quanto quest'ultimo permette che possano usufruire di diversi servizi. Il servizio principale offerto da Internet ai processi in esecuzione negli host è quello di un'interfaccia di programmazione, detta **Interfaccia Socket**, che permette di interconnettere tra loro processi remoti permettendo il "trasporto" di informazioni da uno all'altro.

Un **Protocollo** definisce il **formato e l'ordine dei messaggi scambiati** tra due o più entità e le **azioni intraprese** in fase di *trasmissione* e/o *ricezione* e/o di *un altro evento*.
La struttura in cui è definito il messaggio deve essere nota a priori.

Server == Erogano Servizi (nei Data Center, che contengono cluster etc…)
Client == Richiedono Servizi

Cloud Computing == invece di dover acquistare e mantenere data server e server fisici, è possibile accedere ai servizi che mi fornirebbe un server fisico (capacità di calcolo, archiviazione di dati, database) senza che lo possegga tramite un fornitore cloud. Pago quanto mi è necessario usare.

Reti di accesso e mezzi trasmissivi

Una **Rete di accesso** rappresenta la rete che collega gli host periferici al loro edge router, ossia al primo router nel cammino che porta ad host non appartenenti alla mia rete di accesso. Si utilizza per permettere la comunicazione tra host nella mia rete di accesso e host che non ne fanno parte.

I parametri di riferimento quando si parla di reti di accesso sono il *tasso di trasmissione* e se *l'accesso è dedicato o condiviso*.

Accesso via cavo: (americano) si ha una **stazione di testa** da cui parte un cavo coassiale attraverso il quale passano segnali elettrici. Questi segnali sono divisi in termini di frequenza secondo un Multiplexing a Divisione di Frequenza **FDM** per cui frequenze diverse sono dedicate a canali diversi. Infatti in ogni abitazione è presente uno **splitter** che si occupa di inviare in base alla frequenza i segnali ai dispositivi corrispettivi (tv, modem etc…).
Un **Modem** (modulator/demodulator) è un dispositivo che si occupa di *modulare i segnali digitali 0/1* in segnali analogici elettrici e *demodulare i segnali analogici elettrici* in segnali digitali 0/1.

Esiste una **soluzione ibrida HFC** (hybrid fiber coax) per cui a partire dalla stazione di testa si utilizza un cavo in fibra ottica fino a raggiungere nodi distrettuali più vicini alle abitazioni. In questo modo si velocizzano le cose.
Nella stazione di testa si ha un CMTS da cui partono i vari cavi.

Accesso condiviso == **asimmetria** *Downstream* (dai 40 Mbps agli 1.2 Gbps) e *Upstream* (dai 30 Mbps ai 100 Mbps).
Poiché l'accesso è condiviso se scarico un file questo in realtà arriva a tutte le abitazioni (protetto in termini di privacy da crittografia).

DSL: L'ISP non è più l'operatore della TV via cavo ma **l'operatore telefonico**. Viene *suddivisa la banda sul doppino telefonico in più canali in termini di frequenza*: uno dedicato alla **telefonia**, un altro al **Downstream** e un altro ancora all'**Upstream**. Si avrà quindi anche in questo caso per ogni abitazione uno **splitter** che si occupa di inviare i dati provenienti da un certo canale al dispositivo corretto (**telefono o modem**).

A differenza dell'accesso via cavo invece del CMTS il dispositivo che raccoglie i fili è il **DSLAM**, mentre con CMTS si aveva una linea condivisa si ha in questo caso una **linea dedicata**.
La velocità può dipendere da molti fattori come limitazioni del provider, distanza, qualità del materiale e interferenze.
Grazie al DSLAM i dati provenienti dal modem vanno su Internet, la voce va nella rete telefonica.

Fibra ottica FTTX: FTTH fiber to the home 1 Giga download, FTTB fiber to the building, FTTC o FTTS fiber to the cabinet /street al più 300 metri di distanza dalla fibra il resto del collegamento avviene via tecnologia DSL con doppini in rame, FTTN fiber to the node distanza maggiore di 300 metri, FTTW o FTTR fiber to the wireless/radio la fibra arriva fino alla stazione radio e poi da lì wireless. Minore distanza dal collegamento in fibra, maggiore velocità.

Per la **FTTH** esistono due approcci: **AON** Active Optical Network per cui vi sono una serie di ripetitori con trasmettitori e recettori ottici fino a casa (ethernet commutate) e **PON** Passive Optical Network per cui vengono utilizzati splitter ottici per inviare direttamente il segnale in broadcast ai vari utenti.

Con **PON** si ha un **OLT** nella centrale locale a cui arriva la fibra ottica. L'OLT trasforma il segnale ottico in dati numerici che vengono immessi in un router che invia in broadcast a più abitazioni il messaggio in arrivo. Ancora una volta quindi privacy in download protetta da crittografia.

Rete di accesso FWA (Fixed Wireless Access, mista fibra radio) utilizzata nel caso in cui non vi sono mezzi per estendere il collegamento a determinate abitazioni. Si utilizza la tecnologia wireless.

Quindi fibra e cavo condivisi, DLS dedicata ma con velocità compromessa da molti fattori.

Nelle nostre case non abbiamo solo il modem, ma una rete domestica che interconnette gli host presenti tramite filo (ethernet) o via wireless. Si parla di **Reti Locali Wireless**, costituiti dai tre componenti ***Modem***, ***Router*** e ***WAP Access Point Wireless*** combinate nell'unico dispositivo **WLAN** (collegato alla fibra o al doppino nel caso della DSL, supporta wi-fi e presenta porte ethernet per collegare via cavo).

Accesso Wireless a corto raggio == principale supporto WiFi

Accesso Wireless su scala geografica (decine di km) == principale supporto 4G e 5G

Rete di accesso Aziendale: per aziende, università etc… non si hanno solo i router per l'inoltro di pacchetti fuori dalla rete, i modem e gli host ma anche degli switch che operano a livello di collegamento per interconnettere rapidamente gli host della rete. Tipicamente connessioni via ethernet (più rapido di DSL).

Rete di accesso Data Center: Migliaia di computer interconnessi a formare cluster connessi ad internet. Rete interna molto veloce, poi rete che connette i data center a Internet. (data center per server == erogazione di servizi)

Invio pacchetti

Le applicazioni comunicano tramite scambio di messaggi. Un'applicazione per mandare il messaggio usa l'interfaccia socket del proprio host.
L'implementazione della pila protocollare di Internet nell'host si occuperà di trasmettere il messaggio seguendo un percorso in Internet.
Quando l'host invia un messaggio lo suddivide in frammenti più piccoli, detti **pacchetti**, di lunghezza **L bit**.
La velocità con cui il pacchetto viene trasmesso è detta **tasso di trasmissione** (in bit/sec). Dati R ed L, il **ritardo di trasmissione del pacchetto** è **L/R**.

Ritardo di trasmissione del pacchetto == tempo necessario a trasmettere il pacchetto di L bit nel collegamento di banda R.

Il mezzo attraverso il quale passano i bit da host mittente a destinatario può essere vincolato (es. cavi) o non vincolato (es. radio, il segnale viaggia libero).
Il canale da mittente a destinatario può essere implementato con diversi mezzi, l'esempio più tipico è il **doppino intrecciato** costituito da due fili di rame intrecciati tra loro (intreccio per evitare interferenze, materiale schermante)
Poi **cavo coassiale** == due conduttori di rame concentrici con uno strato schermante.

Fibra Ottica == passa per un materiale flessibile e sottile che conduce impulsi di luce, ciasucno dei quali rappresenta un bit. Supporta elevatissime velocità, bassa attenuazione del segnale per lunghe distanze (pochi ripetitori) e segnale di luce totalmente immune alle interferenze elettromagnetiche. Tratti oceanici.

Canali Radio == mezzi broadcast, si deve adottare uno schema half duplex (non posso trasmettere mentre sto ricevendo). L'ambiente può interferire con il segnale, necessari protocolli per la gestione di ciò. Esistono:
WLAN (*WiFi*), Wide Area (*4G e 5G*), *Bluetooth*, *Microonde terrestri*, *Satellitari*.

Per i satellitari bassa quota (**LEO**), media quota (**MEO**), geostazionari (**GEO**). *Vantaggio di GEO è che sono abbastanza distanti da poter restare fermi* (ruotano sincroni alla rotazione terrestre) *per cui le antenne a terra non devono muoversi per seguirli, svantaggio è la distanza* (più tempo a propagare il segnale, 270 ms di ritardo, la mente umana percepisce qualcosa come istantaneo non sopra i 100 ms).
Satelliti più bassi propagazione più veloce, ma più bassi sono più si muovono velocemente della rotazione terrestre e quindi più complesso seguirli con le antenne.

2.

Nucleo della rete

Maglia di collegamenti che ha come fine quello di connettere tra loro gli host. Internet usa a tale scopo una tecnologia chiamata **packet switching** (**commutazione di pacchetto**). Il nucleo ha due funzioni chiave:
-**Funzione di inoltro** (switching) locale a ciascun router che sposta il pacchetto di ingresso a quello di uscita appropriato. Per farlo: intestazione pacchetto con indirizzo di destinazione – confronto con tabella di inoltro – capisce qual è la linea di uscita appropriata.
-**Funzione di instradamento**: azione globale che determina il contenuto delle tabelle di inoltro dei vari router (sarebbe impensabile una tabella di inoltro con tutti i miliardi possibili indirizzi di destinazione). I percorsi sono determinati tramite algoritmi di instradamento.

Commutazione di pacchetto

Commutazione di pacchetto == tecnica dello store and forward, per cui un router deve ricevere completamente un pacchetto prima di poterlo inoltrare. Questa tecnica è vantaggiosa quando si vogliono inviare più pacchetti attraverso N collegamenti.

Se non vi fossero gli N collegamenti e volessi inviare 3 pacchetti, impiegheremmo tempo 3L/R (ritardo di trasmissione classico).
Se avessi solo un router nel mezzo impiegherei tempo in store-forward 4L/R (invio il primo pacchetto al router, fase successiva invio il secondo ma intanto il router ha inviato a destinazione il primo e così via).

Quindi *ritardo end-to-end (sorgente-destinazione) per la trasmissione di un singolo pacchetto su un percorso di N collegamenti* è
d = N L/R sec
Ma se invece stessi mandando *P pacchetti* è
d = (N + P – 1) L/R sec

Ritardo di accodamento

Si forma **accodamento** quando **arrivano più pacchetti al router di quanti ne possano essere inviati**. I pacchetti in eccesso vengono posti in un buffer in attesa di essere inviati. Poiché non ho memoria infinita se per un periodo abbastanza lungo arrivano più dati di quanti ne possa trasmettere perdo dati (**overflow**).

Un'alternativa alla commutazione di pacchetto (store and forward) è la **commutazione a circuito** per cui buffer e velocità di trasmissione sono riservate tra i due host nell'arco dell'intera comunicazione.
Vantaggi == **risorse dedicate** -> velocità di trasferimento costante e garantita (utile per real time, utilizzata infatti nella rete telefonica tradizionale)
Svantaggi == se i due sistemi non comunicano **la risorsa** a loro riservata **risulta sprecata**.

Due tipi di commutazione a circuito: **FDM** e **TDM**. (time e freq. division mult.)
Con la **FDM** invio contemporaneamente segnali di frequenze diverse e specifiche ottenendo un segnale di frequenza ben specifica che posso riconoscere per dedurre quali frequenze sono state originariamente inviate. Chiaramente poiché limito la larghezza di banda di ogni host mittente è ridotta anche la velocità a cui possono inviare dati.

Con **TDM** il canale è diviso in termini di tempo in elementi di durata fissa, **frame**. Divido il frame in n slot temporali, che posso assegnare periodicamente a ciascun utente. Ogni utente userà il canale alla massima velocità in quello slot, ma non potrà utilizzarlo nel tempo che non gli compete.

Limite della commutazione a circuito è il fatto che un utente che non utilizza il canale comporta spreco delle risorse. La commutazione di pacchetto evita il problema perché non riservo in anticipo le risorse.
Limite della commutazione di pacchetto è che se trovo qualcun altro in trasmissione non posso trasmettere e mi tocca attendere. Infatti non riservando le risorse a priori c'è il rischio di non trovarle quando mi servono (attesa nel buffer, congestione e perdita di dati)

Nel caso del traffico in rete conviene sempre usare la commutazione di pacchetto.

Gerarchia ISP

Non è scalabile la soluzione di collegare tutti gli ISP a tutti gli ISP (O(n^2)).
L'idea è quindi di dividere gli ISP in gerarchie; vi sono **ISP globali** a cui sono connessi ISP più piccoli. ISP diversi possono inoltre connettersi tramite **IXP** (**Internet Exchange Point**), edifici dedicati all'incontro di ISP diversi.
Un altro approccio più diretto è il **peering link**: due ISP si connettono tra loro con un link diretto garantendo uno scambio a costo zero.

Poiché ISP globali non riescono a coprire perfettamente ogni singola regione o nazione, esistono ISP di gerarchia inferiore: **nazionali** e **regionali**.
In generale vale la regola che per il collegamento tra ISP di pari gerarchia nun se paga mentre per ISP di gerarchie diverse paga quello di gerarchia inferiore.

Un ruolo importante in ciò lo giocano anche i **fornitori di servizi** (es. Google), che oltre ad avere datacenter hanno reti globali private che collegano i vari datacenter sparsi nel mondo. Per evitare accordi costosi con gli ISP Tier 1 questi fornitori si accordano con gli ISP di livello più basso, che però hanno collegamento diretto con le varie reti di accesso. In questo modo il servizio erogato è migliore per gli utenti (minore distanza dall'utente).

Chiaramente anche sti fornitori di servizi possono usare gli IXP.

Ritardi e Perdite

Non solo ritardo di trasmissione, vi sono altri fattori da considerare. Anzitutto quando un pacchetto è pronto per essere trasmesso (è arrivato nella sua interezza secondo la tecnica store-forward) vi è un **ritardo di elaborazione** da parte del router per eseguire la funzione di inoltro (leggere il pacchetto, accedere alla tabella etc…). Questo ritardo è nell'ordine dei microsecondi.

Si ha poi il **ritardo di accodamento** che è il ritardo generato dal fatto che un pacchetto che deve essere trasmesso in una certa linea di uscita non può perché ce ne sta già un altro o più ad attendere di essere trasmesso -> il pacchetto va nel buffer. Dipende dal livello di congestione del router (casuale).

Il **ritardo di trasmissione** è quello che dipende dalla formula L/R.

Il **ritardo di propagazione** è il corrispettivo del ritardo di propagazione ma in senso puramente fisico, legato alla lunghezza d del collegamento fisico e alla velocità di propagazione del segnale v (tipicamente velocità della luce 2x10^8)
dprop = d/v

Il **ritardo end to end** rappresenta l'accumulo di ritardi per ogni nodo lungo il percorso sorgente-destinazione per l'invio del pacchetto. Si ha:
dend-to-end = sum-perogni-i (delab-i + dacc-i + dtrasm-i + dprop-i)

Ogni ritardo è misurato in ms.

Definiamo come **intensità di traffico** la formula **L a / R**, dove L lunghezza del pacchetto in bit, R larghezza di banda e *a la velocità media di arrivo dei pacchetti*.

- *Se La/R è si avvicina a zero allora il ritardo medio di accodamento è piccolo*
- *Se La/R si avvicina ad 1 allora ho un ritardo medio di accodamento grande*
- *Se La/R > 1 allora ho più "lavoro" di quanto possa essere servito (arrivano più pacchetti di quanti ne possa mandare) e quindi il ritardo tende a infinito! Posso avere perdite!*

In definitiva, non dobbiamo avere intensità di traffico maggiore di 1 (assicurate perdite), ma neanche troppo vicine a 1 (sotto 0.4). È necessario tenersi bassi poiché man mano che mi avvicino ad 1 cresco esponenzialmente e se raggiungo 1 vado all'infinito.

Traceroute comando per calcolare questi ritardi.
Si inviano 3 pacchetti per fare la media che raggiungeranno il router i-esimo sul percorso verso la destinazione. Il router i rimanda i pacchetti al mittente che calcola l'intervallo tra trasmissione e risposta.

Perdita di pacchetti

Ogni linea di uscita del router ha tipicamente un buffer. Chiaramente il buffer non ha capacità infinita, se il pacchetto trova coda piena viene scartato (perdita di dati). Il pacchetto può quindi essere ritrasmesso dall'host mittente oppure no. Con l'intensità di traffico quindi si fa riferimento al buffer in una specifica linea.

Le prestazioni delle reti sono quindi date da tre parametri principali: **Ritardi**, **Perdite** e **Throughput**.

Il Throughput rappresenta la frequenza di arrivo dei bit. Posso essere interessato a **frequenza istantanea** o **frequenza media** che calcolo su intervalli più lunghi.
Sia Rs larghezza di banda del collegamento dalla sorgente al router, Rc // // dal router alla destinazione.
Se invio informazioni con velocità Rs minore di Rc la media con cui il client riceverà le informazioni tende alla velocità minore Rs. Valo lo stesso se Rs > Rc, il throughput medio sarà Rc.

Quindi il throughput end to end dipende dalla velocità di trasmissione dei collegamenti attraversati dal flusso di dati.

Immaginando di avere un percorso con più collegamenti *il throughput medio sarà pari alla velocità di trasmissione del collegamento più scarso* -> **collo di bottiglia**.

Tuttavia in Internet si deve prendere in considerazione anche la possibilità che server e client diversi comunichino attraverso lo stesso collegamento.
Immaginiamo di avere 10 server e 10 client che comunicano e passano per lo stesso collegamento di velocità di trasmissione R (bit/sec). Allora il throughput end-to-end in questo caso sarò dato, per ciascuna connessione, da ***min(Rc, Rs, R/10)*** dove R/10 perché ho 10 connessioni in comune per lo stesso mezzo di velocità di trasmissione R!

Poiché solitamente R molto molto grande, i colli di bottiglia sono tendenzialmente dettati da client o server.

3

Differenza tra reti di accesso aziendali e domestiche: presenza di switch. Altra differenza è la presenza di commutatori di pacchetto che collegano agli ISP più veloci del normale.

Sicurezza di rete

Internet non era agli albori studiato per la sicurezza (pochi utenti lo utilizzavano).
Packet Sniffing: wireless, accesso via cavo e ethernet condivisa portano i pacchetti in broadcast a diversi utenti. È possibile che un malintenzionato via software packet sniffer possa registrare tutti i pacchetti in arrivo senza scartare quelli che non sono a lui destinati.
Attacco difficile da individuare perché passivo (non immette pacchetti in rete)
IP spoofing: Il malintenzionato usa un IP falso per inviare pacchetti in rete. Lo si fa per tre ragioni: difficile rintracciare la sorgente dell'attacco, può accedere a comunicazioni ristrette solo a certi utenti e può architettare attacco DoS.
DoS == l'aggressore rende una rete, host o altro elemento infrastrutturale non disponibile agli utenti legittimi. Ne esistono tre tipi:
-**Attacchi alla vulnerabilità di sistemi**: il malintenzionato, conoscendo a priori le vulnerabilità delle applicazioni/sistema operativo dell'host che sta attaccando invia pacchetti specifici per comportare blocco di servizio o spegnimento dell'host in questione
-**Bandwidth flooding**: il malintenzionato inonda di pacchetti l'host obiettivo, rendendolo inaccessibile agli utenti legittimi. (possibile se la larghezza di banda da cui l'host manda i pacchetti è vicina a quella per cui la vittima li riceve)
-**Connection flooding**: il malintenzionato stabilisce un enorme numero di connessioni TCP con la vittima, bloccando la possibilità di stabilirne con utenti legittimi.

Poiché con Bandwidth flooding è difficile che un singolo malintenzionato abbia larghezza di banda vicina a quella di arrivo del server attaccato, e anche se fosse sarebbe facile da individuare l'attaccante, si attua tipicamente un attacco **DDoS**. Si tratta di un attacco DoS distribuito per cui i malintenzionati, dopo aver inserito in qualche modo all'interno di moltissimi host che non gli appartengono un malware, iniziano l'attacco inviando pacchetti a profusione da tutti questi host. Infatti il malware inserisce gli host in una **botnet** (rete di macchine zombie) che possono inviare pacchetti a comando.
In questo modo si evita di essere rintracciati e si riesce a bombardare il server.

Linee di Difesa: **Autenticazione** == dimostro che sono effettivamente chi dichiaro di essere (nella rete cellulare SIM hardware, in Internet classico niente hardware che possa garantirlo); **Integrità** == oltre a dimostrare chi dico di essere faccio capire al destinatario se il pacchetto che ho inviato è stato manomesso o contraffatto; **Riservatezza** == per evitare il packet sniffing pacchetti arrivati ad altra gente sono cifrati e non possono leggerli senza chiave di decriptazione; **Restrizioni di accesso** == impedisco l'accesso a chi non autorizzato tramite credenziali; **Firewall** == *middlebox* per filtrare il traffico, oppure le implemento nel router/client. *Off by default* limito in generale pacchetti in entrata, rilevo/reagisco ad attacchi DoS, proteggo da IP spoofing.

Un approccio per comprendere l'architettura di internet è quello della **Pila Protocollare**, la Pila è composta da diversi livelli, ognuno caratterizzato da **protocolli specifici dedicati a quel livello**. **Ogni livello si occupa di garantire un servizio specifico**, compiendo certe azioni, e al fine di completare il servizio si affida al **livello a lui immediatamente inferiore**. Ciò garantisce *manutezione e aggiornamento del sistema più semplici* poiché posso intervenire sull'implementazione di un livello dintanto che ciò non comporta cambiamenti al servizio offerto al livello superiore.

In internet si hanno **5 livelli**, dall'alto verso il basso:

- **Applicazione (application layer)**: ospita le applicazioni di rete, per cui questo livello contiene i protocolli per cui i processi applicativi dialogano tra di loro (HTTP, IMAP, SMTP, DNS etc…);
- **Trasporto (transport layer)**: si occupa del trasferimento dati tra processi (in esecuzione su host differenti). Il livello di trasporto di internet fornisce alle applicazioni un'interfaccia (interfaccia socket) che permette a un processo in un host di comunicare con un altro. (TCP, UDP, trasferimento affidabile a connessione e inaffidabile senza connessione)
- **Rete (network layer)**: trasferimento di pacchetti di rete (detti datagrammi) da un host a un altro (IP, protocolli di instradamento per determinare le tabelle di instradamento)
- **Collegamento (link layer)**: trasferimento di dati tra elementi di rete vicini. La differenza con il livello di rete è che in quel caso avevo i due host ben definiti e nel mezzo tutto internet, il livello di collegamento è limitato ad elementi vicini (Ethernet, PPP, 802.11 (WiFi))
- **Fisico (physical layer)** si occupa dell'effettiva trasmissione del bit sul "filo" generando e/o interpretando i segnali fisici.

Funzionamento generale

Il livello applicativo vuole mandare un messaggio M al fine di far comunicare due processi applicativi tra host remoti. Per soddisfare il servizio deve far riferimento al livello immediatamente inferiore, trasporto, per cui M è incapsulato in un segmento tramite l'aggiunta di un header necessario a soddisfare il servizio del livello di trasporto (trasferimento dati tra host remoti). Per soddisfare il servizio serve affidarsi al livello ancora inferiore, rete, header che porta alla formazione di un datagramma, ancora una volta livello inferiore collegamento, header e frame. Poi il livello di collegamento fa riferimento al livello fisico che invia il frame, poi decapsulato in base al dispositivo a cui arriva.

In particolare inizialmente il messaggio è incapsulato man mano fino al frame che viene sparato dalla scheda di rete dell'host a uno switch fino al router, che lavora a livello di rete e quindi decapsula fino a livello di rete e grazie alle informazioni nell'header del datagramma e alla sua tabella di inoltro capisce dove inoltrare il pacchetto. Reincapsula tutto e manda e così via.
Non tutti i dispositivi in rete implementano tutti i livelli:
Hub fisico, Switch fino collegamento, Router fino rete, host fino applicazione.

Messaggi, segmenti, datagrammi e frame sono detti **PDU** (protocol data unit), tipicamente formati dall'intestazioni con informazioni specifiche sul protocollo utilizzato per quel livello e un payload (livelli superiori). L'intestazione può cambiare man mano che mando il PDU (es timetolive).

Modello ISO/OSI

Prevede due livelli in più: **presentazione** e **sessione**. Presentazione si occupa dell'interpretazione dei dati (crittografia, compressione etc…)
Sessione della sincronizzazione, checkpointing, ripristino dello scambio di dati.
Questi servizi vengono implementati dalle applicazioni se necessari.

4.

Livello applicativo

Ospita le applicazioni di rete (moltissime, es. streaming video (yt, netflix), videogiochi online, telefonia via internet, posta elettronica), poiché l'obiettivo del livello applicativo è quello di scambiare messaggi applicativi tra processi applicativi in esecuzione su host remoti, *scrivere applicazioni di rete non riguarda affatto il nucleo della rete ma solo gli host (periferia)*. Scrivere un'applicazione di rete **non riguarda un singolo sistema periferico ma più**, per comunicare tra loro si fa affidamento al livello di trasporto.

Per quel che riguarda lo schema di comunicazione esistono due principali paradigmi per architettare le applicazioni di rete: **peertopeer** e **client-server**.

Client-Server

Abbiamo un server (spesso presente in DataCenter e replicato per motivi di scalabilità in cluster) che eroga i servizi. Per questo motivo deve avere **IP fisso**, *mentre i client che richiedono i servizi possono avere IP dinamico*. **Due client diversi non possono comunicare direttamente** tra loro, devono prima passare per il server. Inoltre tendenzialmente lo schema di comunicazione è del tipo **richiesta-risposta**, ma esiste anche un altro schema di tipo **Push** per cui *il server può inviare al client che si è registrato informazioni in modo proattivo*.

Peer to peer

Più host comunicano tra di loro chiedendo e fornendo servizi (per questo peer). **Scalabilità intrinseca**, se aggiungo un host questo aggiunge carico al sistema (chiede) ma allo stesso tempo aggiunge capacità di servizio (dà).
I peer non sono fissi e per questo possono anche avere **IP dinamici**, ciò comporta una certa difficoltà nel gestire il tutto.

Processi e Socket

In realtà quando parliamo di client e server facciamo riferimento ai processi applicativi che sono in esecuzione sugli host. Un **processo applicativo è client** quando dà inizio alla comunicazione mentre un **processo server è quello che attende di essere contattato**. Processo == programma in esecuzione.
In una comunicazione un processo o è client o è server. Nel caso del peer to peer, uno stesso processo può assumere ruolo di client e server in sessioni differenti.

I processi comunicanti si inviano messaggi, che sono scambiati tramite livello di trasporto. *Per utilizzare il livello di trasporto le applicazioni usano* **l'Interfaccia Socket**.

L'interfaccia socket è quindi il modo in cui i processi applicativi possono interfacciarsi al livello di trasporto, una sorta di porta attraverso la quale mandano i messaggi. Poiché **il livello applicativo è sotto il controllo dello sviluppatore mentre i livelli inferiori sono sotto controllo del sistema operativo**, una volta incapsulato il messaggio a livello di trasporto ne perdiamo il controllo, per questo è importante scegliere a priori il giusto protocollo a livello di trasporto.

Ogni processo ha un **identificatore** costituito **dall'indirizzo IP a 32 bit** dell'host e dal **numero di porta specifico** per riconoscere il processo in esecuzione (necessario perché ci possono essere più processi in esecuzione). Numeri di porta sotto 1024 processi well known con privilegi di root, applicazioni ben note es. HTTP 80, Mail 25.

Ogni volta che due entità remote comunicano è necessario un protocollo che governi la comunicazione.
Nel caso dei protocolli a livello applicativo questi devono definire:
Tipo di messaggio == se è richiesta o risposta
Sintassi == come sono definiti i campi e le informazioni all'interno di essi
Semantica == il significato delle informazioni nei campi
Regole == per disciplinare come inviare i messaggi e come rispondere

Quando scrivo un'applicazione devo sempre definire un protocollo per questa, cioè definire come i processi comunicano tra loro. Esistono **protocolli proprietari** o **di pubblico dominio**. Questi ultimi definiti da RFC e standardizzati da IETF.

Per scegliere se utilizzare a livello di trasporto TCP o UDP 4 parametri:
Perdita di dati == la mia applicazione tollera perdita di dati (es. applicazioni multimediali possono tollerare, trasferimento file necessitano affidabilità 100%)
Sensibilità rispetto al fattore tempo == applicazioni come telefonia via Internet o videogiochi online per essere efficaci necessitano ritardi bassi
Throughput == applicazioni "elastiche" se sostengono invio dei pacchetti anche con throughput diversi, sensibili alla banda se vogliono di throughput minimo
Sicurezza == cifratura, integrità dati etc.. (qualcuno potrebbe essere interessato ad attaccare la mia applicazione?

TCP e UDP

TCP: **offre** **Trasporto Affidabile** (garanzia che i dati arrivino senza errori, senza perdite e in ordine); **Controllo di Flusso** (regola la velocità di invio dei pacchetti in funzione della capacità del destinatario di processare i dati); **Controllo della congestione** (regola la velocità dell'invio dei pacchetti in funzione della congestione della rete); **Orientato alla connessione** (setup iniziale della connessione via handshaking, si settano le variabili iniziali).
Non offre **temporizzazione** (garanzie sul tempo impiegato al destinatario per ricevere il pacchetto), **garanzie sul throughput minimo** e **sicurezza**.

UDP: **trasporto di dati inaffidabile** (non ho garanzia che i dati vengano ricevuti), **non offre** **affidabilità**, **controllo del flusso**, della **congestione**, **non è orientato alla connessione**, **temporizzazione**, **throughput minimo**, **sicurezza**.

UDP offre il minimo indispensabile per trasformare i protocolli IP in protocolli di trasporto e per questo, *non offrendo garanzie, permette maggiore controllo da parte delle applicazioni in termini di temporizzazione e invio dei messaggi*.

Uso TCP se necessito affidabilità, UDP se posso permettermi perdite.
TCP e UDP legati tra loro poiché **firewall bloccano spesso UDP**, in tal caso passo a TCP e il gioco è fatto.
TCP e UDP non pensati nativamente per la sicurezza, i messaggi sensibili sono visibili in chiaro da chi li intercetta. Oggi si utilizzano standard implementati a livello di applicazione come **TLS** che cifrano i messaggi per poi inviarli via TCP, quando arrivano a destinazione TLS decifra il messaggio.

Web e HTTP

Il **Web** **è un media ipertestuale** che a differenza di altri ipertesti come enciclopedie su disco è globale e può essere contenuto in vari server.
Il Web è costituito da **pagine web**, ognuna contenente un **file HTML di base e degli oggetti/risorse**. Un oggetto può essere un file HTML, un'immagine JPEG, uno script Javascript etc…
Un oggetto è referenziato con un indirizzo in cui la prima parte identifica il nome dell'host **hostname** (server a cui chiediamo l'informazione) e un **path** che indica in ordine gerarchico l'oggetto specifico che abbiamo richiesto.

HTTP protocollo a livello applicativo che usa il *paradigma client-server*, client == **Browser** che richiede oggetti e Server == glieli fornisce.
Web è **Platform Indipendent**, non solo più client possono fare richieste contemporaneamente ma i client *anche di natura diversa* (linux, mac, windows)

HTTP usa TCP perché in quanto trasferimento informazione no perdite.
4 fasi: 1) inizializza connessione TCP -> crea una socket con il server sulla porta 80
2) il server accetta la connessione con il client
3) messaggi HTTP scambiati tra browser (client) e server
4) chiusura della connessione

HTTP 1.0 **connessione non persistente**, per ogni richiesta al server per visualizzare un oggetto toccava cambiare connessione.
Con HTTP 1.1 **connessione persistente**, in questo modo posso fare più richieste senza dover aprire molteplici connessioni. (messaggio keepalive)

HTTP **stateless**, ossia il server non tiene traccia delle informazioni sulle richieste fatte dal client. Ciò non significa che non possa modificare lo stato del server ma che quando faccio una richiesta questa non è in funzione della precedente. Fare qualcosta non stateless difficile perché tanti stati da memorizzare ma soprattutto se uno crasha per ritrovare entrambi lo stesso stato un bordello.

RTT == tempo impiegato per un pacchetto per andare dal client al server e tornare al client (include ritardi di elaborazione, accodamento e propagazione).
Stabilire una connessione TCP significa fare handshake a 3 vie con richiesta, accettazione e un altro messaggio per veicolare i messaggi applicativi.
Per ogni oggetto richiesto al server con una connessione non persistente ho 2RTT (richiesta e accettazione) + trasferimento file.

Svantaggi connessione non persistente == spende 2RTT per ogni oggetto, ha un overhead del sistema operativo per ogni connessione TCP, i Browser aprono connessioni TCP parallele per caricare oggetti referenziati.

Richiesta HTTP e codici di stato della Risposta HTTP

GET per ottenere un oggetto. Posso mettere **POST** per inserire dati (tipo in un form). Oppure uso solo GET scrivendo nell'URL di destinazione ? e poi coppia chiave valore separati da &. Con POST invio molti più dati rispetto al GET.
HEAD come fosse GET ma restituisce solo gli header della risposta, non l'oggetto. **PUT** per creare un nuovo oggetto, non sempre lo posso utilizzare (non posso cambia pagine web come mi pare).

Si hanno dei codici di stato nella risposta HTTP nella prima riga del messaggio, 5 categorie e si capisce qual è guardando la prima cifra.

HTTP porta 80

1xx Informational intermezzo per comunicare stato di connessione o avanzamento richiesta
2xx Successful
3xx Redirect ancora il client deve eseguire azioni prima che la richiesta sia soddisfatta
4xx Client Error la richiesta è sintatticamente scorretta o non può essere soddisfatta
5xx Server Error il server ha fallito nel soddisfare una richiesta valida

Es. 200 OK richiesta ha avuto successo, oggetto richiesto nella risposta
301 moved permanently nuova posizione dell'oggetto
400 Bad Request sintassi errata
404 Not Found il documento non si trova nel server
505 HTTP version not supported il server non supporta protocollo HTTP

5.

Non solo HTTP definisce il Web ma anche molti altri standard come URL, HTML e CSS.

Cookie

HTTP protocollo stateless (vantaggi semplicità nel creare applicazioni, non devo memorizzare stati, se cade la connessione non ho il problema di dover ritrovare lo stato precente e mettere d'accordo host e server) che tuttavia offre **limitazioni**. Se accedo a gmail devo tenere lo stato per cui ho fatto l'accesso altrimenti per ogni richiesta lo dovrei rieseguire, insostenibile.

È resa possibile l'implementazione dello stato via **cookie**, 4 elementi:
-**Riga d'intestazione nella risposta HTTP** dal server in cui è contenuta la richiesta da parte dal server di memorizzare l'informazione di stato cookie
-**Riga d'intestazione nella richiesta HTTP** dal client dove il client immette l'informazione di stato cookie richiesta dal server
-**Un file** con il quale il client si salva il cookie che può essere gestito dal Browser
-**Un database** nel quale il server salva l'associazione cookie-informazione di stato

Vediamo nel pratico. Lo user agent ha cookie per ebay ma accede per la prima volta ad amazon. Poiché sta contattando amazon non include i cookie di ebay e amazon si accorge che lo user agent non ha mai fatto l'accesso e quindi non ha cookie per il sito. Quindi amazon genera un cookie con ID univoco per quell'user agent che salva nel database associandolo a delle informazioni che restituisce via cookie al client. Il client si salva il cookie nel file e quando il browser rifarà accesso al sito di amazon invierà il cookie, amazon accede al database, trova la corrispondenza e consente il normale accesso senza creare nuovo cookie.

Cookie usati per autenticazioni, carrello degli acquisti, stato della sessione dell'utente (es. mail), raccomandazioni.

Visito nytimes e c'è la pubblicità. Per vedere l'oggetto della pubblicità devo contattare oltre al server di nytimes anche quello della pubblicità Adx. Nytimes crea nuovo ID e manda cookie classico che salvo nel file, Adx crea ID e cookie con riga ulteriore d'intestazione **Referrer**, dove sono salvate le informazioni sul fatto che visitavo nytimes. Se poi vado a visitare ebay e c'è un'altra pubblicità Adx mando ad Adx il cookie col Referrer, confronta col database e vede che quindi ho visitato in precedenza nytimes potendo così pianificare la pubblicità. **Ma ciò comporta che Adx possa tracciare i siti che di volta in volta visito**.

Si parla di **Cookie first party** se sono cookie generati da siti che volevo visitare, **Cookie third party** se generati da server/siti che non volevo visitare. Adx diventa in questo senso tracker. Tracciamento di terze parti disabilitato su Firefox e Safari.

Web Caching

Rappresenta uno dei modi per migliorare le prestazioni del Web.
Consiste nel **delegare il compito di rispondere a richieste** ad es. HTTP **non all'origin server ma a un server locale più vicino ai client** che fanno richiesta (la **web Cache**). Quando uno user agent richiede un certo oggetto nel web, grazie a un protocollo la richiesta non arriva all'origin server ma alla web Cache. Se questa non ha l'oggetto in memoria inoltra la richiesta al server, che invia l'oggetto in questione. La cache risponde allo user agent con quest'ultimo e salva in memoria l'oggetto di modo che se altri utenti lo chiedessero ne avrebbero accesso diretto. **L'oggetto è salvato per un periodo limitato** (potrebbe essere aggiornato nell'origin server) **dettato dalla riga Cache-Control** nella risposta da parte dell'origin server. **No Cache** se la cache non può salvarlo in memoria.

3 vantaggi: 1) riduce il tempo di risposta alle richieste (distanza minore == RTT minore) 2) se la Web Cache si trova all'interno di una rete di accesso può ridurre il traffico sul collegamento di accesso (che è spesso collo di bottiglia) 3) riduce il traffico verso l'origin server

Si nota il vantaggio nell'installare una Web Cache piuttosto che passare ad un collegamento d'accesso più veloce.

Il ruolo di Cache può essere assunto anche dal Browser. Inoltre un **obiettivo primario della cache è di non inviare oggetti se non sono copie aggiornate**, per far sì che ciò avvenga (oltre al Cache-Control) il client aggiunge alla richiesta un'intestazione: ***if modified since: data***. Se l'origin server ricevendo la richiesta si accorge che non è stato modificato allora manda **304 Not Modified**, altrimenti viene mandato l'oggetto aggiornato (**GET condizionale**!)

HTTP/2

Con **HTTP 1.1** si possono usare **connessioni persistenti** e **chiedere in pipeline più oggetti**, chiedo cioè n oggetti insieme e mi vengono inviati **FCFS** (first come first served). Problema **dell'HOL head of line blocking** per cui oggetti piccoli devono aspettare d'esser trasmetti dietro a oggetti più grandi richiesti prima. Inoltre se chiedendo tanta roba una si blocca si bloccano tutte.

Con HTTP/2 si risolvono molti dei problemi di HTTP 1.1. Ciò avviene grazie a una maggiore flessibilità concessa al server nell'inviare le risposte:
- HTTP/2 eredita metodi, codici di stato e campi di intestazione di HTTP 1.1 (**retrocompatibilità**)
- **Il server non è più forzato nella modalità FCFS**, può rispondere quindi senza rispettare l'ordine delle richieste GET dello user agent
- **Il server manda PUSH oggetti non richiesti dal client** (se il client visita una pagina Web è probabile richieda diversi oggetti di quella pagina quindi glieli manda a priori)
- **Evita il problema dell'HOL** frammentando gli oggetti in frame che sono inviati intervallandoli tra loro

Mail

Tre componenti: **User Agent** (Browser, programmi di posta elettronica), **Mail Server** (per trasferire la posta tra server e user agent, ma solo in invio) e **SMTP** ***simple mail transfer protocol***.

Due componenti principali nel mail server: **casella di posta** (che contiene la posta in arrivo) e **coda di messaggi** dove vengono posti i messaggi in uscita. Se mando un messaggio uso SMTP che lo mette in coda e prova a inviarlo**. Lo User agent non manda quindi la mail direttamente a destinazione perché l'host destinatario potrebbe essere spento**, la manda al mail server dello User agent di destinazione inserendo il messaggio nella casella di posta. Ogni 30 min prova a mandarlo, dopo un tot di tempo se non arriva allora informa lo user agent mittente. **Quindi è necessario l'utilizzo di server intermediari** per l'invio di posta elettronica.
SMTP usa TCP per garantire trasferimento affidabile, porta 25.

HTTP è protocollo **pull**, poiché tipicamente scarico oggetti.
SMTP è protocollo **push**, poiché tipicamente invio oggetti.

In HTTP ogni oggetto è incapsulato nel suo messaggio di risposta, mentre in SMTP più oggetti vengono trasmessi in un unico messaggio.

NB. si usa il protocollo IMAP per il recupero delle mail dal Mail server, e non SMTP

6.

DNS Domain Name System

Si tratta del servizio directory di Internet. Sappiamo che in Internet esistono **due identificativi** principali (sopra al livello di rete): **indirizzo IP** a 32 bit per implementare la funzione di instradamento a livello di rete e **hostname** per rappresentare univocamente un host (es. server che eroga servizi).
Si necessita di tradurre hostname in indirizzi IP al fine di mappare anche questi host tramite un processo di risoluzione dei nome.

L'approccio più statico per far ciò è l'utilizzo di un **file hosts**, dedicato al nodo specifico, che contiene le associazioni tra indirizzi IP e hostname corrispettivi. Inizialmente questo file era presente in un singolo nodo, ma con l'evolversi di Internet divenne impossibile tenerlo sempre aggiornato ed inoltre tutti gli host dovevano trafficare a quel server per poter eseguire la risoluzione, quindi traffico eccessivo (soluzione **non scalabile**).

Si passò quindi da questo approccio centralizzato a un **approccio distribuito**, il **DNS**, che opera secondo un *protocollo a livello applicativo* e che distribuisce file hosts a nodi diversi nella rete secondo uno specifico ordine gerarchico.

Protocollo applicativo **DNSSEC**, offre sicurezza.

Si utilizza il protocollo applicativo poiché il DNS essendo usato da tutti svolge una funzione critica in Internet, per questo è implementato nella periferia della rete (nel nucleo sarebbe più difficile da gestire).

Non si centralizza il DNS inoltre poiché ***single point of failure***, ***volume di traffico alto***, ***non scalabile***, ***manutenzione***, ***database centralizzato distante*** per molti utenti.

4 servizi offerti da DNS:
-**Traduzione da hostname a indirizzo IP**
-**Aliasing**: l'hostname che vedo in quanto utente è un alias. DNS permette di mappare gli alias a nomi canonici più complessi (per identificare univocamente)
-**Aliasing con mail server**: da hostname a nome canonico di mail server
-**Load Distibution**: l'hostname che l'user agent richiede viene tradotto in più indirizzi IP rappresentanti server diversi che offrono lo stesso servizio. In tal modo si ***distribuisce il carico***.

Come detto *DNS permette distribuzione gerarchica dei file hosts*. **Gerarchia**: Se voglio tradurre un hostname che non conosco affatto contatto anzitutto il **Root DNS server** (in cima alla gerarchia). Questo non offre mappatura diretta di ogni hostname con IP corrispettivo, ma fa riferimento a server di livello inferiore detti **TLD Top Level Domain** name servers, ognuno di essi di gestire hostname accomunati dall'ultima parola dopo il punto (es. .com, .org etc..). Anche i TLD non hanno mappatura diretta ma fanno riferimento a dei server, quelli più in basso nella gerarchia, detti **Authoritative Name Servers** che si occupano finalmente di tradurre l'hostname richiesto in IP (risoluzione del nome).
In realtà gli Authorative server possono talvolta far riferimento ad altri Authorative server per risolvere il nome, ***man mano che scendo nella gerarchia ho suffisso sempre più grande*** fino ad avere la mappatura effettiva.

Nel mondo 13 Root Name Server logici, ognuno con il proprio indirizzo IP, replicati con lo stesso IP in diverse parti del globo per distribuire carico.
Ogni Authorative Server è gestito da una specifica organizzazione.

Fuori dalla gerarchia esiste un altro server importante per il funzionamento DNS: **il DNS name server locale**. Questo svolge la funzione di Default Name Server per ogni host: ogni volta che accediamo ad Internet il DNS name server locale viene configurato ed è questo server che si occupa dei procedimenti legati alla risoluzione dei nomi (contattare i server in gerarchia).

Inoltre offre anche un servizio di **caching**, quando voglio risolvere un nome verifica di non averlo già nel suo file hosts, se non lo ha lo risolve e lo salva in caso di richieste successive (in questo modo evita carico ai server in gerarchia). Tipicamente fornito dall'ISP e implementato nel modem.

Due tipi di Query:
Iterativa: voglio risolvere un hostname. Contatto il DNS name server locale. Se non ha in cache la risoluzione a quell'hostname contatta il Root DNS server, che leggendo l'ultima parola dopo il punto nell'hostname restituisce un referral al name server del TLD corrispettivo che dovrà contattare. Il name server locale lo contatta, TLD mappando l'hostname gli fornisce **referral** per Authorative server, name server lo contatta e risolve finalmente il nome (l'authorative gli manda l'IP corrispondente). In ognuno di questi passaggi è possibile caching.

Ricorsiva: quando il DNS name server locale chiede di risolvere l'hostname al Root Name Server, questo si occupa del lavoro di contattare TLD che contatta Authorative che restituisce IP a TLD che restituisce IP a Root che restituisce IP al name server locale. Non è il top perché comporta carico ai server di alta gerarchia.

Si possono combinare i due approcci sfruttando l'iterativa per server in gerarchie più alte, ricorsive per gerarchie inferiori.

Record DNS

I dati nel file hosts sono salvati in strutture chiamate **RR** (**resource record**). Ognuna di esse rappresenta una quadrupla **(name, value, type, ttl)** dove il name e il value cambia in base al type:
Type A: name == hostname
value == IP dell'hostname (risolvo il nome)
Type NS: necessario per il meccanismo di referral visto nelle query iterative e ricorsive, name == dominio
value == hostname del server che può aiutarmi a risolvere il nome
Type CNAME: per aliasing, name == nome alias
value == nome canonico di quell'hostname
Type MX: per aliasing mail, value == IP del mail server associato a name

Il formato delle query e delle risposte reply DNS è lo stesso.
È costituito anzitutto da **16 bit di identificazione**, ***necessari ad associare la risposta con la corrispondente domanda***. *Infatti DNS sfrutta a livello di trasporto sia TCP che UDP*, con TCP easy associare perché orientato alla connessione, con UDP necessaria identificazione perché non ho legami diretti tra domande e risposte. Poi **16 bit di flag** necessari a molte funzioni come identificare se sto facendo domanda o risposta, se sto chiedendo una ricorsione, se la ricorsione è disponibile, se si tratta di DNS autoritativo etc…
Poi diversi bit dedicati a **contatori** per identificare la grandezza delle sezioni successive. Poi **4 sezioni**: **sezione delle domande** per domande che richiedono dato un hostname di risolvere il nome, **sezione delle risposte** in cui posso avere anche più risposte (load balancing, più IP associati allo stesso hostname), **sezione autoritativa** per il meccanismo di referral in cui viene restituito l'hostname del server da contattare e **sezione aggiuntiva** legata sempre al referral ma con l'indirizzo IP di quel server da contattare (record type A).

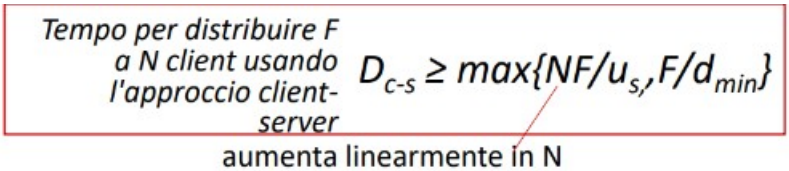
Per inserire un record nel database DNS si forniscono al DNS Registrar nome e indirizzi IP degli authoritative name server. Viene quindi salvato nel TLD corrispondente un record type NS con annesso hostname che gli ho dato e type A con annesso IP dell'hostname. Nei miei server invece salvo type A con IP del mio server e un MX per il mail server che associo.

DNSSEC offre protezione contro attacchi di spoofing, dove il malintenzionato intercetta richieste DNS e manda risposte con IP fasulli indirizzando il mittente a macchine malevoli. **Per DDoS ai Root Server estremamente difficile**: filtraggio del traffico, caching, diversi Root Server. Attaccare i TLD più facile.

Peer To Peer

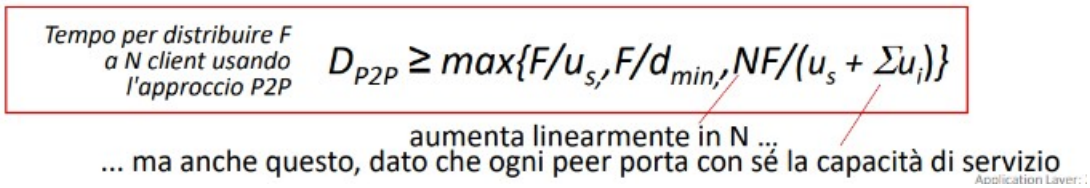
Paradigma per costruire applicazioni di rete dove *non si ha un server sempre attivo ma più client che fanno sia da client che da server*. Ogni client aggiunto appesantisce il carico ma aum

enta anche la forza lavoro, **scalabilità intrinseca**.



Qua approccio sopra paradigma Client-Server. Se il server manda un file di dim. F a N client il tempo necessario il max tra è NF/us con us = upload server e F/dmin dove dmin è il download minore tra gli N client.

La crescita è lineare nel numero di client in client-server!



Qui l'approccio P2P: anche in questo caso il tempo di trasmissione dall'host che funge in quel momento da server cresce linearmente ma la capacità di upload, poiché tutti i peer possono fungere da server, è data da tutti i peer quindi cresce più lentamente. **Quindi P2P non ha crescita propriamente lineare in N**, man mano che gli host aumentano il tempo diminuisce perché l'upload aumenta.

BitTorrent

App p2p per scambiare file. Il gruppo di peer è detto **torrent**, c'è un **tracker** che traccia i peer nel torrent e i file vengono scambiati in **chunk** da 256 kB.
Quando un nuovo peer si aggiunge al torrent:
- **Contatta il tracker** per ricevere la lista dei peer nel torrent. Questo non glieli fornisce tutti ma una cinquantina
-**Tenta di connettersi a quei peer**, quelli con cui si connette sono detti **neighbours**
-Inizia lo scambio di chunk. *Quando scarica il file* desiderato può lasciare la connessione egoisticamente o rimanere come **seeder**.

Il dialogo tra peer è basato sulla richiesta di chunk:
-Ogni peer ha sottinsiemi diversi di chunk. Un peer chiede ad un altro la lista di chunk che possiede e ne scarica in funzione della strategia **rarest first**, ovvero scarica quelli meno diffusi nel torrent. Una rete con chunk del file abbondanti p una rete infatti più robusta
-Un peer appena entrato può ignorare la rarest first chiedendo inizialmente un chunk casuale che può scaricare più velocemente
-Un peer che sta per scaricare il file può entrare in **modalità endgame** e richiedere chunk d'interesse a più peer contemporaneamente.

Problema in P2P == egoismo, per risolverlo strategia **tit-for-tat**.
Ogni peer ha un gruppo di 4 peer vicini aggiornato ogni 10 sec di **peer unchoked** che mi inviano chunk mentre io gliene invio altri, gli altri peer sono

detti **choked** (non gli invio niente). Per dare chunk a chi non ne ha affatto si sceglie ogni 30 sec casualmente per ogni peer un peer vicino detto **optimistically unchoked** a cui mandano chunk senza aspettarsi niente indietro.

Streaming video

CDN == **reti di distribuzione di contenuti** (es. Amazon Video, yt, Netflix).
Due sfide: **scala** (molti utenti richiedono il servizio) e **eterogeneità** (questi utenti hanno capacità diverse)
Per affrontarle similmente a DNS si è creata ***un'infrastruttura distribuita a livello di applicazione***.

Un video è una sequenza di immagini trasmesse a tasso costante (es. 20 fps). **Un'immagine è un array di pixel**, dove ogni pixel rappresenta un colore. In particolare **i colori sono Red, Blue e Green**, per rappresentare la scala di **ognuno di questi tre colori 1 byte** (256 valori) e le loro combinazioni portano a qualsiasi colore nella scala cromatica. **Per un pixel quindi al più 3 byte**, ma rappresentare un video codificando ogni pixel con 3 byte richiederebbe troppi dati. Esistono quindi **schemi di codifica** (codec) che permettono tramite l'ausilio di ridondanze di ridurre lo spazio.
Ridondanza spaziale == se ho un'immagine con stesso colore per diversi pixel non li codifico individualmente ma uso l'informazione legata a quel colore e al numero di pixel che copre
Ridondanza temporale == nel passaggio da un'immagine all'altra in un video codifico le differenze di pixel tra le due senza ricostruirla da zero

Due tipi di codifica: **CBR** bit rate costante in cui codifico ogni frame con lo stesso numero di bit, **VBR** bit rate variabile in cui il bit rate cambia in funzione della quantità di ridondanza. Maggiore bitrate == maggiore dettaglio/qualità.

7. (ancora streaming)

Il problema principale per lo streaming (contenuti registrati inviati man mano dal server) risiede nella **variabilità di banda**.
Nel mondo utopico un video registrato in CBR è inviato dal server frame per frame, appena il client riceve il primo frame inizia a riprodurre perché intanto gli arrivano gli altri frame dal server e quindi riproduzione pulita.

Intanto l'utilizzo dello streaming vantaggioso rispetto a scaricare perché se blocco in mezzo non spreco la banda per inviarmi il resto.
Il requisito principale dello streaming è che la riproduzione una volta iniziata deve seguire linearmente (**vincolo di riproduzione continua**), ad ostacolarlo vi

è il fatto che i ritardi nelle reti sono variabili (**jitter**). Si necessita quindi di una coda al lato buffer per soddisfare vincoli di riproduzione continua, dove il client immagazzini più frame inviati dal server (ne scarico un po' prima di partire).

Tre tipi di Streaming in base al protocollo:
-**Streaming UDP**: sfruttando UDP che non soddisfa controllo di congestione e flusso si ha ***maggiore controllo in termini di temporizzazione***, quindi la coda lato client equivalente a pochi secondi di video. **Svantaggi**: UDP non orientato alla connessione quindi è necessario stabilirne una con altri protocolli anche solo per mettere in pausa il video, inoltre se c'è comunque calo di banda allora i frame non arrivano nel tempo previsto e quindi non è rispettato il vincolo di riproduzione continua
-**Streaming HTTP**: sfrutta TCP con protocollo applicativo HTTP, è come se stessi scaricando un contenuto in rete. Meno controllo temporizzazione ma connessione nativa, esiste header Range che permette di saltare parti del video specificando quali frame scaricare. Se la capacità del client è molto minore rispetto a quella di invio del server (e quindi la coda si riempie velocemente) entra in gioco il controllo di flusso che riduce la velocità di trasmissione.
-**Streaming dinamico adattivo su HTTP** (**DASH**): non ho un unico file che posso scaricare ma diversi (ognuno con URL diverso) ognuno spezzettato in segmenti con vari bitrate, posso quindi scegliere qualità differenti anche durante la riproduzione.

DASH: *il file è diviso in più chunk, ognuno di essi codificato in bit rate diversi e memorizzato in file diversi*. Ogni file in server CDN diversi. Esiste un ***file manifest*** che fornisce l'URL per i diversi chunk.
Quando voglio fare lo streaming prendo il file manifest, decido quale file scaricare. Durante lo streaming stimando periodicamente la banda da server a client può scegliere bit rate più alto e scaricare chunk con migliore bit rate.

CDN

Per implementare servizi streaming con tantissimi utenti non basta un unico cluster, ***necessaria architettura distribuita geograficamente*** (con un solo data center singolo punto di rottura, congestione, percorso lungo verso clienti lontani, **la soluzione non è scalabile**!)

Due apporcci per realizzare l'architettura CDN:
-**Enter Deep** == i server sono collocati dentro le reti di accesso in modo da essere vicini agli utenti e garantire maggiori prestazioni. Prezzo da pagare maggiore complessità gestionale e manutenzione

-**Bring Home** == i CDN sono sparsi in pochi grandi cluster in IXP (internet exchange point) vicino alle reti di accesso.

Vediamo come funziona una CDN. Ipotizziamo che bob voglia vedere un video su NetCinema. Contatta l'homepage dove ottiene l'URL del filmato. Il DNS locale di Bob risolve (trova l'ip) dell'URL in questione contattando il server autoritativo di netcinema che risponderà con un CNAME (che permette di ricavare da un host alias il suo vero nome canonico). Questo nome canonico punta al server autoritativo della CDN, che viene quindi contattato dal server DNS locale di Bob e gli fornisce l'IP del server che gli consentirà di vedere il filmato.

Il server autoritativo della CDN può anche fornire l'IP del cluster più vicino garantendo servizio migliore. Netflix usa CDN ma la opera direttamente, possiede diversi nodi della sua CDN OpenConnect

Livello di trasporto

Per far sì che un processo applicativo possa comunicare con un altro presente in un host remoto come visto è necessario che il livello di applicazione si affidi al livello di trasporto. **Il livello di trasporto infatti supporta la comunicazione remota tra processi remoti**. Si parla di ***comunicazione logica end-to-end***, poiché si limita alla comunicazione tra i due processi *senza preoccuparsi di ciò che avviene effettivamente nel nucleo della rete*. Infatti livello di trasporto, così come applicativo, **sono implementati esclusivamente nei sistemi periferici**.

In poche parole: il livello applicativo vuole inviare un messaggio ad un processo applicativo remoto. Passa il messaggio all'interfaccia socket che permette il passaggio al livello di trasporto, quest'ultimo può frammentare il messaggio e incapsularne i frammenti in segmenti aggiungendo un'intestazione header. Il livello di trasporto si occupa solo della comunicazione logica tra i due host, quindi per far sì che il segmento venga inoltrato nel nucleo della rete deve affidarsi ai servizi offerti dal livello di rete, che incapsula il segmento in un datagramma.

Un router, che lavora fino a livello di rete, quando deve inoltrare un datagramma si limita a far uso dell'header di quest'ultimo, senza considerare affatto il segmento.
Due principali protocolli a livello di trasporto: **TCP** per trasferimento affidabile con connessione e **UDP** non affidabile senza connessione.

Mentre livello di trasporto == comunicazione logica tra processi, livello di rete comunicazione logica tra host.

Poiché il livello di trasporto si occupa della comunicazione logica tra molteplici processi mentre il livello di rete di quella tra i singoli host, è fondamentale un processo di ***multiplexing/demultiplexing*** (da visione di più processi a visione di "singolo host" e viceversa). **NB** mux/demux meccanismo fondamentale per Internet, caratterizza tutti i protocolli a livello di trasporto.

UDP == inaffidabile e non orientato alla connessione
Per sua natura UDP non permette di avere garanzie che i pacchetti che ho inviato arrivino al destinatario, che arrivino senza errori e nell'ordine in cui li ho inviati. Di fatto **UDP rappresenta la più semplice estensione di IP protocollo a livello di rete**: UDP si limita a far comunicare due processi remoti e quindi **attua multiplexing/demultiplexing** e **implementa dei codici per il controllo e correzione di errori**.

TCP d'altra parte offre servizi aggiuntivi rispetto a UDP. *Chiaramente implementa mux/demux*, ma in più è **affidabile** (ossia quando mando una serie di pacchetti ho garanzia che arrivino senza errori o buchi e in ordine), è **orientato alla connessione** (handshaking previo scambio di messaggi) e **implementa controllo di flusso e congestione** (che limitano la frequenza di trasmissione da parte del mittente per motivi diversi).

Né TCP né UDP offrono garanzie in termini di throughput e ritardi, ciò rende come visto difficile l'implementazione di applicazioni multimediali come streaming video che sono sensibili al fattore tempo (temporizzazione).

Multiplexing/Demultiplexing

Sappiamo anzitutto che ogni processo ha la propria socket.
Multiplexing == il protocollo di livello di trasporto riceve i *dati dalle varie socket e li deve incapsulare in un segmento* utilizzando una specifica intestazione, per poi passarlo a livello di rete
Demultiplexing == il livello di trasporto *riceve i segmenti dal livello di rete e, in base all'intestazione, deve affidare i dati alla socket corretta*.

Per implementarli si utilizza la logica seguente. **Sappiamo che ogni processo è identificato da numero di porta per il processo specifico e indirizzo IP dell'host su cui è in esecuzione**. **Un segmento a livello di trasporto contiene informazioni relative a numero di porta di origine e di destinazione**, mentre

il datagramma informazioni solo dell'indirizzo IP di origine e destinazione.
Quando arriva il datagramma a destinazione spacchetta e grazie alle informazioni legate al numero di porta e indirizzo IP riesce a demultiplare.

Il formato di un segmento TCP/UDP è costituito da 16 bit per porta di origine, 16 bit per porta di destinazione, eventuali altri bit per l'intestazione e payload (messaggio incapsulato)

Demultiplexing UDP:
Poiché UDP è senza connessione nella fase di creazione del segmento e del datagramma ***si devono specificare semplicemente*** **numero di porta di destinazione e indirizzo IP di destinazione** ***rispettivamente***.
Quando il datagramma arriva a destinazione e viene incapsulato, guardando IP e numero di porta, si è in grado di mandare i dati alla socket corretta e quindi concludere la demultiplazione con successo.
Se arrivano in UDP più datagrammi con IP sorgenti/numeri di porta sorgenti diversi ma stesso IP e numero di porta destinatario allora i dati saranno inviati comunque alla stessa socket.

Demultiplexing TCP:
Poiché TCP orientato alla connessione, si deve specificare la quadrupla IP di **origine, porta origine, IP destinazione, porta destinazione**.
In particolare si ha una **socket passiva** in attesa di connessione con altri host.
Quando viene accettata la connessione viene creata una nuova socket, detta **socket connessa**, con stesso IP e porta della passiva ma che accetterà dati solo se inviati da uno stesso processo in uno stesso host, quindi con stesso numero di porta e IP d'origine.

NB Mux/Demux fondamentale in internet poiché tecnica usata su tutti i livelli ogni volta che un livello richiede i servizi di quello immediatamente inferiore.

UDP

UDP viene utilizzato poiché per via del fatto che non implementa controllo di congestione e flusso permette di inviare senza restrizioni pacchetti e consente ***controllo più preciso in termini di temporizzazione***.
È utilizzato da protocolli applicativi come **DNS**, **HTTP/3**, e **applicazioni multimediali** (es. streaming video, videogiochi interattivi, telefonia via internet tutti sensibili ai ritardi e throughput)

UDP **implementa checksum** per il controllo degli errori e **non frammenta i messaggi applicativi**, quindi bisogna non eccedere nelle dimensioni del pacchetto.

PDU: 16 bit num porta origine, 16 bit num porta destinazione, 16 bit lunghezza (sia per intestazione che per payload), 16 bit checksum, payload.

In UDP indirizzo IP di origine e porta di origine servono solo per inviare eventuale risposta, non è come nel caso TCP che sono necessari per la connessione.

Checksum

Il mittente tratta il contenuto del segmento come sequenze di interi a 16 bit. Ne faccio la somma e se ho un riporto più a sx lo sommo con la cifra meno significativa, poi faccio complemento a 1 (inverto i bit) e mando questi 16 bit nel campo checksum. I ricevente calcola la checksum allo stesso modo del mittente ma sommando poi anche il valore checksum ricevuto. **Se ho tutti 1 allora nessun errore rilevato (starei sommando valori con il loro complemento, ottengo sempre 1) altrimenti no**.
Un altro metodo verificano la checksum calcolandola come fa il mittente e confrontandola col valore ricevuto.
Codici di errore come checksum offrono una garanzia probabilistica di rilevare errori, ma non è certo che non vi siano stati errori per cui comunque il controllo ha avuto successo.

8.

Trasferimento affidabile

TCP protocollo full duplex che consente trasferimento bidirezionale, per vedere come funziona astrarremo a half duplex.
L'astrazione che il livello applicativo vede è il trasferimento del messaggio da un processo applicativo a un altro, ma nel pratico ciò avviene tramite l'esecuzione di codice nel *mittente e nel destinatario che non hanno visione globale e conoscono solo ciò che gli arriva*.

Nonostante a livello logico stiamo considerando una comunicazione unidirezionale da processo a processo, l'implementazione consiste nello scambiarsi pacchetti durante questo fenomeno affinché il messaggio che volevo mandare arrivi senza errori, senza duplicati e in ordine (affidabilità).

La complessità nell'implementazione dell'affidabilità TCP risiede nel fatto che **il livello su cui trasporto si appoggia è rete, che è inaffidabile**!

rdt_send == da livello applicativo a livello di trasporto (dati che dovranno arrivare al livello applicativo del ricevente)
udt_send == da livello di trasporto a rete per inviare il pacchetto tramite canale inaffidabile
rdt_rcv == da livello di rete a livello di trasporto
deliver_data == da livello di trasporto a livello applicativo
Il destinatario può altresì usare udt_send per inviare segmenti di riscontro al mittente (es. per comunicargli che ha ricevuto il pacchetto)

Canale affidabile rdt 1.0: presupponendo (cosa non vera) che il livello di rete sia affidabile, allora semplicemente mando i dati da applicazione a trasporto rdt_send(data), creo il pacchetto make_pkt(data) e lo mando al livello di rete udt_send(pkt). Il ricevente riceve il pacchetto dal livello di rete rdt_rcv(pkt), estrae i dati extract(pkt, data) e manda i dati al livello applicativo deliver_data(data).
Semplicemente se presuppongo che il canale sottostante sia affidabile non devo costruire nulla che mi garantisca l'affidabilità quindi mando semplicemente il pacchetto.

Canale con errori nei bit rdt 2.0: se vi è la possibilità di errori nei bit l'idea è di ***utilizzare un EDC codice di controllo errori come checksum***. Il mittente per sapere se il pacchetto è stato ricevuto dal destinatario senza errori deve attendere una risposta che è del tipo **ACK** (se è ok) o **NAK** (se aveva errori). Con questo meccanismo inoltre si crea la necessità di attendere ogni volta che invio un pacchetto di ricevere un responso (***stop and wait***).
Quindi **LATO MITTENTE** ***due stati: uno per prendere i dati dal livello applicativo e mandarli e l'altro in attesa del responso***. Inizialmente ricevo i dati, faccio il pacchetto con checksum annessa e lo mando a livello di rete. Mi sposto nello stato di attesa ACK o NAK, se ricevo ACK torno allo stato di prima e mando un nuovo pacchetto, se ricevo NAK resto nello stato di attesa e rimando il pacchetto.

LATO DESTINATARIO singolo stato, se il pacchetto è corrotto (verifico via checksum) mando NAK altrimenti deliver_data e mando ACK.

Questa visione non è ancora completa perché pure ACK e NAK potrebbero essere corrotti.

Canale per gestire duplicati rdt 2.1

Cerchiamo di risolvere il problema per cui mandando ACK o NAK questi potrebbero essere corrotti***. Si aggiunge quindi checksum anche ad ACK e NAK, se il mittente verifica che sono corrotti rimanda il pacchetto***. Ma ciò comporta la **possibilità di pacchetti duplicati**. Infatti se il mittente manda un pacchetto, il destinatario lo riceve e lo manda a livello applicativo inviando ACK ma ACK arriva corrotto, allora rimanda un pacchetto già ricevuto correttamente.
Per risolvere l'idea è considerare i pacchetti con ***un numero di sequenza 0 o 1***.

LATO MITTENTE: si hanno ***4 stati*** totali. Primo stato attende i dati da livello applicativo, li incapsula in un pacchetto con numero di sequenza 0 e manda il pacchetto a livello di rete udt_send, poi transita nello stato di attesa ACK o NAK 0. Se riceve ACK o NAK corrotto o di tipo 1 o NAK rimanda il pacchetto 0, se riceve ACK passa allo stato in cui prende i dati dal livello applicativo e li incapsula in un pacchetto con numero di sequenza 1. Lo manda al livello di rete e passa allo stato di attesa ACK o NAK 1. Se corrotti o di tipo 0 o NAK allora rimanda pacchetto 1, se ACK passa allo stato visto inizialmente del pacchetto 0.

LATO DESTINATARIO: si hanno due stati, il primo in attesa del pacchetto di tipo 0 e il secondo di tipo 1. Sto nello stato di attesa 0, se arriva il pacchetto 1 mando ACK di tipo 1 (capisco che 1 è duplicato e devo far transitare il mittente nello stato che mandi pkt 0), se invece arriva pacchetto 0 mando ACK e passo allo stato di attesa pacchetto 1. Qui al contrario se arriva pacchetto 0 allora ACK prima arrivato corrotto e quindi duplicato e quindi mando ACK di tipo 0, se arriva pkt 1 allora transito nello stato di attesa pkt 0.
Se mi arriva pkt giusto ma corrotto come al solito mando NAK e attendo nello stato in cui mi trovo.

Teoricamente manco serve mettere numero di sequenza ad ACK e NAK perché il destinatario muovendosi tra i due stati garantisce che comunque non si muove fintanto che non riceve il pacchetto con numero di sequenza corretta.

Uguale senza NAK rdt 2.2: sto giro mi serve sequenziare gli ACK per forza. Quando ricevo un pacchetto corrotto, invece di inviare NAK invio ACK dell'ultimo pacchetto ricevuto correttamente così il mittente mi rimanda il pacchetto attuale (es. mi arriva pkt 0 corrotto, vuol dire che il mittente si trova in attesa di ACK di tipo 0, se gli mando ACK di tipo 1 rimanda il pacchetto di tipo 0).

Canali con errori e perdite rdt 3.0:

Stavolta oltre al problema dei duplicati e degli errori si aggiunge **la possibilità che il pacchetto venga proprio perso nel nucleo della rete**. Per gestire ciò l'idea è di utilizzare un **timer**

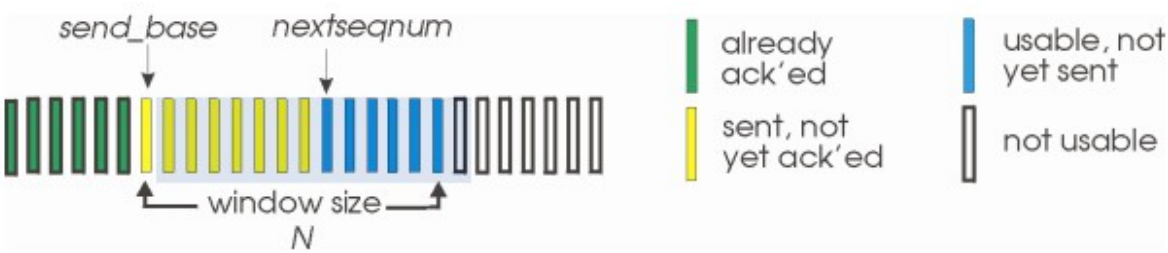
, *tempo ragionevole entro il quale da quando il mittente ha inviato il pacchetto egli si attente di ricevere l'ACK di conferma*.
Se scade il tempo (**timeout**) allora rimanda il pacchetto in questione.
Se si arriva al timeout ma il pacchetto non è stato perso e arriva al destinatario allora il mittente invia un duplicato che però sarà agevolmente gestito come abbiamo visto con rdt 2.2

GoBackN e Selective Repeat

Utilizzare rdt 3.0 così com'è garantisce trasferimento affidabile, ma comporta ***un'utilizzazione del canale pessima*** dove **utilizzazione == tempo nell'arco della trasmissione in cui il mittente trasmette effettivamente i bit**.
Ciò è dovuto al fatto che il meccanismo stopandwait comporta l'attesa per ogni pacchetto inviato di un ACK di conferma prima di poter inviare il secondo.

Per ovviare al problema l'idea è di utilizzare come per HTTP un meccanismo di **pipelining**, per cui il mittente ammette più pacchetti in uscita contemporaneamente. Ciò però non soddisfa stopandwait e quindi rdt 3.0 non funziona più, bisogna complessificare e si possono usare 2 algoritmi: **GoBackN** e **Selective Repeat**.

GoBackN



GoBackN permette di inviare più pacchetti contemporaneamente senza rispettare il meccanismo di stopandwait. Per far sì che ciò sia possibile i pacchetti devono essere ***sequenziati*** con un valore (non più 0 o 1, ma da 0 a N-1). Il mittente può inviare fino ad N pacchetti contemporaneamente, si ha una finestra di dimensione N divisa in pacchetti inviati ma non riscontrati (giallo, **pacchetto più vecchio da riscontrare send_base**) e pacchetti che posso inviare (**primo pacchetto che invierò nextseqnum**).

LATO MITTENTE: a differenza di stop and wait dove dovevo attendere per ogni pacchetto il corrispettivo riscontro, GoBackN si basa sul concetto di **Riscontro Cumulativo** per cui ho un **ACK[n] che rappresenta il fatto di aver riscontrato tutti i pacchetti con numero di sequenza <= n**.
Quando avviene un riscontro cumulativo l'algoritmo permette al mittente di inviare i pacchetti con sequenze successive, iniziando con nextseqnum n+1.
Inoltre ***l'algoritmo imposta di volta in volta un timer per il pacchetto send_base***, **quando scade viene reinviata tutta la sequenza di pacchetti a partire proprio da send_base** (da qui il nome GoBackN).

LATO RICEVENTE:
L'ultimo pacchetto correttamente ricevuto è detto **rcv_base**.
Se il ricevente riceve correttamente un pacchetto **riscontra quello con numero di sequenza immediatamente precedente** (per garantire ordine di sequenza).
Se invece riceve un pacchetto con numero di sequenza diverso *manda un ACK con numero di sequenza immediatamente precedente a rcv_base ignorando il pacchetto attuale*. Infatti se il pacchetto arrivato è sbagliato vuol dire che quello corretto è andato perso e quindi il mittente continuerà a mandare pacchetti con sequenza successiva *finché non scadrà il timer e li rimanderà tutti*.

Per il funzionamento dell'algoritmo è **necessario un buffer nel mittente che contenga la coda di pacchetti da inviare**. Si può anche implementare un buffer di arrivo.

Selective Repeat

A differenza del GoBackN, si ha un **Riscontro Selettivo** e non cumulativo. In questo modo ogni singolo pacchetto viene riscontrato, e per farlo è necessario che ***ognuno di essi abbia un proprio timer***. Allo scadere del timer di un certo pacchetto per cui ancora non è arrivato riscontro il mittente lo ritrasmette. Se arriva il riscontro in tempo lo marca come riscontrato all'interno della finestra.
A differenza del GoBackN *si hanno quindi pacchetti riscontrati nella finestra*

anche sequenzialmente dopo pacchetti non ancora riscontrati.
La finestra è portata avanti solo dal momento che ha completato una serie di pacchetti riscontrati.

Per garantire il funzionamento dell'algoritmo è necessario che vi sia una **coda nel lato destinatario**. Infatti *se il destinatario riceve un pacchetto non in sequenza invece di ignorarlo lo riscontra e inserisce in coda*. Se riceve un pacchetto in sequenza lo manda a livello applicativo, ciò può portare altri pacchetti in coda a poter essere inviati a livello applicativo.

NB ***Un pacchetto viene inviato solo se nella finestra è disponibile il successivo numero di sequenza***

Con Selective Repeat si presenta il problema per cui se utilizzo un numero di sequenza grande ad es. quanto la finestra il destinatario potrebbe non accorgersi di duplicati. Immaginiamo di avere una finestra di dimensione 3 e sequenza modulo 3. Se mando i pacchetti 012 il mittente attende prima di estendere la finestra di ricevere ACK. Il destinatario riceve i 3 pacchetti, li manda a livello applicativo e invia i 3 ACK, aspettandosi adesso di ricevere un nuovo pacchetto con sequenza 0. Ma se vengono persi tutti e 3 gli ACK, il mittente rimanderà i pacchetti 012 e il ricevente non si accorgerà che sono duplicati (ha già spostato la sua finestra!)
La finestra non deve essere più grande della metà del modulo dei numeri in sequenza.

9.
Nelle reti la principale causa di perdite è la congestione.

TCP

Protocollo a livello di trasporto come già detto **affidabile**.
TCP orientato allo ***scambio di messaggi tramite flussi di byte***, per cui la dimensione dei messaggi applicativi che vogliamo inviare non deve rispettare un limite massimo poiché sarà frammentato ed inviato a flussi di byte (con UDP invece bisognava essere accorti da questo punto di vista). *È necessario tuttavia che vi sia un modo per cui le applicazioni identifichino la fine di un messaggio*.

TCP in termini di affidabilità adotta il meccanismo di **Riscontro Cumulativo** e permette il **pipelining**, *regolando la dimensione della finestra in funzione del controllo di flusso e di congestione*.
TCP è inoltre **orientato alla connessione**: prima dello scambio di messaggi

avviene un handshaking a 3 vie (3 segmenti scambiati) che inizializzano vari parametri per garantire uno scambio di messaggi pulito e affidabile.

TCP **protocollo end-to-end** (solo due host connessi tra loro) e **full duplex** (permette lo scambio di flussi di byte vicendevolmente in contemporanea).

Il processo applicativo pone i dati all'interno di un buffer di invio. TCP periodicamente (secondo algoritmi di controllo congestione/flusso) li estrae e incapsula in segmenti che incapsula in datagrammi e spedisce. Quando arriva il datagramma spacchetto, segmento spacchetto, i dati li pongo in un buffer di ricezione che si occuperà di mandarli a livello applicativo.

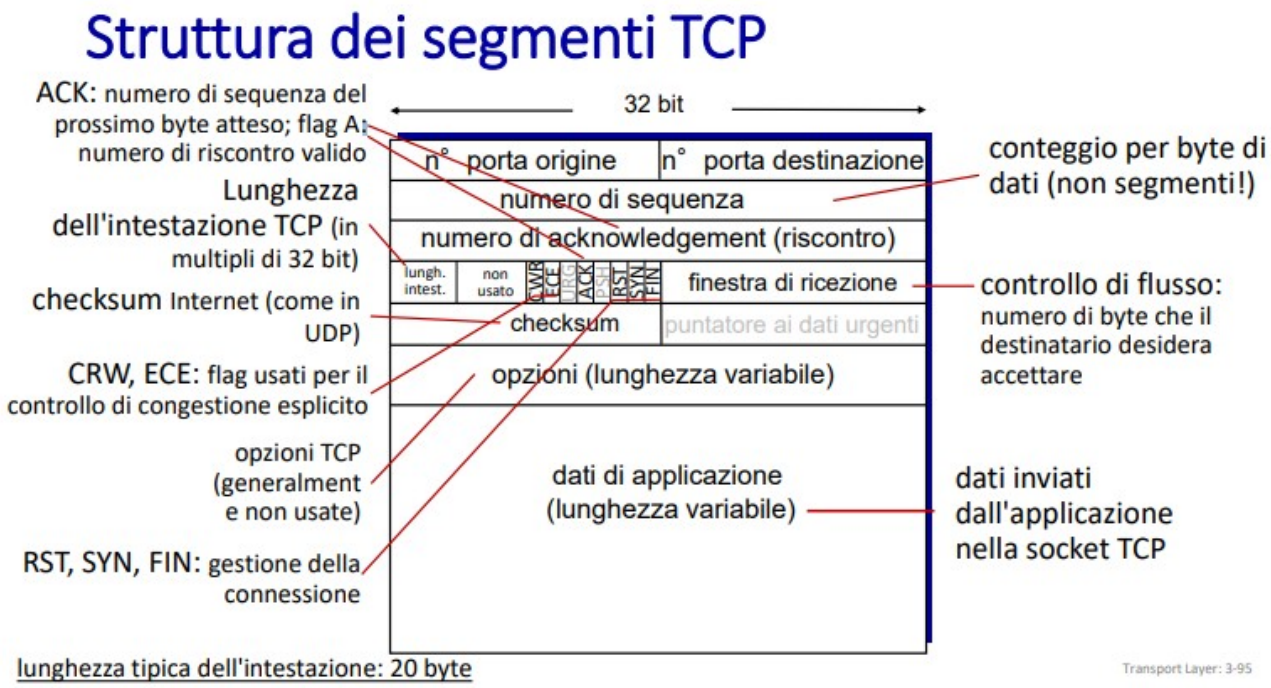
MTU

In generale si vuole che un segmento non venga spezzato in più pacchetti di rete. Un segmento è formato da header e payload applicativo, la lunghezza del payuload applicativo non può essere maggiore di un parametro detto **MSS maximum segment size**.

Per calcolare il valore di MSS e capire quindi qual è la dimensione massima di messaggio che posso trasmettere per segmento *devo far riferimento ad un altro valore a livello di collegamento*: **l'MTU Maximum Transmission Unit**.
Si tratta di un valore che rappresenta la *dimensione massima in byte di un pacchetto affinché possa essere inviato usando un certo protocollo a livello di collegamento*. ***MTU == header rete Hn + header trasporto Ht + MSS***.
L'MTU cambia in base al protocollo a livello di collegamento, se ad esempio uso Ethernnet vale 1500. Sapendo che in genere Ht e Hn = 20 -> MSS = 1460.

Quindi un host **per conoscere MSS deve conoscere anzitutto MTU**, per farlo esistono **Algoritmi di MTU Discovery** che lo permettono che comunicano al mittente l'MTU più bassa nel percorso mittente-destinatario.
Se un pacchetto IP eccede la MTU, *viene inoltrato al mittente il messaggio che gli comunica la cosa* e inoltre viene **frammentato (IPv4)** o **scartato (IPv6)**.

PDU TCP



16 bit porta origine, 16 bit porta di destinazione (ricorda che a differenza di UDP la socket TCP è identificata da IP e porta destinatario+IP e porta sorgente)

Si ha poi il **numero di sequenza**. In TCP a differenza di ciò che abbiamo visto *il numero di sequenza non dipende dal numero di pacchetti inviati ma dal numero di byte inviati*: se invio un singolo byte il numero di sequenza sarà 1, se invio un flusso di 8 byte il numero di sequenza sarà 7.
Si ha il **numero di ACK**: in particolare *rappresenta il prossimo byte atteso* es. se invio 10 byte avrò numero di sequenza pari a 9 e quindi in ACK 10 (prossimo atteso).
Si hanno dei **flag** tra cui uno ***dedicato al controllo della congestione***, 16 bit dedicati alla **finestra di ricezione per il controllo di flusso**, 16 bit di **checksum** per il controllo degli errori come in UDP.
Si hanno poi delle **opzioni TCP** spesso non utilizzate *di lunghezza variabile*, così come è di lunghezza variabile il payload applicativo; *per identificare quindi la* **lunghezza dell'intestazione** *di hanno 4 bit in multipli di 16 bit* (4 byte). Tipicamente Ht TCP lunga 20 byte.

Quando viene ricevuto ACK (poiché riscontro cumulativo) di un byte tutti i byte fino a quest'ultimo (escluso) vengono riscontrati e quindi la finestra si sposta a quello successivo. Per gestire i segmenti fuori sequenza si usa tipicamente un buffer lato client, ma *dipende dall'implementazione e non c'è una regola generale*.

A manda a B #seq = 42 e ACK = 79. Significa che sta mandando byte con numero di sequenza 42 e fa l'ACK dei byte che gli sono arrivati fino a 78.
B risponde inviando un nuovo byte ad A e facendo l'ACK del 42 appena arrivato: #seq = 79 e ACK = 43 (Seq == numero byte inviato, ACK == numero byte che mi aspetto).

TCP chiaramente usa anche un **timer** per evitare perdite. *Ma come lo calcola*?
In base a RTT roundtrip time (tempo impiegato tra invio e ricezione dell'ACK).
Se timeout più piccolo di RTT per definizione non posso ricevere un riscontro, se ***timeout troppo grande allora ritardo nel riparare la perdita***.

L'ideale è quindi avere timeout vicino a RTT, ma leggermente di più per sicurezza. Per stimare RTT posso fare un **SampleRTT** inviando un pacchetto e contando quanto tempo impiego a ricevere l'ACK, ma RTT è variabile!
L'idea è quindi di fare una **media mobile livellata esponenziale** per cui prendo il campione attuale moltiplicato per alpha e lo sommo con il campione precedente moltiplicato per alpha – 1. Tanto più scelgo grande alpha, tanta maggiore importanza darò al SampleRTT attuale. Alpha raccomandato 0.125.

Questo valore è **l'Estimated RTT**, calcolo poi il **DevRTT** allo stesso modo ma usando Beta con 0.25 e facendo la sottrazione invece che la somma. Sommo ***Estimated RTT e 4*DevRTT ottenendo il valore timeout da utilizzare***.

Comportamento di TCP

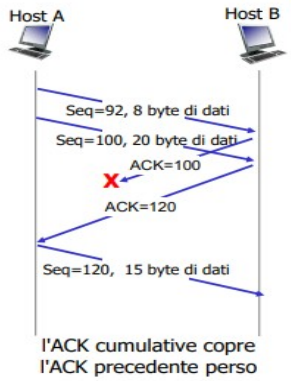
LATO MITTENTE: quando l'applicazione vuole mandare dati viene creato il segmento TCP e ***impostato il numero di sequenza*** (in base eventualmente ai byte che sono già stati inviati con altri segmenti). Quando il segmento viene inviato inoltre parte un **timer**. Si può pensare che ***il timer sia associato al segmento più vecchio*** (**sendbase**) ancora non riscontrato.

Quando si arriva al timeout il segmento sendbase viene ritrasmetto con timer raddoppiato (a differenza di GoBackN dove si ritrasmetteva l'intera sequenza di pacchetti).
Quando arriva l'ACK il ***sendbase è aumentato*** (ACK cumulativo), e il timer riavviato a valore normale (se ci sono ancora segmenti inviati e non riscontrati).

LATO RICEVENTE: se arriva un segmento con flusso di byte ***ordinato il ricevente non li manda subito a livello applicativo, ma attende qualche istante***. Infatti essendo TCP basato su **riscontro cumulativo** se arrivano altri

segmenti anch'essi ordinati con il flusso di byte nel buffer manda tutto insieme a livello applicativo e un riscontro cumulativo più grande, risparmiando tempo.

Se arriva un segmento fuori ordine d'altro canto il destinatario manda subito un **ACK duplicato** per i byte che sta ancora attendendo (estremità inferiore del buco), di modo che il "buco" di byte mancanti possa essere quanto più presto colmato.



In quest'immagine si capisce *l'importanza dell'ACK cumulativo nel pipelining*: mentre il primo ACK viene perso il secondo mi riscontra tutti i byte precedenti, perciò il mittente può spostare in avanti la sua finestra di invio e continuare a trasmettere nuovi dati.

Perché ACK duplicato se arrivano flussi di byte successivi?
Perché in TCP esiste la **ritrasmissione rapida**: **una volta ricevuti 3 ACK addizionali per gli stessi dati il mittente capisce che si è creato un buco nel buffer destinatario e quindi ritrasmette immediatamente**, indipendentemente dal timer, **il segmento non riscontrato con più picccolo numero di sequenza**.

Controllo del Flusso

Controllo del flusso == il destinatario controlla il mittente in modo che non invii troppi dati e troppo velocemente, in funzione della propria capacità di gestirli

Nel caso di TCP il controllo del flusso è implementato tramite il campo nella PDU **rwnd** (**finestra di ricezione**) ***che indica quanti byte ancora liberi sono presenti nella coda di ricezione***. Sia **RcvBuffer** la dimensione del buffer di ricezione, allora rwnd è pari alla RcvBuffer – byte occupati e i byte occupati sono uguali all'ultimo byte ricevuto – l'ultimo byte letto

```
rwnd = RcvBuffer - (LastByteRcvd - LastByteRead)
```

In tal modo viene limitata la frequenza di invio di dati da parte del mittente affinché il buffer non vada in overflow e i dati persi.
In particolare si vuole che il numero di byte disponibili da inviare dal mittente (LastByteSent – LastByteAcked) sia <= di rwnd.

Handshaking

Necessario per dimostrare che entrambi gli host sono disponibili ad una connessione e **per concordare i parametri di connessione** (es. numeri di sequenza iniziali e receive buffer)

Handshaking a 2 vie: *il client invia la richiesta di connessione al server che dovrà accettarla*. La **problematica** nasce dal momento che ad esempio prima della conferma di connessione scade il timeout e il mittente rimanda la richiesta: se poi questa richiesta duplicata arriva al server dopo che la connessione si è chiusa questo non riconosce il fatto che è un duplicato e quindi stabilisce una connessione a vuoto.

Handshaking a 3 vie: inizialmente client è in **stato di closed** e server in **stato di listen**. Il client invia un segmento **SYN** (flag SYN a 1) *per indicare che vuole stabilire una connessione*, numero di sequenza inizializzato x (**casuale**, non zero!). Client entra in **stato SYNsent**.
Il server riceve SYN e risponde con **SYNACK** (SYN e ACK flag a 1) con numero sequenza ACK x+1. Il server entra nello **stato SYNSENT**.
Il client riceve SYNACK e risponde con un **ultimo ACK** a cui può anche iniziare ad "allegare" byte applicativi. Il client entra in **stato ESTAB** (***established***). Il server riceve l'ACK ed entra anche lui nello **stato ESTAB**, può avere inizio la comunicazione TCP.

Con queste 3 fasi non vi è più il rischio che venga creata una connessione a vuoto.

Chiusura connessione:
La connessione viene chiusa con il **flag FIN = 1**. Il client manda un segmento di **FIN** al server ed entra nello **stato FIN_WAIT_1**, non può più inviare dati ma

solo riceverne. Quando il server riceve il FIN risponde con un **ACK**, passa allo **stato CLOSE_WAIT**. Ricevuto l'ACK dal server il client passa allo **stato FIN_WAIT_2** nell'eventualità che il server mandi altri dati. Quando il server è pronto a chiudere manda un **FIN** anch'esso e passa allo stato **LAST_ACK**. Il client quindi capisce che il server è pronto a chiudere e manda un **ACK** entrando nello **stato TIME_WAIT**. Ricevuto l'ACK il server potrà chiudere la connessione mentre ***il client dovrà aspettare 120 secondi***. *Ciò è necessario perché se il server non riceve l'ultimo ACK allora potrei perdermi la ritrasmissione di FIN da parte dal server*. Se riaprissi la connessione con lui mi potrebbe arrivare proprio quel FIN con i numeri di sequenza giusti che mi porterebbero a problemi.

Attacco SYNFLOOD

Attacco di sicurezza per cui vengono volutamente mandati dei SYN senza voler stabilire davvero la connessione. In particolare l'aggressore manda un segmento TCP SYN con IP fasullo, il server rispondendo alloca e inizializza le variabili e i buffer di connessione (sfrutta delle risorse!) e invia il segmento SYNACK alla porta e all'indirizzo IP (fasullo) di origine, transitando nello stato SYNrcvd.
La rete tenta di consegnare il segmento SYNACK all'IP fasullo, raggiungendo eventualmente un altro host che non c'entra nulla e non risponderà.
Il server non riceverà quindi l'ACK ed eventualmente (dopo un minuto o più) rilascerà tutte le risorse associate a questa connessione mezza aperta.

In questo modo l'aggressore è riuscito a consumare risorse del server. Se ci si organizza per inviare numerosi SYN a un server obiettivo, si può montare un attacco di DoS (spesso sotto forma di DDoS).

SYNcookie per fronteggiarlo vaffanculo.

Congestione

Mentre con il controllo di flusso si salvaguardava la trasmissione del mittente in funzione della capacità del destinatario di ricevere i dati (***singolo mittente sovraccarica il destinatario***) con la **congestione** sono **molti i mittenti che mandano troppo velocemente congestionando la rete**.

I sintomi della congestione sono lunghi ritardi e pacchetti persi (accodamento nei buffer dei router, overflow).

Caso con code infinite: immaginiamo di avere due host che inviano dati e si passa attraverso un router. Sia R la larghezza di banda dei collegamenti di

ingresso e uscita del router. Se entrambi gli host inviano dati con una frequenza di trasmissione lambdain che si avvicina a R/2 ho **intensità di traffico** (utilizzazione del canale) **che si avvicina ad 1**, *allora i pacchetti iniziano ad accodarsi*. Prima o poi arriveranno, ma con ritardo sempre maggiore.
Caso code finite: **in tal caso posso avere delle perdite**, quindi la frequenza di dati inviati è lambdain1 >= lambdain che contiene **anche le ritrasmissioni**! Utopicamente si vorrebbe un host che con conoscenza perfetta sappia quando è il caso di ritrasmettere o meno, ma non è così e quindi capita che pacchetti vadano ritrasmessi causa timeout prematuro, intensificando gap tra pacchetti inviati e pacchetti effettivamente necessari al destinatario.
Collasso di congestione: si crea una dinamica per cui ogni router ruba banda a connessioni provenienti da altri router, **nessun router riesce quindi effettivamente ad inviare niente**, accodando sempre più (*timeout, ritrasmissioni, ancora più code*) e dati trasmetti all'infinito, **la rete smette di funzionare.**

Insieme di servizi forniti da un protocollo == **modello di servizi**

La semantica di TCP è un flusso di byte affidabile consegnato in sequenza. È compito del livello applicativo ideare un modo per cui si possa identificare la fine di una richiesta per TCP (payload variabile)

10.

La larghezza della finestra in TCP è regolata da due algoritmi: controllo del flusso e controllo di congestione.
Timeout determinato come somma del valore medio di RTT + margine di sicurezza (che è 4 volte la stima della variabilità dell'RTT). *Ciò è per evitare timeout prematuri e quindi ritrasmissioni inutili*
Controllo di flusso si riferisce al destinatario che controlla il mittente affinché non invii dati più velocemente di quanto il destinatario non possa gestire.

Controllo di congestione controlla più mittenti affinché non inviino troppi dati troppo velocemente perché la rete li gestisca

Controllo di congestione TCP

Mentre il ***trasferimento affidabile tratta i sintomi della congestio***ne (se si perdono pacchetti vengono reinviati), il ***controllo della congestione si occupa di evitare che questa si aggravi fino allo scenario del collasso di congestione***.

Nella variante classica TCP deduce congestione in rete tramite approccio end-to-end (la congestione viene dedotta dagli host senza supporto dal nucleo).

L'idea alla base del controllo della congestione è aumentare la velocità di trasmissione nel tempo e ridurla non appena rilevo un sintomo di congestione di rete, come perdita di pacchetti.
In particolare ***TCP aumenta la velocità linearmente***: *ogni RTT incrementa la velocità di trasmissione di 1 MSS maximum segment size* (**Incremento Additivo**), non appena *rileva un evento di perdita dimezza la velocità* (**Decremento Moltiplicativo**).
AIMD == Additive Increase Multiplicative Decrease

Un approccio di questo tipo **permette di sondare la banda disponibile** perché se si vuole evitare congestione trasmetto pochissimo mentre se non c'è congestione trasmetto sempre di più.

TCP Reno:
- Se la congestione è rilevata dal fatto che mi è arrivato **triplice ACK** (il pacchetto è sicuramente andato perso) si entra in fase di **Fast Recovery** attuando il Decremento Moltiplicativo
- Se invece il pacchetto è stato perso causa timeout allora torno nella fase di **Slow Start** tagliando il tasso di invio ad 1 MSS

TCP Tahoe: precedente a Reno, non usava AIMD e semplicemente si tornava sempre a 1 MSS e slow start

Quando la crescita esponenzialestat di slow start passa a lineare si entra nella fase di **Congestion Avoidance**.

Congestion Window

Sappiamo che **TCP è un protocollo orientato all'invio di flussi di byte e che ammette pipelining** (invio contemporaneamente più byte senza aspettarmi per ognuno di essi immediato riscontro).
Per regolare il numero di byte che posso inviare contemporaneamente in pipelining abbiamo visto la rwnd finestra di ricezione, per il **controllo del flusso**. Il ricevente comunicava al mittente lo spazio che ha in coda e il mittente regola la trasmissione di conseguenza

Con il controllo di congestione *si aggiunge un ulteriore vincolo*: la ***cwnd finestra di congestione***. Immaginiamo di avere la serie di byte: si ha una serie continua in verde (inviati e riscontrati, continua perché TCP riscontro cumulativo), una serie continua in giallo di byte inviati ma non riscontrati, una in blu di byte che potrei usare ma aspetto il livello applicativo che mi mandi roba e in grigio che non posso utilizzare (se li usassi starei inviando troppo

velocemente).
TCP per il controllo di congestione limita la trasmissione ponendo la parte di byte in giallo <= di cwnd e cwnd viene regolata dinamicamente in risposta alla congestione della rete osservata:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$$

Anche il controllo di flusso però regola il tasso d'invio e per farlo **impone che l'ampiezza della finestra sia <= di rwnd**, dove rwnd erano lo spazio libero nel buffer di ricezione del destinatario.

Si accoppiano le cose con la formula

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{rwnd}, \text{cwnd}\}$$

Assumendo che il buffer di ricezione sia sufficiente grande, possiamo trascurare il vincolo della finestra di ricezione (che assumiamo sempre maggiore della finestra di congestione)

Slow Start == si parte da 1 MSS e si aumenta la velocità di trasmissione (allargando cwnd) ***esponenzialmente*** ad ogni RTT (per ogni ACK ricevuto).
Si arriva a cwnd grandi molto velocemente e si passa alla fase di **Congestion Avoidance** (crescita lineare) **quando cwnd raggiunge ½ del suo valore prima del timeout** (valore ssthresh, inizialmente 64 KB molto grande perché viene trascurato, poi impostato a ½ del valore non appena si arriva a una perdita).

Mentre TCP Tahoe usa sempre lo Slow Start come anticipato con TCP Reno si sfrutta anche la **Fast Recovery** per cui ssthresh viene come al solito impostata a metà della cwnd al momento della perdita ma invece di ripartire con slow start dimezzo cwnd e riprendo con l'incremento additivo (AIMD).

TCP Cubic

Si tratta di un'estensione di TCP Reno che sonda ancora meglio la larghezza di banda utilizzabile. **L'intuizione è, data wmax dimensione della cwnd al momento della perdita di pacchetto, di crescere esponenzialmente da wmax/2 e rallentare di molto man mano che mi avvicino a wmax (punto di rottura)**.
L'aumento di cwnd è dato dalla *funzione cubica della distanza tra l'istante corrente e l'istante K in cui cwnd raggiungerà nuovamente wmax*.

Controllo di congestione basato su ritardo RTT

Sia TCP Cubic che Reno rilevano la congestione principalmente per via del sintomo della perdita di pacchetti (timeout e triplice ACK).
Tuttavia esistono altri modi, più sottili, per rilevarla. Uno di questi è osservare l'RTT. Se l'RTT aumenta significa che per la rete si è atteso in coda di buffer e quindi che si sta riempiendo (router collo di bottiglia, potrei arrivare a congestionarlo).

Sappiamo che il throughput massimo per cui posso mandare contemporaneamente dati in pipeline è ***cwnd/RTT*** (diviso RTT perché per ogni RTT mi arriva un ACK cumulativo per cui posso mandare altra roba).
RTTmin rappresenta il valore minimo di RTT che ho quando il percorso in rete non è congestionato.
Quindi il mio ***throughput massimo*** *è* ***cwnd/RTTmin***.
Il throughput che misuro sul momento è cwnd/RTT.
-**Se quest'ultimo è vicino a quello massimo allora il percorso non è congestionato**;
-**Se invece il throughput misurato è considerevolmente più piccolo di quello massimo** inviare ancora più dati sarebbe insensato, si accoderebbero e contribuirei alla congestione. Per cui **mi conviene ridurre la velocità di trasmissione per far svuotare la coda ed evitare la congestione.**

Diversi TCP distribuiti adottano approcci come questo oltre a quello legato alle perdite, come BBR impiegato per google.
Questo approccio è vantaggioso perché si prevengono le perdite!

ECN

Un altro modo per essere proattivi ed evitare perdite è avere informazioni direttamente dalla rete. Si parla di protocolli **Explicit congestion notification**.

Ciò che accade in questo caso è che si **utilizzano 2 bit nel campo Type of Service (ToS) nell'intestazione del pacchetto IP** come ECN. Il mittente imposta questo ether a valore 10 per informare i router del fatto che è in grado di trattare quelle notifiche. Quando un pacchetto così marcato attraversa un router congestionato, questo lo vede e ci mette 11, per informare l'host a cui arriverà il pacchetto del fatto che questo ha attraversato un router congestionato. Quando il destinatario riceve un pacchetto IP di questa marcatura, nell'ACK setta il bit ECE per informare il mittente della presenza di una congestione. Quando il mittente apprende l'informazione, dimezza cwnd.

Viene quindi coinvolto in tutto ciò sia IP (necessario per informare i router, livello di rete) che TCP (sarà il destinatario a livello di trasporto a capire e informare il mittente della congestione).

NB. Si ricordi che gli host possono solo decidere di rallentare, cambiare router se uno è congestionato (**instradamento**) è deciso dalla rete.

14.

Livello di Rete

Livello applicativo e livello di trasporto hanno in comune il fatto di essere implementati all'interno degli host (periferia della rete). Un processo applicativo, per inviare un messaggio applicativo ad un altro processo remoto, si affida al livello di trasporto (socket, multiplexing, demultiplexing) per inviarlo in modo affidabile (TCP) o inaffidabile (UDP). Il livello di trasporto a sua volta offre una comunicazione logica tra due host diversi, senza preoccuparsi di quale percorso compiono i pacchetti nel nucleo della rete. *Per far sì che tuttavia i pacchetti arrivino effettivamente dal mittente al destinatario*, **il livello di trasporto deve affidarsi al livello di rete** che incapsula i segmenti in datagrammi e si occupa delle **funzioni di inoltro** e di **instradamento**.

Per tale ragione il livello di rete è il primo livello nella pila protocollare di internet a preoccuparsi del nucleo della rete, infatti ***i router nel nucleo implementano fino al livello di rete***!

Inoltro == trasferisce i pacchetti in ingresso all'adeguata linea d'uscita
Instradamento == determina a priori l'intero percorso seguito dai pacchetti dall'origine alla destinazione tramite algoritmi di instradamento.

Possiamo vedere queste due funzioni come appartenenti a due piani distinti:
Piano di dati == si occupa delle ***funzioni locali al singolo router***, come quella di inoltro. *Non si ha visione d'insieme della rete, ma solo di ciò che il router deve svolgere*
Piano di controllo == si occupa di ***funzioni dalla visione più ampia***, è legato ad una vera e propria **logica di rete**. Per effettuare l'instradamento è infatti necessario conoscere la rete affinché si capisca come instradare i pacchetti da mittente a destinatario.

Il piano di controllo può essere implementato con due approcci:
- (tradizionale) **Algoritmi di Instradamento** per cui *in ogni router è implementato un algoritmo* che, tramite la comunicazione con gli altri router, permette di scrivere correttamente le tabelle di inoltro
- **SDN** **Software Defined Networking**: approccio più recente per cui *la funzione di instradamento viene determinata e "scaricata" a partire da un componente terzo esterno, un server remoto*.

Funziona sostanzialmente così: per quel che riguarda la funzione di inoltro ogni datagramma è costituito da un header Hn contenente le informazioni legate all'indirizzo IP dell'host di destinazione. Il router guardando quest'ultimo lo confronta all'interno di una Tabella di Inoltro Locale che contiene associazioni IP-linea di uscita. Chiaramente non posso avere tabelle con associazioni per ogni singolo IP, pertanto la logica è di ragionare per blocchi di indirizzi secondo parametri che vedremo successivamente.
Per quanto riguarda invece la funzione di instradamento questa si occupa di mantenere aggiornate le tabelle di inoltro: nel caso dell'approccio tradizionale se ne occupano gli algoritmi di instradamento implementati nel router, nel caso invece di SDN il server remoto esegue gli algoritmi e il router ad esso collegato scarica le tabelle di inoltro che ne derivano (piano di controllo nel router sostanzialmente assente).

Chiaramente è possibile lavorare sul livello di rete per implementare diversi **modelli di servizio**, che possono offrire *garanzie in termini di temporizzazione, throughput, ordine di consegna dei datagrammi…*
Tra i vari modelli di servizio tuttavia quello che ha preso maggiormente piede

nel tempo è il **best effort**, che non offre garanzie di alcun tipo!
Ciò è dovuto al fatto che ***già il livello di trasporto e applicativo sono in grado di gestire situazioni complesse***, come quelle di consegna affidabile (TCP) oppure di applicazioni sensibili al fattore tempo (DASH per lo streaming che permette di adattare la qualità in base alla larghezza di banda).
Grazie alla distribuzione dei server che offrono servizi (es. CDN) inoltre non è così stringente la necessità di garantire servizi anche a livelli bassi come quello di rete, ed **è quindi preferibile adottare un approccio semplice, scalabile e facile da implementare come best effort**.

Architettura del router

Un router ha per definizione più porte di ingresso e di uscita.
Al suo interno sono presenti due "strutture" che si occupano di compiti diversi: la **struttura di commutazione** dedicata al ***trasferimento dei pacchetti da una linea di ingresso verso una specifica linea di uscita*** (a seconda delle regole dettate dalle tabelle di inoltro) e un **processore di instradamento**, che ***eseguendo gli algoritmi di instradamento si occupa di scrivere e mantenere aggiornate le tabelle di inoltro***.

Quindi ***struttura di commutazione == piano di dati***,
processore di instradamento == piano di controllo

Piano di dati e piano di controllo lavorano in misure temporali diverse.
Infatti *affinché la commutazione non sia collo di bottiglia* è necessario che il trasferimento dei pacchetti dalla linea di ingresso alla corretta linea di uscita avvenga **nell'ordine dei nanosecondi** (veloce almeno quanto la velocità di trasmissione). Nel caso del processore questo invece può lavorare anche **tra i millisecondi e i secondi**, infatti non è necessario aggiornare le tabelle così frequentemente.

La prima funzione svolta dalla porta di ingresso è quella di **terminazione di linea**, legata al *livello fisico*. Si passa poi al livello di collegamento dove *il frame viene decapsulato tirando fuori il datagramma per svolgere la funzione di inoltro a livello di rete*.
Mentre **fisico e collegamento sono implementati in hardware**, **rete una via di mezzo**. Hardware per ciò che abbiamo appena descritto, e quindi anche inoltro, velocizza il tutto. Software per gli algoritmi di instradamento che girano nel processore di instradamento.

Nelle architetture più recenti la scelta della linea di uscita corretta avviene localmente alla linea di ingresso, secondo un meccanismo "**match plus action**". Esistono due approcci:
- **Inoltro Tradizionale basato su Destinazione** per cui la porta di uscita è scelta in funzione esclusivamente dell'IP di destinazione
-**Inoltro Generalizzato** per cui la porta di uscita è scelta in funzione anche di altri parametri.

Inoltro basato su Destinazione

Poiché con 32 bit circa 2^32 possibili indirizzi IP, sarebbe impensabile che ogni router abbia tabelle così grandi. Pertanto ***si ragiona in termini di blocchi di indirizzi*** (se hai un certo prefisso allora vai nella prima linea di uscita, se ne hai un altro allora vai nella terza etc…). *Tuttavia che succede se ho due blocchi di indirizzi per cui qualche indirizzo coincide*? La priorità va al matching con **Prefisso più Lungo**.

Questa operazione viene implementata in hardware tramite delle **TCAMs** (ternary content addressable memories). Si tratta di memorie particolari, che seguono la logica del **content addressable**: *un indirizzo IP a 32 bit è passatto alla memoria che restituisce il contenuto della tupla nella tabella di inoltro corrispondente a quell'indirizzo in tempo sostanzialmente* ***costante***.

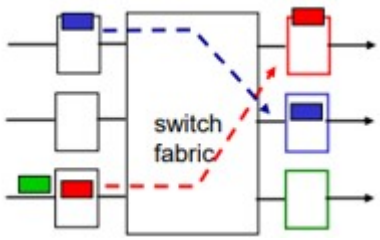
Per garantire tempistiche così veloci ***non si scorrono tutte le celle, ma vi è un comparatore per ogni riga che le confronta tutte parallelamente***. Si memorizzano i prefissi dal più lungo al più corto in modo che la prima corrispondenza selezionata sia quella dal prefisso più lungo.

Struttura di commutazione

Come detto si occupa del trasferimento dei pacchetti dalle porte di ingresso a quella specifica di uscita. Deve lavorare nell'ordine dei nanosecondi ed essere anche più veloce dei collegamenti di ingresso al router! In particolare se si hanno N ingressi e larghezza di banda in ingresso R, si vuole che **il tasso di trasferimento** da parte della struttura di commutazione **sia almeno NxR**. ***Voglio infatti evitare a tutti i costi che vi sia accodamento nelle linee di ingresso***, se mi arrivano N pacchetti dalle linee di ingresso devo essere in grado di trasferirli prima che in tempo L/R me ne arrivino altri ancora.
Per questa ragione *negli esercizi si trascura il ritardo di accodamento nelle linee di ingresso e di elaborazione, considerando solo il possibile ritardo di accodamento nelle linee di uscita*.

Tre approcci:
-**Commutazione in memoria**: utilizzata dai primi router: *i pacchetti vengono immagazzinati anzitutto in memoria, si occupa poi il processore di instradamento di indirizzarli alla corretta linea di uscita*.
Particolarmente lenti per via del fatto che i pacchetti transitano due volte in memoria, e larghezza di banda della memoria lenta.
-**Commutazione tramite bus**: il processore di instradamento non interviene più nella commutazione. *I pacchetti vengono trasferiti a partire dalla porta di ingresso direttamente verso la giusta porta di uscita tramite bus*. **Il bus può divenire collo di bottiglia** dal momento che due pacchetti, anche indirizzati a porte di uscita diverse, non possono attraversare contemporaneamente il bus. *Con bus a 32Gbps comunque buone soluzioni per router domestici*.
-**Commutazione attraverso rete di interconnessione**:
Esistono diversi approcci. Uno è attraverso **crossbar switch**: ogni porta di ingresso ha bus dedicato e si hanno *nxn switch* per interconnettere i collegamenti (dove n è numero di porte ingresso e uscita). ***Grazie alla crossbar è possibile trasmettere pacchetti destinati a linee di uscita diverse assieme***.
Tuttavia la crossbar è molto costosa (nxn switch), quindi tendenzialmente si usano approcci simili ma con meno switch e il compromesso per cui alcune linee potrebbero incrociarsi (es. **multistage switch**).

Parlando dell'accodamento d'ingresso anche in questo caso (come visto per HTTP) può presentarsi la problematica dell'**HOL** (***head of line blocking***).
Immaginiamo di avere due pacchetti provenienti da linee di ingresso diverse. Uno blu sopra e uno rosso sotto. Non posso far transitare il rosso che deve arrivare alla linea di uscita in alto perché sta transitando il blu che passa nel mezzo. La cosa peggiore: dietro il rosso c'è in coda un pacchetto verde, che potrebbe intanto tranquillamente essere trasferito (il pacchetto verde in HOL).



Accodamento in uscita

Abbiamo visto come la struttura di commutazione trasferisca pacchetti alla linea con un ***tasso di almeno NxR***. Tuttavia il router, nella sua terminazione di linea in uscita, può inviare pacchetti a un tasso R. Capita quindi che arrivino in

uscita dalla struttura di commutazione più pacchetti di quanti il router ne possa inviare. Pertanto è necessario:
Buffering: *un buffer che contenga i pacchetti in arrivo dalla struttura di commutazione ancora non inviati.* ***Se il buffer si riempie eccessivamente congestione e perdita di pacchetti***.
Disciplina di scheduling: *si scelgono quali datagrammi in coda di uscita devono essere trasmessi*.

Ma quanto deve essere grande il buffer? Anzitutto **chiariamo il fatto che un buffer enorme non è la soluzione anzi creerebbe problemi: RTT sempre più elevati, problemi per applicazioni sensibili al fattore tempo, TCP meno reattivo alla congestione**.
L'idea è quindi trovare una dimensione ideale in base al valore di RTT e alla capacità del collegamento di uscita C. Studi più recenti ha mostrato che dati N ingressi la dimensione buffer consigliata è **RTT x C / rad(N)**.

In caso di overflow quali pacchetti devo scartare? Esistono due approcci: **Tail Drop** in cui ***scarto il pacchetto che ha trovato la coda piena*** e **Priorità** in cui ***i pacchetti sono marcati con una certa priorità*** e scarto quello con priorità più bassa. *Ciò è utile per far passare pacchetti con l'obiettivo di segnalare la congestione*.

In che ordine invio i datagrammi nella coda?

Diversi approcci:
-**FIFO (first in first out)**: *invio il pacchetto arrivato prima in coda*
-**Schedulazione con priorità**: *ho più code, in ogni coda inserisco solo pacchetti con una certa priorità*. Mando di volta in volta il pacchetto nella coda con priorità più alta non vuota (*un pacchetto di bassa priorità non potrà essere inviato fintanto che una coda con classe più alta non sarà vuota*)
-**Round Robin RR**: *sempre più code in base alla priorità, ma stavolta il collegamento non scandisce solo dall'alto verso il basso ma cicla*.
-**Weighted Fair Queueing WFQ**: **generalizza Round Robin**. Non mi limito a ciclare inviando un pacchetto per coda, ma *cerco di garantire maggiore servizio a pacchetti con priorità più alta*. Ciascuna classe i ha peso wi e riceve quantità ponderata di servizio ad ogni ciclo pari a wi/sum(wj) (somma per ogni coda del suo peso)

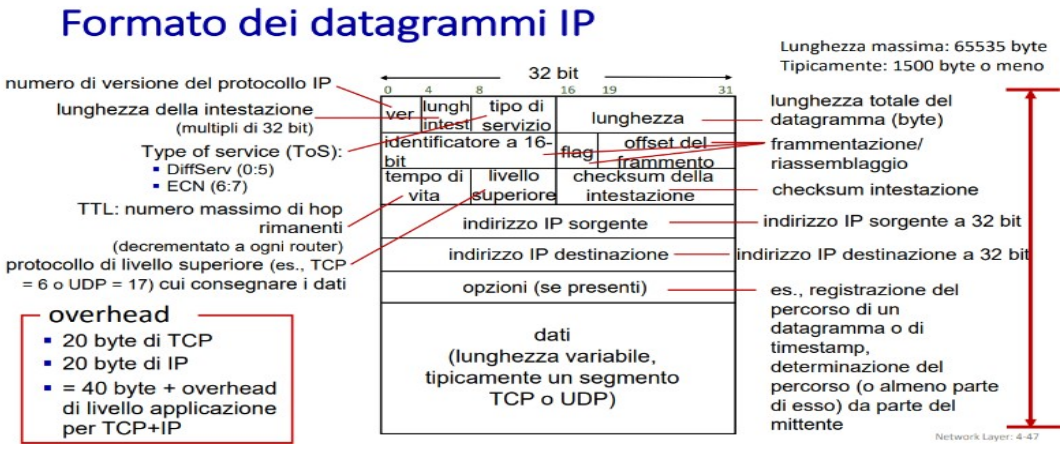
15.

Tre componenti principali per l'implementazione del livello di rete: **Algoritmi di instradamento** (eseguiti nei router oppure via SDN, il loro output sono le tabelle di inoltro locali ai router), **protocollo IP** e **protocollo ICMP**.

Protocollo IP

Sappiamo che in generale *un protocollo deve definire il formato dei messaggi e le azioni che devono essere intraprese quando invio un messaggio, lo ricevo oppure avviene altro*.

A livello di rete il protocollo principalmente utilizzato è **IP** (internet protocol). Insieme a TCP è il protocollo più importante di Internet.
IP definisce come sono strutturati i datagrammi, gli indirizzi e come devono essere manipolati i pacchetti.



Ver: 4 bit dedicati a indicare ***la versione di IP utilizzata*** (nel nostro caso IPv4, ma ve ne sono altre come IPv6)
Lungh. Intestazione: ***4 bit a multipli di 32***. Poiché un datagramma ha intestazione di almeno 20 byte, 160 bit, 32x5 e quindi sempre almeno 0101 in questo campo
TypeOfService: si divide in due parti, **DiffServ** *per utilizzare un modello di servizio diverso dal* ***best effort***, l'altra parte dedicata al **controllo di congestione**.
Sappiamo infatti che *per TCP esistono due approcci per controllare la congestione*: **tramite sintomi** (ritardi, perdite) e **tramite esplicita notifica da parte della rete ECN** (explicit congestion notification).

Vi sono *2 bit dedicati ad ECN*, uno per indicare che l'host lo supporta e l'altro viene messo ad 1 quando il datagramma passa per un router congestionato. Quando il datagramma arriva al destinatario questo si accorge dei due 1 nell'**ECN**, imposta quindi valore 1 al campo **ECE** (explicit congestion echo)

nel datagramma di ACK. *Quando il mittente lo riceve dimezza la finestra di congestione cwnd* e imposta ad 1 il bit **CWR** (congestion window reduced)
Lunghezza: rappresenta la lunghezza effettiva del datagramma, **in multipli di 8** bit (1 byte). *Vi sono 16 bit dedicati* per un totale di 2^16-1 == 65536 byte, ma tendenzialmente causa MTU maximum transmission unit a livello di collegamento non superiore a 1500 byte.
TTL TimeToLive: rappresenta il numero di hop rimanenti al pacchetto prima di venire scartato. Necessario perché potrebbero crearsi dei cicli in rete per cui il pacchetto continua a circolare sprecando risorse, inoltre se arrivasse ad un destinatario con esattamente lo stesso numero di sequenza che si aspetta allora accetterebbe un pacchetto sbagliato. Quindi TTL per evitare spreco di risorse e garantire affidabilità TCP.
Checksum: implementata anche a livello di rete, perché farlo? Potrei avere protocolli superiori che non la implementano. Checksum fatta solo sull'intestazione, il problema è che con TTL cambia sempre e quindi deve essere ricalcolata in ogni router attraversato. Ma la struttura di commutazione vuole essere più veloce possibile (nanosecondi) e ciò è costosto, infatti in IPv6 rimossa la checksum
Livello Superiore: identifica il protocollo a livello di trasporto a cui si fa riferimento, necessario per multiplexing/demultimplexing
IP Mittente/Destinatario (16 bit e 16 bit)
Opzioni: bit variabili, senza di esse 20 byte
Identificatore a 16 bit, Flag e Offset per la frammentazione

Frammentazione IPv4

Come detto **Identificatore a 16 bit**, **Flag** e **Offset** sono campi dedicati esclusivamente alla **frammentazione**. *IPv4 frammenta un datagramma dal momento che questo eccede in dimensione la MTU* (che è la massima dimensione del payload di un frame). Se ho una MTU di 1500 byte e voglio mandare un datagramma da 4000, questo viene frammentato in 3 datagrammi più piccoli di dimensione 1500, 1500 e 1000. Ci sono da considerare anche i 40 byte di intestazione in più per i nuovi datagrammi, quindi 1040 byte nell'ultimo datagramma di cui 20 di intestazione.

L'idea è che il mittente si occupi di riunire questi frammenti nel datagramma originale. Come fare? Si utilizza anzitutto ***l'identificatore a 16 bit per capire che più frammenti fanno parte dello stesso datagramma***. Per ***capire poi l'ordine in cui riunirli si guarda l'offset*** (se 0 primo frammento, se offset = dimensione del primo frammento allora secondo frammento etc…).

L'offset è espresso in multipli di 8 bit. Si ha poi il campo ***flag***, sono 3 bit: **R** riservato sempre a 0, **DF don't fragment** per identificare il fatto che mi trovo di fronte ad un frammento da *non riframmentare*, **MF more fragments** per capire se ci sono altri frammenti dopo di me e *identificare quindi l'ultimo frammento (MF = 0)*.

La frammentazione è un processo complesso che è costoso, soprattutto per la struttura di commutazione dei router che deve lavorare nell'ordine dei nanosecondi, **infatti con IPv6 frammentazione eliminata**.
Si preferisce un approccio per cui **si limita la dimensione del datagramma** e si usano **algoritmi di path MTU discovery** per capire la dimensione di MTU minima nel percorso mittente-destinazione e regolare così la dimensione dei datagrammi.

Un semplice meccanismo per conoscere l'MTU nel collegamento di interesse è *l'invio di pacchetti* ***con bit DF don't fragment impostato a 1****, se il router non può mandarlo perché eccede l'MTU allora lo scarta e invia al mittente il messaggio* ***ICMP*** *"Destination Unreachable: Fragmentation Required".*
Tuttavia ICMP talvolta bloccato per motivi di sicurezza, algtri algoritmi di Path MTU Discovery più complessi ma robusti

Indirizzamento IP

Un indirizzo IP è un valore a 32 bit che è *associato a un'interfaccia* di un host/router.
Un'interfaccia rappresenta la connessione dell'host/router al collegamento fisico. Quindi un indirizzo IP non identifica un host/router in sé, quanto la sua interfaccia.
Tipicamente i router hanno molte interfacce (più collegamenti in uscita) *mentre gli host ne hanno 1 o 2* (ethernet cablata e/o wireless).

Per interconnettere più interfacce si deve far riferimento al livello di collegamento e fisico! Il livello di collegamento infatti permette la interconnesione tra le varie interfacce tramite l'ausilio di **switch**, che sono dispositivi che come i router si occupano della ***commutazione dei pacchetti ma solo fino al livello di collegamento***.
Quindi è possibile connettere più host senza l'ausilio di router ma solo

tramite switch, che ne collegano le interfacce.
Gruppi di host di questo tipo, connessi solo da switch e non da router, fanno parte di una sottorete (subnet).

Gli host parte di una stessa sottorete hanno tutti lo stesso prefisso.
Gli IP hanno quindi una struttura divisa in due parti:
-**Parte della sottorete**: prefisso che identifica la subnet
-**Parte dell'host**: i rimanenti bit che identificano l'host

Sottoreti diverse devono avere prefissi diversi.

Per identificare una sottorete devo immaginare di rimuovere tutti i router, i collegamenti rimanenti rappresentano una specifica sottorete (*in quest'ottica anche lo spazio tra due router rappresenta una sottorete*).

Per identificare l'indirizzo di una subnet si utilizza la notazione **CIDR**.
Questa notazione è del tipo a.b.c.d/x dove a,b,c,d sono ognuno un byte e x è il numero di bit appartenendi al prefisso di sottorete.
Es. 223.1.2/24 indica che di questo indirizzo IP i primi 24 bit identificano la rete, gli altri sono variabili per identificare gli host parte di quella subnet.

Per capire quanti possibili host posso avere nella subnet porto il valore in binario:
11011111.00000001.00000010.00000000

In questo caso 2^8-1 possibili router nella subnet, a eccezione di:
-*tutti 0 nei bit degli host* **identifica la subnet in sé**;
-*tutti 1 nei bit degli host* **identifica il'indirizzo di broadcast diretto**.

Prima di CIDR si mettevano a 1 tutti i bit legati al prefisso di subnet per identificarla, si parlava di maschera di sottorete **subnet mask**.
Ora un indirizzo IP di una subnet rappresentato in CIDR è comunque detto subnet mask.

Dato IP specifico e una maschera posso ottenere l'effettivo prefisso di rete facendo l'AND tra i due.

Broadcast diretto == ***mettere 1 solo ai bit dedicati agli host della sottorete, identifica il fatto che mando il pacchetto a tutti gli host della sottorete in questione***. Tendenzialmente possibilità ristretta poiché ci si potreebbe costruire un attacco di DoS.
Broadcast limitato == ***Tutti 1 (255.255.255.255) mi permette di inviare in automatico il datagramma a tutti gli host della mia stessa sottorete***.

Prima di Cider esisteva una classificazione degli indirizzi **classful addressive**, per cui gli indirizzi erano suddivisi in classi A,B,C,D,E.
Concentrandoci sulle prime 3, queste erano dedicate a subnet rispettivamente di 8 bit prefisso, 16 e 24. ***Il problema di questo approccio era che anzitutto si dovevano sprecare dei bit nel prefisso per identificare il tipo di classe*** (es. A 0, B 10, C 110) e inoltre classificare in questo modo non soddisfava tutte le possibili sottoreti, ***non era possibile dosare gli indirizzi dei possibili host nella sottorete perché erano prestabiliti in base alla classe di appartenenza***.

Sysadmin, DHCP

Come fa un host a ottenere un IP nella sua sottorete?

Due approcci: **1)** esiste nella rete un amministratore, il **sysadmin**, ***che decide l'assegnazione degli indirizzi IP*** e lo comunica all'host installandolo "a mano"
2) Si usa il protocollo **DHCP** (Dynamic Host Configuration Protocol) che consente all'host entrato in una sottorete di ***ottenere automaticamente l'IP e altre informazioni attraverso l'interazione con un server DHCP***.

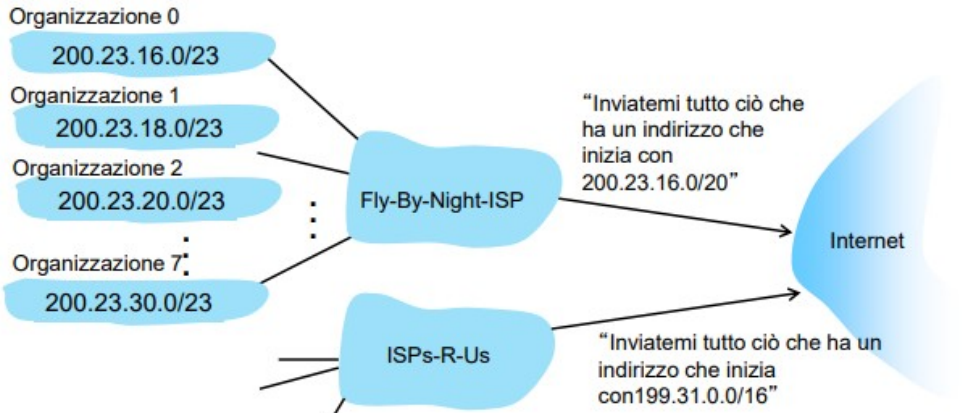
Questo protocollo permette di riciclare IP di host che se ne sono andati dalla rete e **la sua dinamicità supporta gli utenti mobili che si uniscono e abbandonano di continuo la rete**.
NB. *Nel momento che cambio una subnet non posso mantenere una connessione TCP attiva perché cambio IP di origine*!

Come funziona DHCP:
Il modem ADSL/Fibra (che funge da router, punto di accesso, modem etc…) ingloba anche il server DHCP.
Quattro fasi per ottenere l'IP:
1-l'host appena connesso alla subnet ***manda in broadcast limitato*** 255.255.255.255 un messaggio di **DHCP discover** per individuare i server DHCP
2-Il server DHCP riceve il messaggio e risponde con un messaggio **DHCP offer** sempre in broadcast, *con annesso un campo con l'indirizzo IP "offerto"*
3-L'host, poiché potrebbe ricevere più offerte da vari DHCP, invia un messaggio di **DHCP request** in cui *allega l'IP specifico di modo che il server DHCP capisca che il messaggio in broadcast è indirizzato a lui*
4-Il server DHCP *se l'IP è ancora disponibile* risponde con un **DHCP ACK**: l'host ottiene finalmente l'IP della subnet.

Come fa una subnet ad ottenere il proprio indirizzo?

<u>Ottiene l'assegnazione di una porzione dello spazio d'indirizzi dal suo provider ISP</u>.
L'ISP infatti *possiede un prefisso di indirizzi e da questo, estendendolo, può ricavare blocchi di indirizzi che può assegnare ad host diversi*.

Si ha un vero e proprio **<u>indirizzamento gerarchico</u>** in termini di ISP: si parte da IP con prefisso minore che portano i pacchetti ad arrivare nel dominio dell'ISP e da lì, riducendo il prefisso, verso le sottoreti specifiche (organizzazioni) che appartengono a quell'ISP!



127.0.0.1 altro indirizzo speciale associato all'interfaccia di loopback (una sorta di interfaccia connessa a se stessa che permette la comunicazione tra processi diversi della stessa macchina)

<u>16.</u>

I 2^32 indirizzi disponibili per IPv4 sono circa 4kk, non è abbastanza per gli standard attuali di Internet!
Due modi per risolvere il problema: **<u>NAT</u>** e **<u>IPv6</u>**

NAT Network Address Translation

<u>L'idea su cui si basa l'approccio NAT è la volontà di far sì che gli host esterni alla mia sottorete identifichino gli host della mia subnet tutti con lo stesso indirizzo IP.</u>
Ciò offre **molti vantaggi**, primo fra tutti in termini di ***<u>sicurezza</u>***: grazie a quest'approccio di norma un host esterno non sarà in grado di connettersi direttamente con un server della mia sottorete.
Altro vantaggio risiede nella ***flessibilità in termini di cambiamento di IP*** nella mia subnet, con NAT posso cambiare IP dinamicamente senza problemi poiché

agli occhi esterni resterà sempre lo stesso e quindi non sono costretto a comunicarlo all'esterno.

Inoltre **ha senso utilizzare NAT solo se gli indirizzi della mia sottorete sono privati**, ossia **unici esclusivamente a livello locale**.
In questo modo sottoreti diverse possono presentare stessi indirizzi IP, e qui entra in gioco il "risparmio" di indirizzi.
Alcuni indirizzi privati: 10/8, 172.16/12, 192.168/16.

Come viene implementato?
Immaginiamo che **un host con indirizzo privato** (es. 10.0.0.1) voglia inviare un pacchetto ad un host fuori dalla propria subnet. Per prima cosa, utilizzando il livello di collegamento, inoltra il pacchetto al router di bordo (router di primo hop, per l'accesso al nucleo della rete). Il router, leggendo il datagramma, si accorge che l'IP mittente è privato e quindi che deve attuare la tecnica NAT. L'idea è che il destinatario veda IP e numero di porta mittente come quelli NAT del router, e quindi il router li deve convertire. Tuttavia, in caso riceva una risposta dal destinatario, **il router deve ricordare IP e numero di porta mittente originari, e per far ciò utilizza una Tabella di traduzione NAT**.
*Il router salva nella tabella l'associazione tra **IP e numero di porta mittente** originari (lato **LAN** local area network) con **IP e numero di porta NAT** (lato **WAN** wide area network), e inoltra il pacchetto*.
Se viene ricevuta una risposta, *il router riconosce di essere il destinatario* e quindi, guardando la tabella, traduce l'IP e numero di porta NAT in quelli originali *inoltrando pacchetto al mittente corretto* nella subnet.

Il NAT è tipicamente implementato nei router di accesso a reti locali e quindi anche i nostri modem fungono tipicamente da NAT.

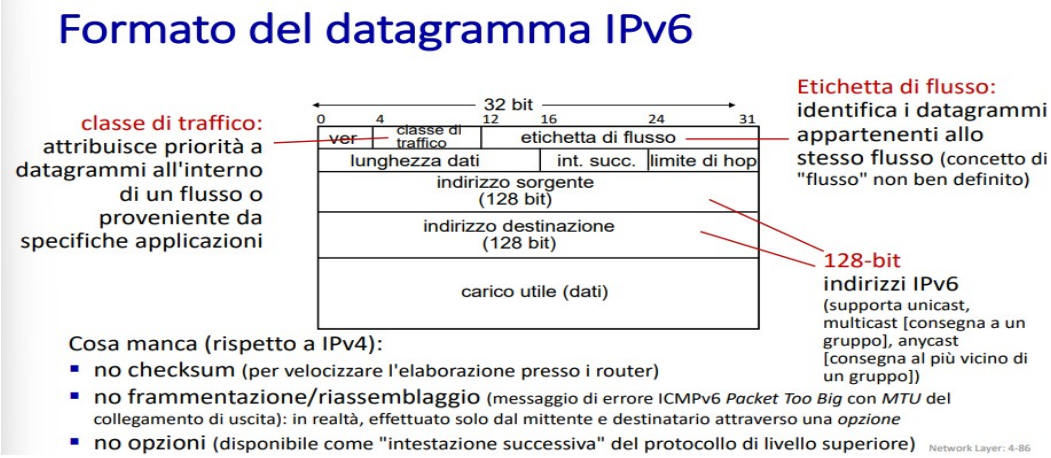
NAT tecnologia controversa poiché ***un router si dovrebbe spingere solo fino al livello di rete*** in termini di implementazione, ***ma se usa NAT arriva anche a livello di trasporto*** cambiando addirittura IP e numero di porta del datagramma e segmento (viola il principio punto-punto secondo cui la rete dovrebbe limitarsi a trasferire dati nel nucleo, senza decapsulare i datagrammi).

Se volessi installare un server accessibile a tutti ma ho connessione NAT allora è un problema, necessito di tecnologie ***NAT traversal*** per bucare la barriera del NAT.

IPv6

Inizialmente NAT era utilizzato per ovviare al problema della scarsità di indirizzi IP, ma con l'arrivo di IPv6 ciò non rappresentava più un problema. **IPv6 venne infatti introdotto per risolvere il problema dei pochi indirizzi IP e per velocizzare la velocità di elaborazione/inoltro (struttura di commutazione) dei router.**
Il primo problema è stato risolto semplicemente utilizzando ***indirizzi di lunghezza pari a 128 bit***, il secondo ***rendendo la PDU IPv6 molto più semplice di IPv4 eliminando*** campi come **checksum**, **frammentazione** (non propriamente vitali). Il campo delle **opzioni** non è più presente ma comunque implementato in modo diverso.

Inoltre si è introdotto il concetto di flusso come "**first class citizen**": mentre IPv4 si occupava solo di inoltrare i datagrammi, *IPv6 permette di riconoscere il fatto che una serie di datagrammi fanno parte di uno stesso flusso* (tramite un numero di 20 bit) e quindi offrendo loro servizi diversi (priorità).



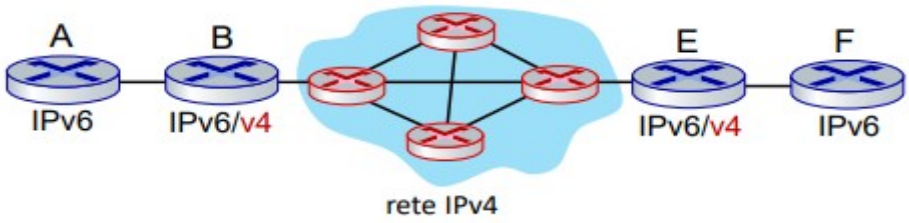
PDU IPv6:
Ver == versione di IP che sto utilizzando (nel nostro caso IPv6)
Classe di traffico == necessaria per ***garantire servizi particolari a specifici flussi di byte o datagrammi provenienti da specifiche applicazioni*** (es. multimediali)
Etichetta di flusso == necessario per ***identificare datagrammi appartenenti allo stesso flusso*** (20 bit)
Lunghezza == per identificare la lunghezza del payload del datagramma
Limite di hop == TTL
Intestazione successiva == per identificare quale protocollo sto utilizzando a livello di trasporto (multiplexing/demultiplexing)
Indirizzo IP sorgente/destinazione == ***128 bit*** na fraccaa

Le opzioni che erano di lunghezza variabile sono state rimosse al fine di garantire lunghezza costante all'intestazione TCP.
Tuttavia *sono implementate in modo particolare incapsulando il segmento in un'intestazione in cui è possibile inserire le opzioni di lunghezza variabile incapsulato nel datagramma effettivo*! (**opzioni tra livello di rete e livello di trasporto**)

Tunneling

Il passaggio da IPv4 a IPv6 ***è in corso da oltre 20 anni*** per via del fatto che si lavora nel nucleo della rete ed è impossibile aggiornare tutti i router insieme.
In generale, **cambiamenti a livello di rete (nel nucleo della rete) sono molto più difficili e lenti, mentre a livello applicativo più "semplice" e veloce**.

Attualmente una delle soluzioni per ovviare al problema è quella del **tunneling**.
L'idea di base è sostanzialmente trasportare datagrammi IPv6 come payload di datagrammi IPv4 passando per router che non supportano IPv6.



B è consapevole della rete IPv4 e del fatto che E è il primo router che implementa IPv6. Quindi B, con in mano un pacchetto IPv6, lo incapsula in un pacchetto IPv4 con destinazione E. Il pacchetto viene inoltrato per la rete IPv4 fino a raggiungere E, che vedendosi come destinataria lo spacchetta e si ritrova in mano il pacchetto IPv6 di cui può continuare normalmente l'inoltro.

Si parla di tunnel perché logicamente è come se il pacchetto arrivasse direttamente ad E, ma in realtà passa nel mezzo della rete IPv4.

Inoltro generalizzato

L'inoltro tradizionale basato su destinazione prevede che in base all'indirizzo IP del datagramma, confrontandolo nella tabella di inoltro, so a quale linea di uscita del router inviarlo.
Questo inoltro è basato su un paradigma detto **match plus action** (trovi associazione -> fai qualcosa).
Si può utilizzare questo paradigma per introdurre **l'Inoltro Generalizzato**, per cui ***il match non è più legato solo all'indirizzo IP di destinazione ma a più***

campi di intestazione (anche di livelli diversi!) e le actions possono essere molteplici (scarta/modifica/copia etc.. il pacchetto).

Quando si parla di inoltro generalizzato la tabella è detta **Tabella dei Flussi** e non si parla più di semplici router ma di **packet switch**.
La tabella di flussi definisce regole per cui in base a uno specifico match si arriva a un'azione specifica:

Match == *corrispondenza con valori dei campi di intestazione*
Actions == se match posso scartare (**drop**), modificare (**modify**), inviare al controllore (**log**), inoltrare (**forward**) etc…
Priorità == *se vi sono pattern sovrapposti vi sarà modo di dare priorità all'uno o all'altro*
Contatori == necessari per **statistiche** (es. quanti pacchetti hanno fatto scattare una regola, quando è avvenuto l'ultimo aggiornamento etc..)

OpenFlow

Standard per inoltro generalizzato reso popolare dall'SDN.

OpenFlow fornisce infatti un protocollo che permette al controllore di interagire con i router per configurarli.

Si vede come si tratta di inoltro generalizzato **per via del matching che dipende da pattern d'intestazione differenti definiti anche a più livelli**. Su questo OpenFlow possiamo prendere decisioni sulla base di 12 campi diversi. Vediamo qualche esempio di azioni corrispondenti a certi pattern.

Un packet switch può essere un dispositivo diverso in base alla funzione match/action che gli si vuole attribuire.
Es. router match IP action inoltro, switch match MAC action inoltro, Firewall

match IP e numeri di porta action consentire o negare il passaggio, NAT match IP e porta action riscrive l'indirizzo e la porta.

È importante ribadire come più in generale esista una **logica di rete** globale, *per cui pacchetti devono essere inviati da un host mittente ad uno destinatario*, di cui il controllore è "consapevole". *I packet switch invece non ne sanno nulla, sta proprio al controller definire le regole affinché tutto funzioni correttamente*.

La logica di rete globale è "istanziata" nelle tabelle di flusso affinché i packet switch facciano il loro lavoro localmente, organizzando "inconsapevolmente" questi percorsi.

Un packet switch è tale dal momento che, presentando una tabella di flusso, in base a un matching definito da specifiche regole si ha una certa action. Queste regole sono a tutti gli effetti una forma di programmazione sempre più in voga e estesa, con linguaggi come **P4**.

Middlebox

Middlebox == ***qualsiasi dispositivo nel percorso da host origine a destinazione che svolge una funzione diversa da quella normale di un router IP***.

Si tratta quindi di ***dispositivi che lavorano nel nucleo della rete ma che non si occupano del semplice inoltro***. Ve ne sono tantissime: **Firewall**, **NAT**, **Load Balancer**, **Cache**, **Application Specific** (fornitori di servizi) etc…

Con l'SDN abbiamo disaccoppiato il piano di controllo dal piano di dati, trasformando il piano di controllo in software in esecuzione su server remoti. Una logica simile di disaccoppiamento si ha nella **NFV** (**Network Functions Virtualization**). L'idea alla base di questo approccio è di **eseguire le funzioni di rete (es. router, switch, firewall) utilizzando hardware "generico"** (COTS) tramite VM o container sfruttandone le risorse di calcolo e lo storage.

Inizialmente il livello di rete offriva il servizio minimale di inoltro dei pacchetti da un router ad un altro nel nucleo, con l'arrivo delle middlebox e l'inoltro generalizzato ***sempre più funzioni all'interno del nucleo di rete***!

Principi architetturali di Internet:

- **Connettività semplice** (trasferimento di datagrammi tra host);

- **Protocollo IP** (offre omogeneità a livello di rete e nasconde l'eterogeneità sottostante);
- **Intelligenza, complessità alla periferia** (edge) della rete.

Implementare e fare manutenzione di programmi in rete è molto più complesso di quanto non sia farlo alla periferia, si pensi alle rivoluzioni a livello applicativo degli ultimi anni e al tempo invece impiegato (e ancora in corso) nel passaggio da IPv4 a IPv6.
L'argomento end-to-end suggerisce proprio di implementare questo tipo di funzionalità all'estremo, e non nel mezzo.
Ha senso talvolta "completare" queste funzionalità a livello di rete per migliorare le prestazioni.

Oggi l'intelligenza si trova non solo nella periferia, ma anche nel nucleo della rete(seppure ridotta rispetto alla periferia) (abbiamo visto middlebox, dispositivi di rete programmabili NFV etc..).

18.

Affinché la funzione di inoltro basata su destinazione (piano di dati) funzioni correttamente è necessario *scrivere adeguatamente le tabelle di inoltro* per ogni singolo router. Come fare? Entra in gioco la **funzione di instradamento**, basata sugli ***algoritmi di instradamento*** (**piano di controllo**) che, con una visione più larga della rete (***logica di rete***) permettono la scrittura nei vari router di tabelle di inoltro corrette.
Inoltro == *spostare pacchetti da una delle linee di ingresso alla corretta linea di uscita* (dati)
Instradamento == *identificare il percorso corretto che il pacchetto deve seguire nel nucleo della rete affinché giunga al destinatario* (controllo)

Le tabelle di inoltro sono quindi scritte in funzione del piano di controllo che è conscio della logica di rete, quindi inoltrando per ogni router (che è conscio solo dei next hop, non della rete in sé (piano di dati)) è resa valida anche la funzione di instradamento.

Due approcci per instradamento come già anticipato:
-**Tradizionale** == ogni router implementa di per sé il piano di controllo e quindi esegue gli algoritmi di instradamento da sé
-**SDN** **Software Defined Networking** == server remoto a cui sono connessi dei router si occupa di eseguire gli algoritmi di instradamento e di aggiornare le

tabelle dei router
(***logicamente centralizzato il server, ma in realtà sistema distribuito***)

In ***SDN il server non è veramente centrallizzato per questioni fault tolerance e affidabilità***. Se da una parte un processo muore dall'altra ce n'è una copia che garantisce il continuo funzionamento del sistema. Inoltre la distribuzione del carico permette un aumento di capacità del sistema.

Nel caso dell'approccio tradizionale affinché ogni router esegua gli algoritmi di instradamento è necessario che comunichi con i suoi vicini. Per farlo esistono protocolli specifici detti **protocolli di routing**. Le tabelle sono scritte direttamente dal router per il router.

Nel caso dell'approccio SDN il piano di dati funziona esattamente come per l'approccio tradizionale, per quel che riguarda la funzione di instradamento **gli algoritmi sono eseguiti da processi remoti** e **le tabelle di instradamento aggiornate da un controllore remoto** che le calcola e installa nei vari router ad esso connessi.

Algoritmi di Instradamento

Il principale obiettivo di un algoritmo di instradamento è ***determinare un buon percorso da sorgente a destinazione attraverso il nucleo della rete***.
Ma cosa si intende con "**buon percorso**"?
Anzitutto si astrae il nucleo della rete ad un **grafo** pesato in cui i nodi sono i router e gli archi i collegamenti tra di essi. In base al peso che affidiamo ad un collegamento determiniamo il concetto di "buon percorso".

Se ad esempio volessimo un percorso basato sul concetto per cui impiego poco tempo ad arrivare a destinazione, peso gli archi in termini di tempo necessario ad attraversare un collegamento.
Infatti il costo di un percorso coincide con la somma dei pesi dei collegamenti attraversati.
Altri vincoli che si potrebbero porre sono ad esempio quelli per cui non si vuole far passare i pacchetti per router sotto il dominio di specifici ISP/Paesi (immagina tempi di guerra).
Tre parametri per definire gli algoritmi di instradamento:
- **Grado di visibilità della rete**: rappresenta il tipo di informazioni che utilizzo per calcolare i vari percorsi.
Due tipologie: **Globali** == a partire da una singola sorgente ricavo le distanze verso tutti gli altri router. ***Approccio basato sulla conoscenza completa della***

topologia e costo dei collegamenti in rete, gli algoritmi che ne fanno parte sono detti ***algoritmi link state*** (es. Dijkstra).
Decentralizzati == vengono calcolati iterativamente i percorsi da un router verso tutti gli altri ***partendo però dalla sola conoscenza del costo del collegamento verso router vicini***, gli algoritmi che ne fanno parte sono detti ***algoritmi distance vector*** (es. Bellman Ford)
- **Velocità di cambiamento dei percorsi**: algoritmi **statici** per cui i percorsi sono aggiornati lentamente (ad esempio con intervento umano) e **dinamici** per cui vengono aggiornati più velocemente *anche in risposta a cambiamento di topologia e costi di collegamento*
- **Sensibilità al carico**: algoritmi **sensibili al carico** se determinano i percorsi in funzione dell'attuale livello di congestione verso uno specifico router, **insensibili al carico** se non lo fanno.
Attualmente meglio insensibili perché con sensibili cambio da router congestionato a non -> cambiano tutti -> nuova congestione -> cambio ancora **Quindi oscillazione tra percorsi e instabilità**.

Dijkstra

È algoritmo **link state**, quindi **globale** (***la topologia e il costo dei collegamenti nella rete sono noti a tutti i nodi***). *Queste informazioni sono ottenute tramite un* ***algoritmo di link state broadcast***. Tutti i nodi hanno le stesse informazioni.

Vengono calcolati i percorsi di costo minimo da un nodo sorgente a tutti gli altri nodi (fornendo quindi la tabella di inoltro a quel nodo).
Si tratta di un algoritmo iterativo: infatti *dopo k iterazioni si conoscerà il cammino minimo verso k destinazioni*.

Dijkstra costruisce anche l'albero dei cammini minimi tenendo traccia del predecessore, in caso di pareggi si può risolvere arbitrariamente.

Dijkstra costa O(mlogn) con coda di priorità. Per inviare in broadcast tutte le informazioni (algoritmo globale) un nodo impiega O(n), per ogni nodo O(n^2).

Bellman Ford

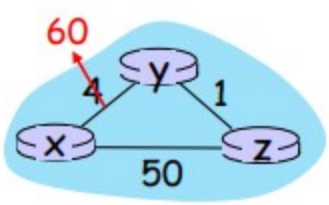
Algoritmo **distance vector**, ossia **decentralizzato** (***iterativamente calcola i percorsi verso gli altri nodi conoscendo inizialmente solo il costo del collegamento ai vicini***).

Si basa sull'equazione di programmazione dinamica per cui:
dx(y) = minv{cx,v + dv(y)}

Il valore calcolato è distanza stimata detta **DV**.
Quando un router cambia DV verso un altro nodo lo comunica ai suoi vicini, che cambiano il proprio DV di conseguenza (e rimandano DV aggiornato ai vicini).
Andando avanti con le iterazioni si arriva a far convergere la stima DV alla distanza effettiva verso quel nodo specifico.

Ciò che bisogna ricordare **che differenzia gli algoritmi link-state da quelli distance-vector** è che nel primo (Dijkstra) vi è una parte iniziale di **flooding** in cui ogni router annuncia gli stati dei collegamenti a tutti gli altri router (quindi ogni router consapevole della situazione di rete!)

Nel caso di Bellman Ford un router non conosce invece la topologia della rete, ma solo il costo che i vicini hanno verso determinate destinazioni.
Ciò può comportare dei problemi: **conteggio all'infinito**.



Immaginiamo che il collegamento da y a x passi da pesare 4 a 60. Ma allora y, guardando le distanze di z da x, si accorge che quest'ultimo dista da x 5 (poiché z passa per y costo 1 e verso z costo 4, ancora non aggiornato). Ma allora y aggiorna la sua stima verso x a 1+5 = 6. Poi z vede che la distanza verso x da y passa da 4 a 6, aggiorna la sua stima da 5 a 1+6 = 7 e così via, fino a che z non considera più conveniente il collegamento per y scegliendo quindi il collegamento diretto di costo 50.

Un modo per risolvere il problema è l'utilizzo **dell'inversione avvelenata**, per cui ***ogni nodo comunica al suo nodo adiacente attraverso cui costruisce un cammino minimo verso un altro nodo che la distanza verso quel nodo è infinito***.
Es. z comunica ad y che per arrivare a x impiega infinito, quando in realtà impiega 5 passando per y. In questo modo non inizia il circolo vizioso.

Tuttavia l'inversione avvelenata funziona solo per risolvere cicli che riguardano nodi adiacenti.
Quindi, **per evitare problemi, l'idea è di trovare un modo efficace per**

rappresentare l'infinito affinché non si compiano troppe iterazioni prima di stabilizzare la rete.

Un altro svantaggio legato agli algoritmi distance vector, ***oltre al tempo impiegato a stabilizzarsi*** in casi come quello visto adesso, ***è la possibilità che un router comunichi informazioni errate***. In particolare, se un router dice di avere un cammino di costo bassissimo verso qualsiasi destinazione, allora tutti ci passeranno -> **buco nero**.

Mentre LS se comunica un costo sbagliato questo è legato solo ad uno dei suoi collegamenti, con DV si sputtana tutto perché è legato a un percorso intero. Con LS infatti ogni router calcola solo la propria tabella mentre con DV la stima di un router è usata anche da tutti gli altri

Con LS il tempo per calcolare le distanze è al più O(n^2), con DV tempo di convergenza variabile (può anche convergere molto lentamente causa conteggio all'infinito)

Con LS i messaggi inviati sono O(n^2) (flooding, link state broadcast) mentre con DV si scambiano messaggi solo tra router adiacenti.

19.

La trattazione di rete vista con gli algoritmi di instradamento è stata piuttosto idealizzata, dal momento che abbiamo considerato internet come una sorta di unico e grande grafo.

Questo non è ciò che avviene nella realtà: applicare un singolo algoritmo di

instradamento su tutto internet comporterebbe **diversi problemi** tra cui:
- non **scalabilità**: milioni e milioni di router da prendere in considerazione, miliardi di possibili destinazioni. ***Risolvere gli algoritmi di instradamento richiederebbe decisamente tempo inaccettabile*** (link state flooding eccessivo, distance vector tempo di convergenza eccessivamente lungo)
- mancanza di **autonomia amministrativa**: un **amministratore di rete** (es. ISP) potrebbe voler g***estire il proprio dominio di rete con specifici algoritmi di instradamento, e potrebbe non volere che router appartenenti ad altri domini conoscano la topologia della propria rete***.

Per risolvere questi problemi *Internet è suddiviso in più piccole reti*, dette **AS Autonomous System** o domini.
Ogni AS comprende tipicamente router gestiti dalla stessa amministrazione.
Gli ISP più grandi possono avere più AS interconnessi tra loro.

L'instradamento a questo punto si divide in due tipi:
-**Intra-AS**: ***l'instradamento limitato ai router facenti parte dell'AS***.
È necessario che per ogni AS si utilizzi un algoritmo di instradamento specifico.
Ogni AS possiede un router particolarmente importante, detto ***router gateway*** (o di bordo) che permette all'AS di collegarsi ad altri AS.
-**Inter-AS**: ***l'instradamento di pacchetti da un AS ad un altro***, avviene grazie al router gateway. Quindi i gateway effettuano instradamento sia intra che inter-AS

Avere più AS permette chiaramente di risolvere il problema di scalabilità poiché *a questo punto gli algoritmi link state e distance vector faranno riferimento esclusivamente alla topologia e al peso dei collegamenti dell'AS in questione*.

Nasce adesso il problema di *come capire, a partire da un AS, verso quale AS adiacente propagare un pacchetto destinato a qualcuno fuori dal mio autonomous system*. ***Per fare ciò entrano in gioco sia l'instradamento intra-AS che l'instradamento inter-AS***.

- **Gli AS adiacenti al mio devono comunicarmi quali destinazioni sono raggiungibili attraverso di essi**, di modo che i miei router possano riempire le tabelle di inoltro correttamente. Queste informazioni dovranno essere quindi propagate a tutti i router del mio AS (**inter-AS**)

- **Per inviare nel pratico i pacchetti fino al router di bordo corretto e propagare le informazioni in arrivo** devo sfruttare uno specifico algoritmo di instradamento (**intra-AS**)

Per identificare gli algoritmi di comunicazione Intra-AS si parla propriamente di **protocolli di instradamento**, vediamone alcuni:
-**RIP** standardizzato fine anni ’80, distance vector (BellmanFord) col tempo sempre meno utilizzato causa problemi causati da questo tipo di algoritmi (*conteggio all’infinito, buchi neri, lungo tempo per convergere*);
-**EIGRP** == RIP raffinato
-**OSPF** == ***Open Shortest Path First***: protocollo **attualmente più utilizzato**, si basa su ***instradamento link state*** (**Dijkstra**) ed è preferito rispetto a DV perché, nonostante se implementato sensibile al carico comporta oscillazioni, non dà i problemi dati da Bellman Ford.

OSPF

Come detto è un protocollo che ***sfrutta link-state***. In quanto tale *è necessario che ogni router dell’AS conosca la topologia e lo stato dei collegamenti della rete affinché l’instradamento possa avvenire correttamente*, vengono quindi adottati algoritmi di link-state broadcast.
OSPF si distingue da un semplice Dijkstra per diverse peculiarità:
-**Sicurezza**: *i messaggi in OSPF sono autenticati* al fine di evitare intrusioni dannose
-**Ammette cammini minimi multipli**: salva nelle tabelle di inoltro più percorsi percorribili dallo stesso costo (se esistono), in tal modo si possono alternare per bilanciare il carico a più router (da fare attenzione perché con TCP si sforza così la necessità di riordinare i pacchetti
-**Suddivide l’AS in ulteriori aree**, ***in ognuna delle quali esegue flooding e Dijkstra***. In particolare tutte le aree sono dette **aree locali** eccetto una, **l’area 0** o **dorsale**, che le interconnette tutte.
Si distinguono quindi ***router di confine d’area*** (appartenenti al contempo ad un’area locale e alla dorsale), ***router di confine*** (coincidono con i router gateway visti prima), ***router locali*** e ***router di dorsale***.

Il router di confine d’area diffonde nell’area le informazioni apprese dalla dorsale sui percorsi da seguire per inviare pacchetti fuori dalla propria area e **invia alla dorsale le informazioni sulle destinazioni raggiungibili a partire dalla propria area.**

Quando un pacchetto deve essere inviato fuori dall'area si manda al router di confine d'area, che sa a chi mandarli (ha ricevuto le informazioni dalla dorsale, che le ha ricevute a sua volta dagli altri router di confine d'area).
Se voglio inoltrare un pacchetto fuori dall'AS passo direttamente per il router di confine.
Si possono avere più router di confine e di confine d'area, in tal caso la cosa si complica.

BGP

A differenza dei protocolli di intra-AS appena visti, *che possono anche essere diversi poiché si limitano alla visione del singolo AS*, quando si parla di **instradamento inter-AS per far sì che tutti gli AS possano comunicare tra loro è necessario l'utilizzo di un protocollo comune a tutti loro**.
Parliamo di **BGP**, questo protocollo permette ad ogni AS di:
- **eBGP** == *il router gateway del mio AS ottiene le* ***informazioni relative alle destinazioni raggiungibili passando per un altro AS*** *dal suo router gateway*, e *viceversa* gli comunico quali destinazioni il mio AS può raggiungere.
-**iBGP** == *a partire dal router gateway del mio AS vengono* ***propagate*** *queste informazioni a tutti i router del mio AS* ***affinché possano popolare correttamente le loro tabelle di inoltro***
-**Policy** == *posso decidere di evitare AS sotto il dominio di amministrazioni con cui non voglio avere a che fare* (economie ISP, guerra etc…)

I router gateway applicano sia protocollo eBGP e iBGP.

Due router gateway che comunicano tra loro via BGP sono detti **peer**.
La comunicazione avviene tramite una **connessione TCP semipermanente**, nell'arco della quale i due peer possono scambiarsi vicendevolmente le **rotte** (percorsi) ***verso diversi prefissi di reti di destinazione***. Questa comunicazione è detta **Sessione BGP**.

Una **rotta** annunciata da un peer è caratterizzata da:
- **prefisso** == ***la destinazione effettiva annunciata***
- **attributi**, tra cui **AS-PATH** == elenco degli AS per cui si dovrà passare fino a destinazione e **NEXT-HOP** == interfaccia del router che inizia il path.

Messaggi che due peer si scambiano sono del tipo:
OPEN == apre la connessione TCP col peer remoto e autentica il peer mittente
UPDATE == annuncia un nuovo percorso o ne rimuove uno vecchio
KEEPALIVE == mantiene attiva la connessione e usato per ACK di OPEN

NOTIFICATION == usato per chiudere la connessione e segnalare errori nello scorso messaggio

Inoltre, per la questione delle **policy**, ***un router può decidere se accettare/declinare un percorso verso una specifica destinazione*** e anche ***se non annunciare percorsi ad uno specifico AS vicino***.

Talvolta possono esistere più rotte verso lo stesso prefisso. In tal caso ciò che si fa è una delle seguenti cose (in ordine decrescente di priorità):
- **Uso un attributo di preferenza locale** (policy, scelgo personalmente quale rotta intraprendere)
- **Percorro l'AS-PATH più breve**
- **instradamento a "patata bollente"**: scelgo il percorso con next-hop router più vicino (voglio togliermi il pacchetto il prima possibile

Da questa gerarchia di scelte capiamo qual è la principale ***differenza tra instradamento intra e inter-AS***: con **intra-AS priorità assoluta trovare buoni percorsi nel mio AS** affinché i pacchetti siano instradati con efficienza, con **inter-AS priorità assoluta la policy adottata dal mio amministratore**.

In generale un ISP vuole trasferire solo i pacchetti che sono originati dai propri clienti o destinati ai propri clienti, non vuole occuparsi dei pacchetti di altro traffico (traffico di transito).

Per implementare queste politiche si possono "manipolare" gli annunci delle rotte.
Si può utilizzare CIDR per riassumere le destinazioni in prefissi più piccoli in modo da ridurre la dimensione delle tabelle di instradamento.

Perché diversi instradamenti Intra- e Inter-AS?

politiche:
- inter-AS: l'amministratore vuole avere il controllo sul modo in cui viene instradato il suo traffico, su chi passa attraverso la sua rete
- intra-AS: singolo amministratore, quindi le politicche sono meno problematiche

scalabilità:
- il routing gerarchico consente di ridurre le dimensioni delle tabelle e il traffico di aggiornamento.

prestazioni:
- intra-AS: può concentrarsi sulle prestazioni
- inter-AS: le politiche sono dominanti rispetto alla prestazioni

SDN

È il secondo approccio per l'implementazione del piano di controllo.
A differenza dell'approccio tradizionale dove si devono utilizzare dei ***router***

monolitici che implementano sia **piano di dati** (struttura di commutazione) sia **piano di controllo** (processore di instradamento***), con SDN si ha un'entità remota (logicamente centralizzata) che si occupa di eseguire gli algoritmi di instradamento e, tramite un controllore, di aggiornare le tabelle di inoltro***.

Vantaggi nell'uso di SDN:
-**maggiore flessibilità sul controllo del flusso del traffico** (è più semplice manipolare il traffico dei pacchetti, mentre con tradizionale devo ridefinire artificialmente i pesi sui collegamenti con SDN inoltro generalizzato e via)
-**si evitano errori di configurazione** (non devo implementare il piano di controllo su ogni router)
-**favorisce API, come OpenFlow, per l'inoltro generalizzato**
(più semplice installare tabelle di flusso via SDN piuttosto che farlo implementando ciò per ogni singolo router)

Per comprendere l'implementazione di SDN ***si può pensare al controllore remoto come una sorta di sistema operativo, sul quale sono in esecuzione diverse applicazioni*** (progr

ammabili) come routing, bilanciamento del carico, controllo di accesso…

Il controllore comunica con i router affinché possa comunicargli dati ricavati dall'esecuzione dei processi ***tramite un'interfaccia*** **Southbound** (che sfrutta protocolli come Openflow, che oltre all'inoltro generalizzato come API si occupa anche della comunicazione). *La comunicazione con i router via Southbound è importante anche affinché i router comunichino al controllore informazioni di stato utili ai processi per fare calcoli*.
La comunicazione tra processi e controllore avviene invece attraverso un'interfaccia **Northbound**, il controllore quindi fornisce dati e ne riceve altri ancora da questi ultimi.

Il vero e proprio cervello sul piano di controllo non è quindi propriamente il controllore ma i processi, che in realtà sono da lui scorporati.
Il controllore è implementato come un sistema logicamente centralizzato, ma in realtà è distribuito per questioni di **prestazioni**, **scalabilità**, **tolleranza ai guasti**, **sicurezza**, **robustezza**.

SDN è stato sviluppato prevalentemente nello scenario intra-AS, la prossima sfida è estenderlo per inter-AS e altri scenari particolari come ultra-

affidabilità, ultra-sicurezza, scenari real-time.
L'SDN è fondamentale per le reti cellulari 5G.

Esempio di interazione tra piano di dati e controllo via SDN:
Il router s1 ha un guasto su un collegamento, utilizza il messaggio di stato della porta OpenFlow per comunicarlo al controllore (interfaccia Southbound).
Il controllore riceve il messaggio, aggiorna le informazioni relative al collegamento e si attiva quindi automaticamente Dijkstra.
Dijkstra accede alle informazioni sul grafo della rete, sullo stato dei collegamenti e calcola i nuovi percorsi.
A quel punto il processo dedicato al routing interagisce con il componente flow table computation per calcolare le nuove tabelle di flusso, e una volta calcolate il controllore le installa nei packet switch via interfaccia southbound.

OpenFlow come protocollo *lavora tra controllore e switch* (Southbound), usa **TCP** per lo scambio di messaggi con crittografia opzionale e *non va confusa con l'API OpenFlow* che viene utilizzata per specificare le azioni di inoltro generalizzate.

Esistono tre classi di messaggi in OpenFlow:
- **Controller to switch**; include i seguenti messaggi chiave:

  - **Features**: il controllore invia richieste agli switch per venire a conoscenza delle loro caratteristiche;
  - **Configure**: vengono impostati i parametri di configurazione dello switch;
  - **Modify State**: per aggiungere o modificare il loro stato (ciò include l'installazione delle voci nelle tabelle di flusso);
  - **Packet out**: il controllore può inviare questo pacchetto da una specifica porta dello switch.

- **Asynchronous** (switch to controller); con i seguenti messaggi chiave:

  - **Packet in** è il messaggio con il quale lo switch può inviare al controllore un pacchetto per farlo ispezionare
  - **Flow removed** è una modifica con cui viene indicato che una voce nella tabella di flusso è stata cancellata (molto importante, lo abbiamo visto in Dijkstra: ogni switch deve comunicare lo stato di ciascuna porta al fine di garantire il corretto instradamento)
  - **Port Status**: informare il controllore di una modifica su una porta.

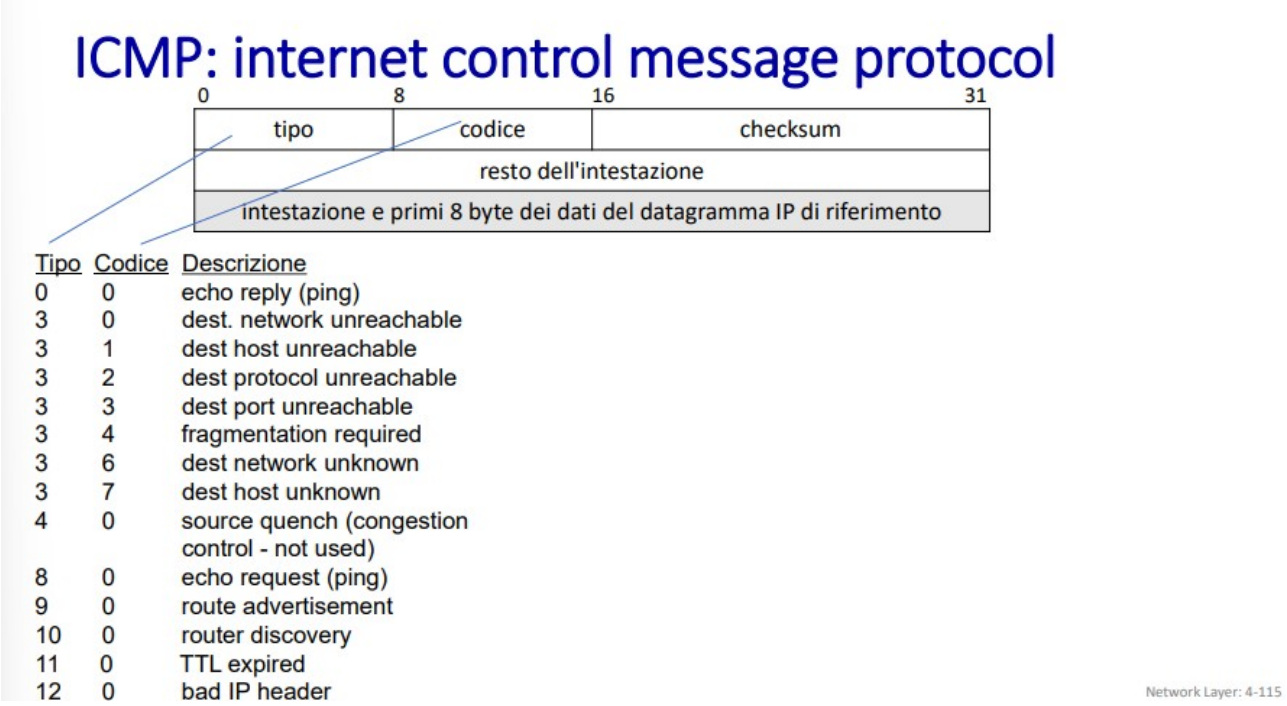
20.

Vediamo dei protocolli a livello di rete utili per scambiare informazioni quali diagnostica/configurazione/monitoraggio e informazioni di controllo.

ICMP

Si tratta di un **protocollo dedicato allo scambio di informazioni di rete**, come ***segnalazione degli errori*** e ***richiesta/risposta echo***.
Il messaggio ICMP è incapsulato nel datagramma così come lo sono i segmenti del livello di trasporto.
Tuttavia non consideriamo ICMP come protocollo a livello di trasporto poiché non svolge una vera e propria funzione di trasferimento quanto di comunicazione di informazioni specifiche.

PDU ICMP:



-**Tipo e Codice** utilizzati ***per implementare una classificazione a due livelli del messaggio ICMP*** (con tipo capisco a che tipologia appartiene l'errore, con codice qual è l'errore nello specifico)
-**Checksum classica**
-**Resto dell'intestazione** utilizzabile in base al tipo di messaggio
-**Altri 8 byte** dedicati alla ***copia dei primi 8 byte dei dati del datagramma di riferimento***: si utilizzano i primi 8 byte dei dati poiché il datagramma contiene nei dati il segmento a livello di trasporto, e nei primi 8 byte del livello di trasporto trovo i numeri di porta per capire gli specifici processi

Alcuni esempi importanti di messaggi ICMP:
3 4 ***FRAGMENTATION REQUIRED*** fondamentale per gli **algoritmi di Path MTU discovery** (metti Dont Fragment a 1 caso IPv4, quando un datagramma supera la dimensione della MTU viene reinviato al mittente il messaggio ICMP dove viene specificata la dimensione MTU che si è superata).

3 2 ***dest protocol unreachable*** e 3 3 ***dest port unreachable*** (protocollo e porta di destinazione non sono attivi)

8 0 ***echo request*** da cui mi aspetto una echo reply 0 0 che contiene le stesse informazioni. Con echo riempiamo tipicamente il resto dell'intestazione con un identificatore e un numero di sequenza e ponendo nei dati anche l'orario in cui l'ho spedito posso calcolare l'RTT quando torna a me.

11 0 ***TTL expired***

Lo potrei utilizzare per la **traceroute**, per capire quanti router ci sono nel cammino verso un destinatario. L'idea è di mettere come porta destinazione una "improbabile". Man mano aumento la TTL fin quando, invece di ricevere TTL expired, ricevo un 3 3 port unreachable

Gestione e configurazione della rete

Una **rete** (**AS**) è un insieme di migliaia di componenti software e hardware che interagiscono tra loro. Non è semplice farla funzionare, occorre un processo di ***gestione di rete*** per **coordinare e operare hardware e software, persone controllino e verifichino che la rete funziona come dovrebbe**.

Tutte queste funzioni sono oggi "integrate" nell'SDN: il controllore logicamente centralizzato infatti, tra le altre cose, monitora anche la topologia della rete e lo stato dei collegamenti.

Vediamo alcuni protocoli usati prima di SDN ma **ancora comunque attuali** poiché inclusi come API southbound per ottenere informazioni per il controllore.

Una rete è costituita da **sistemi periferici, switch e router**. Si tratta in toto di apparati hardware e software che voglio poter gestire ed eventualmente configurare (**dispositivi di rete gestiti**).
Ciò che permette la gestione dei dispositivi è la presenza di un **agente** in ognuno di essi che interagisce con un **server di gestione**.

Quando il server interagisce con l'agente osserva sostanzialmente dei dati, che definiscono lo stato del dispositivo in termini di tre categorie:
- **Dati di Configurazione**: ***dati che posso impostare per configurare il dispositivo*** (es. velocità del collegamento, dimensione del buffer etc..);
- **Dati Operativi**: ***dati scritti dal dispositivo nel mentre funziona*** (es. se si parla di router i dati operativi potrebbero essere le tabelle di instradamento che sto calcolando);
- **Statistiche**.

Sul **server di gestione** gira del ***software tramite il quale possiamo estrapolare ed elaborare i dati in questione***, oppure ***inviare ai dispositivi dei dati di configurazione***. Ciò viene fatto tipicamente con un certo grado di supervisione da parte di gestori umani (**human in the loop**).

Per comunicare con gli agenti il server di gestione utilizza **protocolli di gestione**. Ne esistono tre principalmente:
- **CLI** (***linea di comando***) == approccio più semplice per cui l'operatore scrive i comandi su una console del dispositivo o esegue lo script da remoto ad es. via ssh
- **Protocollo SNMP/MIB** == ***l'operatore interroga/imposta i dati*** (modellati come una sorta di **database MIB**) ***utilizzando il protocollo SNMP***
-**Protocollo NETCONF/YANG** == *poiché storicamente SNMP utilizzato per la lettura e quindi monitoraggio più che per la scrittura* e quindi configurazione, *si è sviluppata quest'alternativa*.
Simile concettualmente a SNMP ma con enfasi maggiore sul definire una configurazione di rete piuttosto che singoli dispositivi (**configurazione multidispositivo**). **YANG è il linguaggio tramite il quale definiamo i dati accessibili dal dispositivo gestito**.

SNMP

In genere utilizza come protocollo di trasporto **UDP**.
Dal punto di vista applicativo ***due modalità di comunicazione***:
- **classica richiesta risposta** (il server di gestione manda una richiesta all'agente)
- **modalità trap** (l'agente manda proattivamente messaggi trap al server, *tipicamente per informarlo di un imprevisto*)

Nella PDU di questi messaggi ***deve essere presente un ID che associ la richiesta alla risposta, poiché tipicamente usiamo UDP***). I dati acceduti nel dispositivo sono modellati nella **MIB** che può essere visto come un **database gerarchico** in cui, per trovare un dato specifico, devo seguire un certo percorso.

I dati sono definiti dal linguaggio **SMI**, che svolge quindi un ruolo simile a YANG per NETCONF

NETCONF

Per **NETCONF** il discorso è simile a SNMP, si tratta semplicemente di un'alternativa più moderna.
L'obiettivo, come per SNMP, **è gestire ma soprattutto configurare** (cosa per cui SNMP non era stata nativamente progettata) **in maniera attiva i dispositivi sulla rete**.

Come SNMP ha le trap, NETCONF ha le **notifiche** (sono la stessa cosa).
Come avviene l'interazione tra due estremi nel protocollo NETCONF?
Tramite una **chiamata a procedura remota rpc** fatta stabilendo una sessione con un protocollo di trasferimento dati affidabile e sicuro, inviando messaggi **xml** atti a codificare sia la richiesta che la risposta.

YANG è il linguaggio di modellazione dei dati utilizzato per specificare struttura, sintatti e semantica dei dati di gestione della rete NETCONF.

Livello di collegamento

Il livello di collegamento si trova immediatamente sotto il livello di rete nella pila protocollare di Internet. *Mentre il livello di rete si occupa di trasferire i pacchetti lungo un percorso ben definito da mittente a destinatario*, **il livello di collegamento si occupa nel pratico di trasferire il pacchetto di un nodo a quello a lui adiacente** (dove ***nodo == host/router/switch*** e adiacente significa collegati tramite lo stesso collegamento).

Se voglio inviare un pacchetto a un host in un'altra sottorete, il datagramma viene incapsulato in un frame con destinazione il router di primo hop (router di bordo). Quando il frame arriva lì il router lo decapsula e osserva che il datagramma è destinato altrove: svolgendo la funzione di inoltro reincapsula il pacchetto in un frame con destinazione il router di hop successivo e così via fino alla destinazione, dove il destinatario si accorge di esserlo e quindi decapsula tutto.

Poiché i collegamenti possono essere di diversa natura (wireless, ethernet cablata..), e il livello di collegamento si occupa del trasferimento dei pacchetti un collegamento alla volta, non sorprende che questo livello presenti moltissimi protocolli diversi.
Servizio principale del livello di collegamento == trasferire i dati da un nodo a quello fisicamente adiacente lungo un collegamento fisico.

Servizi livello di collegamento

- **Accesso al collegamento**: è necessario un *protocollo che si occupi di disciplinare l'accesso al mezzo trasmissivo*. Per capire a chi inviare i dati viene utilizzato inoltre un **indirizzo MAC** (limitato al livello 2).
-**Affidabilità tra nodi adiacenti**: nel caso di ***collegamenti decisamente poco affidabili*** (come Wireless) ***conviene risolvere l'affidabilità direttamente tra due nodi adiacenti senza forzare i meccanismi end-to-end di TCP o QUIC***. In questo senso entra quindi in gioco anche il servizio di **Controllo e Correzione degli errori.**
-**Framing**: il datagramma è incapsulato in un frame con intestazione e trailer
-**Controllo di flusso**: come per TCP si vuole gestire la frequenza di frame inviati a partire da un nodo in funzione della capacità del nodo adiacente di gestirli.
-**Half Duplex** se il collegamento non supporta la possibilità di inviare frame contemporaneamente da una parte e dall'altra, **Full Duplex** se invece è possibile.

Il livello di collegamento (e fisico) sono implementati in ogni singolo host **dall'adattatore di rete** o dalla **scheda di rete**, che si collegano al bus del sistema.
Il datagramma viene incapsulato in un frame a cui sono aggiunti i bit per la correzione degli errori implementando trasferimento affidabile e controllo di flusso.

Rilevazione e correzione degli errori

Per far sì che un trasferimento sia affidabile è anzitutto necessario rendersi conto della presenza di un errore. *Viene utilizzata un'informazione ridondante* (**EDC**) che sono **bit aggiuntivi necessari a comprendere se vi è stato o meno un errore** quando il pacchetto arriva a destinazione.

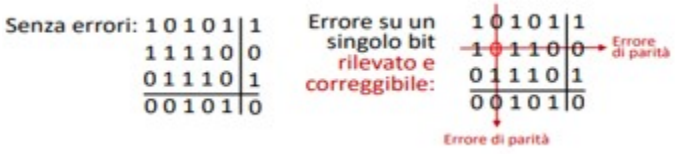
Se i dati sono corrotti, ***per correggerli abbiamo due possibilità***:
- **ARP**: il pacchetto viene scartato e me lo faccio **ritrasmettere**
- **Codici di Correzione degli errori FECC**: oltre che rilevare l'errore via EDC, lo **correggo** pure.

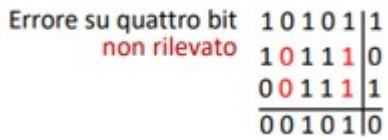
Controllo di parità

Si aggiunge un bit di parità alla sequenza di d bit di dati che sto inviando.
Il bit può essere di **parità pari** o **parità dispari**: *se parità pari perché aggiungendo il bit ottengo un numero pari di 1 nella sequenza, dispari se ottengo numero dispari di 1 nella sequenza*.
Il ricevente calcola quindi la parità dei d bit ricevuti e calcola anch'esso il bit di parità. **Se diversi allora errore**!

Vantaggio: poco spazio in più da usare
Svantaggio: **rileva solo numero dispari di errori**! (già se un bit cambia due volte torna alla parità che aveva prima)

Con il controllo di **parità bidimensionale** è necessario utilizzare più spazio ma offre garanzie decisamente migliori. I d bit vengono disposti lungo una matrice. Si fa il controllo di parità per ogni riga e per ogni colonna, poi il controllo di parità tra i bit risultanti. Con questo metodo **si possono rilevare tutte le combinazioni di al più 3 errori** e **si possono correggere gli errori singoli**!
Per avere parità pari si calcola il bit di parità con lo XOR.





21.

Ma quanto è efficace in fin dei conti l'EDC parità bidimensionale? **Ha senso utilizzarlo chiaramente solo se la quantità di errori che ci si aspetta è bassa e gli errori sono indipendenti**, ma *nella realtà gli errori tendono ad occorrere in burst e quindi a non essere indipendenti*.
In pratica quindi se un bit è errato c'è una probabilità maggiore che anche i successivi lo siano.

CRC Controllo di ridondanza ciclica

Si tratta di un EDC che permette di rilevare una percentuale maggiore di errori rispetto agli EDC visti finora, è **largamente utilizzato in Ethernet e Wireless**.

Immaginiamo di **voler inviare d bit**. Mittente e destinatario **si accordano su una sequenza di r+1 bit**, detta **polinomio generatore G**.
Il mittente aggiunge alla sequenza di d bit (D) r bit finali (**R**), ***calcolati*** secondo un'aritmetica particolare **affinché <D,R> sia divisibile per G**.
Per trovare R il mittente infatti anzitutto aggiunge a D un numero pari a r zeri, e dopodiché inizia una serie di divisioni per G finché non trova un resto.
Questo resto sarà proprio il valore R.
Quando il mittente riceve <D,R>, effettua nuovamente la divisione per G.
Se ottiene resto pari a 0 (poiché abbiamo costruito <D,R> affinché sia divisibile per G) **allora non sono stati rilevati errori**, altrimenti errore.

L'aritmetica utilizzata in CRC è un'aritmetica modulo due particolare, infatti *non vengono effettuati riporti o prestiti nelle addizioni/sottrazioni*.
Ciò comporta che **la divisione tra bit corrisponde sostanzialmente ad effettuare lo XOR bit a bit**.

Ma perché in UDP e altri protocolli non viene usata CRC, che ha molte più possibilità di rilevare gli errori? Poiché **per implementare CRC, a causa della sua complessità, è necessario utilizzare programmi software più lenti rispetto ad hardware** (come può avvenire invece per checksum!).

Quindi per protocolli il cui scopo ultimo è ad esempio favorire quanto più possibile il controllo sulla temporizzazione, senza eccessive esigenze in termini di affidabilità, ha senso utilizzare checksum o bit di parità piuttosto che CRC.

il generatore non viene scelto casualmente ma con proprietà tali da consentire che questo possa rilevare diversi tipi di errore.

CRC permette di rilevare tutti gli errori a burst di lunghezza inferiore a r+1 bit.
Inoltre se G ha numero pari di bit a 1, allora è in grado di rilevare qualsiasi numero dispari di errori (quindi tutti gli errori fino a r+1).

Protocolli di accesso multiplo MAC medium access control

Esistono due tipi principali di collegamento:
-**Punto-Punto** per cui ***ho solo due nodi specifici, un mittente e un destinatario, alle estremità del collegamento***
-**Broadcast** per cui più nodi possono comunicare contemporaneamente tra loro. In particolare ***se un nodo invia un frame questo viene ricevuto da tutti gli altri nodi connessi***.

Per quel che riguarda il collegamento broadcast, ***se più nodi trasmettono contemporaneamente si possono creare delle collisioni*** per cui i dati inviati originariamente risulteranno essere corrotti all'arrivo. È quindi necessario adottare dei protocolli specifici, detti **protocolli di accesso multiplo**, che regolino la trasmissione di dati nel canale da parte dei vari mittenti.
Ne esistono di molti tipi, vengono utilizzati in base alle esigenze del collegamento. Dividiamo questi protocolli in tre tipologie:

- **Suddivisione del canale**: basati sull'idea di ***dividere il canale in "pezzi" in base a tempo, frequenza o codice***. In particolare abbiamo già incontrato, parlando della commutazione di circuito, **TDM** e **FDM**! (time e frequency division multiplexing). Un altro protocollo di suddivisione del canale che vedremo è quello di divisione di codice.
Si ricorda che *con TDM ogni mittente trasmette a velocità massima R per un periodo di tempo limitato*, mentre *con FDM ogni mittente trasmette continuamente ma usando una porzione di banda pari ad R/M* con M numero di utenti che trasmettono.

Ha senso usarla solo se il canale è molto utilizzato! Infatti poiché assegno in maniera statica ad ogni trasmettitore una porzione di esso spreco se non trasmettono! Ha senso usarla solo se il canale è molto utilizzato

-**Protocolli ad accesso casuale**: ***quando un nodo ha dati da inviare, trasmette alla massima velocità senza alcun coordinamento***. Ciò implica che due nodi possono trasmettere contemporaneamente -> **collisioni** -> **questi protocolli sono progettati appositamente per rilevarle e recuperarle** attraverso ritrasmissioni

-**Protocolli a rotazione**: ***uniscono "il meglio" dei protocolli a suddivisione del canale e ad accesso casuale***.

Protocolli ad accesso casuale: Slotted ALOHA

Con questo protocollo si fanno **diverse assunzioni**: ***si divide il tempo in diversi slot temporali pari al tempo necessario di trasmettere un frame***, ***tutti i nodi trasmettono frame di uguale dimensione***, ***un frame può essere trasmesso solo all'inizio di uno slot temporare*** *e* ***deve esserci sincronizzazione dei nodi*** (nel senso che devono tutti sapere quando un nuovo slot temporale sta per iniziare)

Quando un nodo può trasmettere un frame, lo trasmette all'inizio dello slot utilizzando il canale al massimo delle sue capacità. **Se si verifica una collisione (la rilevo tramite ACK dal destinatario) allora ci si aspetta che questa venga rilevata entro la fine dello slot temporale corrente**.
Quando trasmette e non vi è collisione allora il nodo può eventualmente trasmettere un nuovo frame nello slot temporale successivo.
Quando trasmette e vi è collisione allora si randomizza: con probabilità p il nodo potrà trasmettere allo slot temporale successivo. Se invece con probabilità 1-p non può trasmettere allora ripete il ragionamento per lo slot successivo ancora.

Il vantaggio nel randomizzare sta nel fatto che se tutti ritrasmettessero allo slot successivo vi sarebbe alta probabilità di collisione e quindi spreco di slot.

Vantaggi: algoritmo **decentralizzato e semplice**, inoltre **se è solo uno il nodo a dover trasmettere può utilizzare il canale a piena potenza**
Svantaggi: per la questione probabilistica **diversi slot risultano sprecati** (nessuno trasmette), inoltre **è necessario un minimo di sincronizzazione** per cui tutti i nodi sanno quando inizia un nuovo slot.

Con Slotted ALOHA 37% degli slot utilizzati, quindi in media un nodo sfrutta lo 0.37 R della capacità trasmissiva del canale.
Esiste la **variante ALOHA puro** che funziona allo stesso modo ma senza slot, la percentuale scende allo **18%.**

Ciò che rende così difficile l'approccio per i protocolli di accesso multiplo è il fatto che nei wireless non è banale osservare che un canale è occupato.
Con il filo la questione è più semplice, per questo Ethernet cablato vecchia scuola (condiviso) utilizzava un protocollo di accesso casuale detto **CSMA**.

Protocolli ad accesso casuale: CSMA

Il protocollo CSMA si basa sull'idea di effettuare il **rilevamento della portante**: *quando un nodo vuole trasmettere verifica anzitutto se il canale risulta essere libero o occupato*. Se è occupato non trasmette, altrimenti invia il frame.

Esiste una variante di CSMA che *si occupa anche di rilevare eventuali collisioni*: **CSMA/CD (Collision Detection)**. È vero infatti che un nodo trasmette non appena rileva il canale libero, *tuttavia il ritardo di propagazione nella trasmissione comporta che talvolta comunque più nodi possano contemporaneamente rilevare il canale libero e tramettere,* ***comportanto la verifica di collisioni***.

Vediamo come si comporta CSMA/CD:
1) il nodo effettua il rilevamento della portante. Se il canale è libero inizia la trasmissione, altrimenti attente
2) Se il frame viene inviato nella sua interezza senza problemi allora passa eventualmente a trasmettere il frame successivo
Se il viene rilevata una collisione allora il nodo ***invia un segnale di JAM (interferenza)*** per segnalare a tutti gli altri nodi che vi è appena stata una collisione
3) Il nodo entra in una fase di **binary exponential backoff**: sia m il numero di collisioni avvenute fino a quel momento. Viene scelto un numero casuale tra 0 e 2^m-1 (K): il tempo che il nodo dovrà aspettare sarà il tempo necessario per inviare 512 bit moltiplicato per K.

Il valore di m è limitato a 10***. Questo approccio è vantaggioso perché se si verificano più collisioni più nodi dovranno aspettare più tempo di modo che si rallentino le eventualità di collisione***.

L'equazione che descrive l'efficienza di questo protocollo è:

$$efficienza = \frac{1}{1 + 5d_{prop}/d_{trasm}}$$

dove dprop è ritardo di propagazione e dtrasm tempo in cui un nodo trasmette dati.

L'efficienza tende a 1 (ossia si usa il canale al massimo) **se il ritardo di propagazione tende a 0** (e quindi 0 collisioni) **e se il tempo di trasmissione tende a infinito** (cioè un nodo tiene il canale occupato di continuo.

Questo protocollo è quindi chiaramente molto più efficiente di Slotted ALOHA. Ma perché non utilizzare sempre questo? ***CSMA/CD pretende che le collisioni vengano rilevate immediatamente e, a differenza di ALOHA, con metodi più rapidi del semplice ACK di risposta dal mittente***.
Mezzi di comunicazione cablati come Ethernet sono in grado di rilevare rapidamente e con relativa facilità le collisioni, quindi usano largamente questo protocollo. Altri mezzi trasmissivi, come Wireless, non se lo possono permettere.

Protocolli a rotazione: Polling

Vi è un **controllore centralizzato** che periodicamente invia a ciascuno dei nodi connessi un **token** *che gli permette di trasmettere per un certo tempo*.
Il controllore può agire a rotazione oppure, tramite un'implementazione a rilevamento della portante, può decidere di far trasmettere un altro nodo dal momento che un nodo a cui ha affidato un token non sta trasmettendo.

Diversi svantaggi: essendo centralizzato il controllore rappresenta un **singolo punto di rottura**, i vari **messaggi di controllo sprecano banda del mezzo trasmissivo** e poiché i client devono attendere il token da parte del controllore per trasmettere **vi è un ritardo prima della trasmissione** vera e propria che comporta un **thoughput effettivo minore rispetto ad R**.

Bluetooth utilizza Polling.

Token Passing

Simile al Polling ma meno centralizzato, ***i token sono scambiati ciclicamente tra i client***. Ancora svantaggi legati alla latenza e al singolo punto di rottura, stavolta legato al concetto di token.

NB. Se ho un canale broadcast a singolo trasmittente chiaramente non ho bisogno di un protocollo ad accesso multiplo
NB nelle reti di accesso via cavo in **downstream si utilizza FDM** per suddividere la banda del cavo coassiale tra canali TV e modem.
In upstream invece entra in gioco il problema broadcast e quindi **protocollo di accesso multiplo**. **Tutti gli utenti**, similmente a Slotted ALOHA, **si contendono secondo un algoritmo di accesso casuale determinati slot temporali del canale upstream.**

22.

Ha quindi senso usare protocolli a suddivisione del canale se il canale è molto utilizzato, protocolli ad accesso casuale se non è troppo utilizzato (altrimenti eccesso di collisione).

Una **LAN** **Local Area Network** identifica una rete dominante su un diametro ristretto (abitazione, edificio, scuola…).
Le tecnologie utilizzate per connettere i dispositivi nella LAN sono:
-**Ethernet** (cablato, usato anche in altri campi)
-**WiFi**

Una LAN rappresenta, agli occhi del livello di rete, una subnet. Agli occhi del livello di collegamento invece rappresenta una rete vera e propria.
Ma come identifica i dispositivi il livello di collegamento, affinché possa inviare i frame tra nodi adiacenti correttamente?
Tramite **l'indirizzo MAC**, limitato al livello di collegamento.

Nonostante abbiamo a disposizioni gli indirizzi IP del livello di rete infatti, si è preferito un ***approccio di indipendenza tra livelli***, in modo tale che se il frame dovesse incapsulare un datagramma appartenente ad un protocollo particolare senza identificativi IP possa comunque svolgere il proprio ruolo senza problemi. Inoltre gli indirizzi MAC hanno ***vantaggi in termini di portabilità***.

Differenze tra indirizzo IP e MAC:
- l'indirizzo IP e MAC sono assegnati ***entrambi alle interfacce di rete***, ma quest'ultime hanno due indirizzi: a livello di rete (IP) e ***a livello di collegamento (MAC)***!
- l'indirizzo IP va da 32 bit di IPv4 a 128 bit di IPv6, ***MAC conta 48 bit***
- ***Gli indirizzi IP possono anche essere dinamici***, gli **indirizzi MAC invece sono associati Read-Only ad ogni dispositivo dal produttore, che ottiene a sua volta un blocco di indirizzi dall'IEEE**.

(In realtà è possibile "cambiare" indirizzo MAC via software per privacy)

MAC, per la sua natura di "codice fiscale", offre vantaggi in termini di **portabilità**. Se porto il dispositivo da una LAN a un'altra non ho problemi a livello di collegamento (mentre a livello di rete si adottano le tecniche viste in precedenza per identificare un nuovo IP e diffondere l'informazione che mi trovo in quella specifica sottorete):

Ma il passaggio del frame attraverso i collegamenti deve avvenire in funzione dell'instradamento deciso dal livello di rete, per tale ragione **per scegliere** il nodo adiacente corretto (e quindi **indirizzo MAC di destinazione** a livello di collegamento) **devo "tradurre" l'indirizzo IP a livello di rete in quello MAC corretto**.
Per fare ciò si utilizza un ***protocollo per la risoluzione degli indirizzi***: **ARP**.

ARP

L'idea alla base è ***la presenza di una tabella ARP all'interno di ogni dispositivo di livello 3 della sottorete***. Questa tabella contiene **l'associazione tra indirizzi IP e MACaddress**, e un campo **TTL** per cui un'associazione, tipicamente, non resta valida per più di 20 secondi.

Per popolare la tabella avviene ciò che segue:
Il nodo A, che vuole mandare un pacchetto ad un certo IP di destinazione, necessita di conoscere il MACaddress corrispondente. Per farlo invia una ***richiesta ARP***: si tratta di *un messaggio inviato in broadcast al livello due* (FF.FF.FF.FF.FF.FF) quindi a tutti i dispositivi della sottorete.
Questo messaggio contiene ***IP sorgente, destinazione e MAC di A***.

Quando un nodo riceve la richiesta ARP decapsula il frame e ***se vede che l'IP di destinazione coincide con lui allora invia una risposta ARP associando il proprio MACaddress*** (la può inviare poiché il mittente ha incluso il proprio MAC nella richiesta), *altrimenti ignora il messaggio*.

E se la volessi inviare un pacchetto fuori dalla mia sottorete?
Anzitutto il mittente si accorge di un evento del genere per via del fatto che il prefisso del proprio indirizzo IP dettato dalla codifica CIDR è diverso rispetto a quello dell'IP destinatario.

In tal caso allora ***il mittente imposta come IP destinatario quello del router di bordo*** (gateway router/router di primo hop) *che conosce grazie al protocollo* **DHCP** (protocollo di gestione della rete che assegnando l'IP al dispositivo gli ha fornito altre informazioni, tra cui questa). Ma il mittente non conosce il MACaddress del destinatario, *per ricavarlo usa quindi **ARP** come appena visto*.

Quando il pacchetto arriva al router di bordo, questo lo spacchetta e arriva all'IP di destinazione del datagramma, per poi compiere la classica procedura di inoltro. Quando si arriverà alla sottorete prevista il router di bordo di quella sottorete per inviare il pacchetto al destinatario utilizza ancora una volta ARP affinché possa ricavarne l'indirizzo MAC.

Ethernet

Tra i principali collegamenti utilizzati nelle LAN vi è Ethernet. Si tratta di una tecnologia risalente agli anni ’70 che ancora oggi è ampliamente utilizzata per via del fatto che è *rimasta al passo con la velocità*.

Inizialmente implementata connettendo i vari dispositivi con **approccio a bus**, *con cavo coassiale. La rottura di un singolo collegamento avrebbe comportato totale perdita di funzionalità della LAN.*
Approccio successivo è stato quello con **topologia a stella con hub**. ***Un hub è un dispositivo che lavora a livello fisico occupandosi di rigenerare i segnali.***
Il problema di questi approcci erano le collisioni: due dispositivi non potevano comunicare contemporaneamente con altri.

L’approccio migliore, ancora in utilizzo, è a **topologia commutata** (**switched**). *La comunicazione tra i nodi è regolata da uno **switch**, dispositivo che lavora fino al livello di collegamento*. Quando un frame arriva allo switch, prima di inoltrarlo verifica che il collegamento è libero altrimenti lo bufferizza.
Lo switch quindi lavora nella modalità store and forward già vista per i router.

PDU Frame Ethernet

l'interfaccia trasmittente incapsula il datagramma IP (o altro pacchetto di protocolli di livello di rete) in frame Ethernet

Preambolo: ***7 byte 10101010 necessari per “risvegliare” la scheda di rete del ricevente e sincronizzare il suo clock con quello del trasmittente, poi un byte 10101011 per indicare l’inizio effettivo dell’intestazione***
Indirizzo di destinazione e sorgente: 48 byte per il MACaddress
Tipo: indica che *protocollo si sta utilizzando a livello superiore*. Necessario quindi per il fenomeno di **demultiplexing**
Payload
Controllo di errore CRC: 4 byte dedicati. Se viene rilevato un errore, allora il frame è scartato.

Ethernet è un protocollo **senza connessione**, **inaffidabile** e che **utilizza come protocollo di accesso multiplo** quello di accesso casuale **CSMA/CD** con exponential binary backoff.
Ethernet non rappresenta un solo standard.
Ne esistono molti e diversi, ciò che li **accomuna** sono il **protocollo MAC** e il **formato dei Frame**, mentre ciò che li **differenzia** sono le **velocità**

Esiste un limite nella lunghezza dei fili, legata 1) alla capacità del mezzo di trasmettere segnale (man mano che aumenta la distanza il segnale si attenua) e 2) collisione.
Il limite di un cavo in rame di ethernet è tipicamente di 100 metri.

Switch

Si tratta di commutatori di pacchetto di livello 2.
Svolgono un **ruolo attivo** nella LAN *in quanto memorizzano e inoltrano i pacchetti secondo il meccanismo di store and forward*, filtrando la loro destinazione in base all'indirizzo MAC di destinazione che presentano.

Gli switch **sono dispositivi trasparenti**: host e router sono inconsapevoli della loro presenza, infatti gli switch non hanno MAC associato e se lo hanno non svolge un ruolo nella commutazione dei pacchetti.
Si tratta di dispositivi che **supportano collegamenti eterogenei** poiché non inficiano sulla giunzione elettrica che li collega ad altri nodi.
Inoltre sono **dispositivi plug and play** e che svolgono la funzione di commutazione tramite un meccanismo di **autoapprendimento**.

Se due nodi vogliono comunicare con lo stesso nodo destinatario lo switch li mette in coda affinché si evitino collisioni.
Grazie all'accodamento dei pacchetti gli switch eliminano aprioristicamente la possibilità di collisioni. *Le uniche collisioni che vi possono essere sono tra mittente e switch e tra switch e mittente*!
In questo senso esistono due possibilità:
- se il collegamento consente **modalità full-duplex** *allora mittente e switch possono comunicare tra loro senza problemi* (non serve nemmeno protocollo di accesso multiplo)
- se il collegamento in **modalità half duplex** *allora è necessario utilizzare il protocollo* **CSMA/CD** *per regolare eventuali collisioni*.

Come avviene nel pratico la commutazione?
Come anticipato i router si basano su un concetto di **autoapprendimento**.
Ogni switch presenta una **tabella di commutazione**, dove ***è presente l'associazione (se esiste) tra indirizzo MAC, linea di uscita e un attributo TTL*** (in quanto le destinazioni in LAN possono cambiare).

Quando un nodo con un certo MACaddress invia un pacchetto e questo passa per lo switch, anzitutto lo switch (se non lo ha già salvato) *aggiunge alla tabella di commutazione l'informazione relativa al fatto che a quella specifica linea corrisponde il MACaddress di quel nodo, impostando anche il TTL*.

Dopodiché, per inoltrare il pacchetto al destinatario corretto (se non esiste corrispondenza nella tabella) applica un processo di **flood**, ***ossia invia il pacchetto lungo tutte le linee di uscita che non ha salvato all'interno della tabella***. *Se invece esisteva associazione tra MAC destinazione e linea di uscita*, **inoltra il pacchetto selettivamente** (*ossia lo inoltra solo dal momento che linea di uscita e linea di ingresso di quel pacchetto non coincidono*, in tal caso infatti significa che mittente e destinatario coincidono e quindi deve aggiornare la tabella, scarta il frame).

- *router:* calcolano le tabelle usando algoritmi di instradamento, indirizzi IP
- *switch:* autoapprendimento della tabella di inoltro usando il flooding, indirizzi MAC

Mentre i router tramite gli algoritmi di instradamento possono trovare percorsi ottimali aciclici, ***è necessario che gli switch siano interconnessi ad albero per evitare che il traffico broadcast circoli per sempre***. (**Spanning Tree Protocol**)

Una domanda che potrebbe sorgere spontanea è: perché a questo punto utilizzare i router? Non potrei creare un'unica grande rete con solo switch?

Il problema è che Ethernet e in generale le reti di livello 2 **non sono pensate per essere scalabili**, la prima cosa che ci suggerisce ciò è che le reti di livello 2 si basano molto sul concetto di diffusione lungo un singolo dominio di broadcast (la rete di livello 2 in questione), che deve essere limitata nel numero di nodi!

Ciò inoltre può comportare **problemi di sicurezza e privacy** (es. inviare messaggi per cui le destinazioni nelle tabelle di commutazione non corrispondono, malintenzionati che possono sniffare i pacchetti).

VLAN

Immaginiamo che un utente di Computer Science si sposti nell'ufficio EE (connesso **fisicamente** allo switch EE) ma voglia rimanere connesso **logicamente** allo switch CS. Come fare?
Esiste un'alternativa usando le così dette VLAN (Virtual Local Area Network).
L'idea è che ***posso usare le VLAN per poter definire più LAN virtuali su un'unica infrastruttura LAN fisica***.

Quando si parla di **port-based VLAN** la storia è molto semplice e intuitiva: se uno switch ha ad esempio 16 porte, posso assegnarle in modo dinamico tra le varie VLAN: dal punto di vista logico sarà come ad esempio se avessi due switch; le prime 8 porte sono per uno switch e le altre 8 per un altro switch.

Come fare poi a farli parlare tra loro? Tramite un **routing** implementato nello switch originale! In pratica i produttori combinano gli switch con i router. In questo modo l'inoltro tra reti di livello 2 non sarà più supportato solo dal livello 2 ma sarà supportato con il livello di rete.

Una **porta trunk** trasporta frame tra VLAN definite su più switch fisici. Per capire di che VLAN sto parlano quando mando il frame questo deve contenere più informazioni legate proprio all'ID VLAN.
Il frame ethernet è "allungato" aggiungendo 2 byte di tag **Protocol Identifier**, un tag con **Informazioni di Controllo**.

23.

Wireless

Si tratta di un approccio nel collegamento per cui la propagazione del segnale avviene attraverso onde elettromagnetiche, non cablato.
Dobbiamo distinguere due concetti quando si parla di wireless:
- **Wireless** == ***il collegamento in sé***, anche nel senso più strettamente fisico
- **Mobilità** == si tratta della ***possibilità per cui un dispositivo nella rete si sposta anche tra sottoreti diverse***, senza riscontrare problemi.

Esistono diverse tecnologie Wireless, *ciò che principalmente le differenzia sono la velocità di trasmissione e il raggio di copertura*.
Le principali componenti di una rete Wireless sono:
- **Host**: gli host possono direttamente comunicare tra loro via Wireless oppure tramite una Stazione Base (dipende dall'approccio). *Non tutti gli host sono uguali in termini di mobilità*.

- **Collegamento Wireless**: si tratta di un collegamento senza filo che sfrutta la trasmissione dei segnali come onde elettromagnetiche. Si tratta di un **collegamento chiaramente in broadcast**, e per questo deve essere regolato dall'utilizzo di specifici ***protocolli di accesso multiplo*** (**MAC** medium access control). *Si ricorda che in wireless è difficile rilevare immediatamente le collisioni*.
- **Stazione Base**: si occupa di propagare il segnale in forma wireless verso tutto Internet (*tipicamente in forma cablata*) oppure da tutto Internet verso gli host wireless connessi. Una stazione base può essere una vera e propria torre radio (4G o 5G) oppure un semplice dispositivo WLAN domestico.

Bluetooth inteso come la tecnologia di rimpiazzo di cavi più semplice possibile. Copre un breve raggio e ha bassa capacità trasmissiva.
Si ha poi la **tecnologia WiFi**: tutti sotto lo stesso standard 802.11 ma differiscono in termini di coperture e velocità.
Infine si ha la **tecnologia 4G e 5G**: copertura maggiore in assoluto (chilometri) e con 5G velocità sempre più alta.

Esistono **due principali modalità** per costruire una rete wireless:
- **Modalità infrastruttura**: *gli host sono collegati ad un'infrastruttura dove è presente la **stazione base***, che **si occupa di fornire tutti i servizi tipici di rete** come instradamento, indirizzamento, traduzione DNS etc…
In questa modalità non si deve dimenticare il problema dell'**handoff**, *per cui si deve essere in grado di gestire la possibilità che dispositivi progettati per la mobilità si spostino tra infrastrutture diverse*
- **Modalità ad hoc**: *gli host comunicano tra di loro direttamente entro il raggio di copertura wireless*, senza l'ausilio di una stazione base.
Si presuppone in questo caso che i servizi di rete vengano forniti direttamente dagli host.

Wireless per la sua natura non cablata **comporta alcuni problemi**:

- **Attenuazione del segnale**: presente anche nel cablato ma *in wireless molto più evidente.* Infatti in Wireless può dipendere dall'***attraversamento di ostacoli*** e dall'***attenuazione nello spazio libero***. La formula che definisce quest'ultima è **(fd)^2, quindi maggiori frequenza e distanza maggiore sarà l'attenuazione**.

-**Propagazione lungo cammini multipli**: *il segnale può arrivare più velocemente all'host tramite un percorso diretto ma anche, dopo varie riflessioni nello spazio circostante, arrivare successivamente*. Si devono quindi distinguere i segnali di arrivo: si utilizza un ***Tempo di Coerenza*** (intervalli in cui l'host si aspetta di ricevere un segnale specifico). Ciò comporta diminuimento di banda (solo in un certo tempo l'host si aspetta di ricevere un certo segnale)

- **Interferenza**: può essere causata da ***altri dispositivi che trasmettono sulla stessa sequenza*** oppure da ***rumore elettromagnetico ambientale***.
Per evitare queste interferenze la soluzione tipicamente è cambiare banda.

Legato a questo discorso vi è l'**SNR Rapporto Segnale Rumore**, si tratta del ***rapporto tra il segnale che ho ricevuto e il rumore di fondo***.
Maggiore è l'SNR migliore è il segnale, infatti significa che non è stato compromesso eccessivamente dal rumore e posso quindi comprenderlo.
Inoltre **maggiore SNR implica minore BER Tasso di errore sui bit**.
Dato uno schema di modulazione **maggiore è la potenza a cui mando il segnale, maggiore è SRN e minore è BER**. *Ma trasmissioni a maggiore potenza implicano maggiore consumo di energia (critico per dispositivi a batteria) e interferiscono con altre trasmissioni*.

Inoltre una tecnica di modulazione con più elevato tasso di trasmissione dei bit comporta BER più alto.

Terminale Nascosto: *Dati tre terminali ABC dove A e B possono parlare, B e C possono parlare e A e C non possono parlare, A e C possono causare (senza saperlo) interferenza presso la destinazione B* (A e C comunicano in contemporanea!)
Rappresenta un problema perché a differenza di Ethernet **con Wireless non posso rilevare con facilità le collisioni non appena avvengono**, quindi *non si possono utilizzare protocolli come CSMA/CD*.
Vi sono altre soluzioni, per WiFi **CSMA/CA.**

CDMA

Stiamo parlando del terzo protocollo appartenente alla catergoria di protocollo ad ***accesso multiplo per suddivisione del canale*** (insieme ad FDM e TDM).
CDMA rappresenta infatti **la suddivisione del canale in termini di codice**.
Facciamo alcune assunzioni per spiegare il procedimento: ***si assume che tutti i nodi sono sincronizzati nella trasmissione***, che *trasmettono segnali di pari intensità* e ***che i bit 0 rappresentano il valore -1 e i bit 1 rappresentano il valore 1***.

Per prima cosa **con CDMA viene assegnato ad ogni nodo nella rete un codice di dimensione m, tale che ogni codice sia ortogonale rispetto agli altri** (ossia il loro **prodotto scalare è uguale a 0**).
Quando devo inviare un bit, allora moltiplico il suo valore per il codice.
Se quindi voglio inviare il bit 0 ottengo il codice ma con segno opposto (ho moltiplicato per valore -1), se voglio inviare il bit 1 ottengo il codice normale.
Se quindi si vogliono inviare d bit, in realtà saranno inviati dxm bit.

Quando il ricevente ottiene il messaggio fa il prodotto scalare per il codice che possiede e divide per M (dove M lunghezza del codice), ottenendo così i bit originali.
Nel caso in cui avvenga un'interferenza, poiché assumiamo che le trasmissioni sono sincronizzate, ***si ottiene un segnale i cui bit rappresentano la somma dei due originali***. *Quando il ricevente ottiene il segnale, facendo il prodotto scalare per il codice e dividendo per M, ottiene comunque il segnale originale nonostante l'interferenza*!

WiFi e WLAN

Standard più recenti come n, ac e ax si sono spostati verso la frequenza di 5 GHZ. Inoltre standard che vogliono coprire più raggio si concentrano su frequenze più basse (coerentemente alla formula (fd)^2 per l'attenuazione nello spazio libero).

Lo standard 802.11 di wi-fi può usare sia modalità ad hoc che modalità ad infrastruttura.
Nella modalità ad infrastruttura in **particolare identifichiamo come stazione base un access point AP** e **l'insieme di AP e host che lavorano sotto lo stesso raggio di copertura come BSS Basic Service Set**.

Un WiFi che lavora sotto una certa frequenza divide quest'ultima in canali più piccoli, che si sovrappongono parzialmente.
Ad es. con 2.4 GHz ho un numero di canali pari ad 11, affinché possa essere sicuro di trasmettere senza interferenze devo utilizzare solo canali che non si sovrappongono che nel caso di questa frequenza sono 1, 6 e 11.

Quando si utilizza un AP quindi questo è configurato per ricevere e trasmettere solo frequenze di uno specifico campo, affinché non si creino interferenze con altri AP nelle vicinanze.

Un concetto importante correlato al WiFi è quello di **associazione** (tra host e Access Point AP). Esistono due approcci:
- **Approccio Passivo**: è l'AP ad inviare nel proprio raggio di copertura un ***frame beacon*** **contenente il proprio nome e indirizzo MAC**. In questo modo un host può osservare i vari AP accessibili nella propria posizione, potendo anche sceglierlo in funzione della potenza di segnale in quel punto.
Quando un host vuole connettersi ad uno specifico AP vi è tendenzialmente una *fase di autenticazione* prima che possa avvenire l'associazione.
Una volta associato, il prossimo passo per l'host è inviare un messaggio DHCP, attraverso l'AP, affinché ottenga un IP nella sottorete.
- **Approccio Attivo**: l'host invia dei frame sonda nella rete. Quando un AP lo riceve risponde con un frame sonda di risposta. A quel punto l'host invia un frame di richiesta di associazione all'AP e quest'ultimo potrà accettare o no.

CSMA/CA Collision Avoidance

Come anticipato ***uno dei problemi principali del Wireless è l'impossibilità di rilevare facilmente le collisioni non appena avvengono***, come invece è possibile fare via Ethernet. Due motivi principali:
- **Terminale Nascosto** (possibilità di interferenza in un punto tra due host senza che loro lo sappiano)
- **Questioni tecnologico/economiche**: *è difficile e costosto progettare un dispositivo wireless che possa rilevare onde di bassa intensità (collisioni) mentre sta trasmettendo altri segnali di maggiore intensità*.

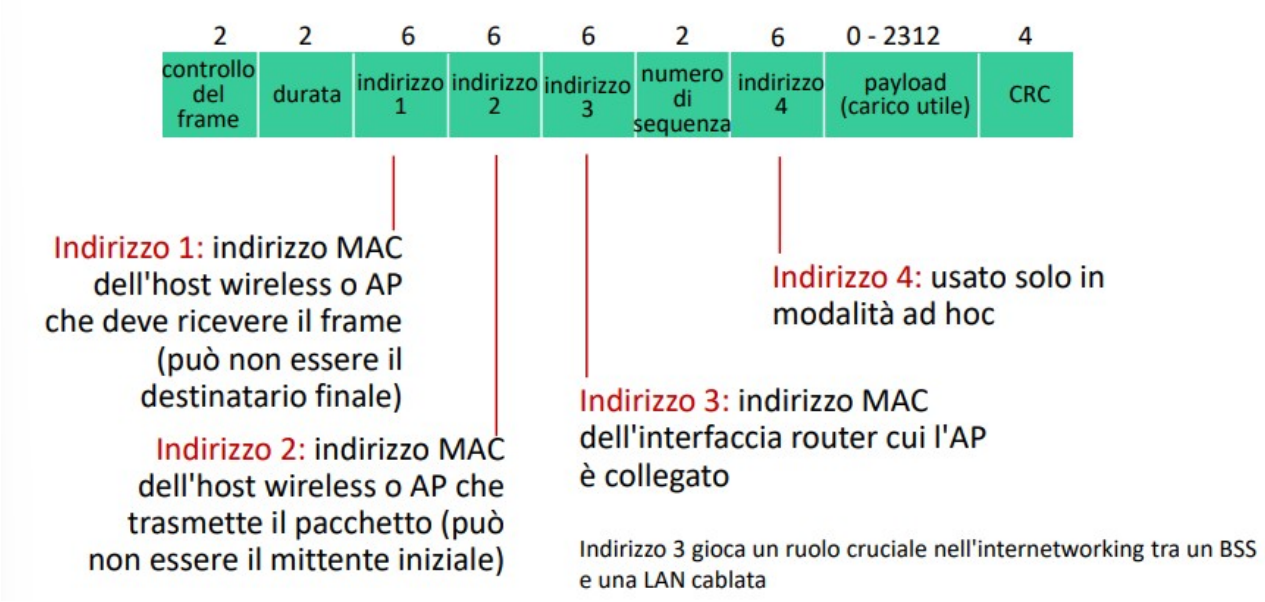
Per Wireless quindi non si può usare CSMA/CD che prevedeva di rilevare immediatamente la collisione, si usa **CSMA/CA** che ***effettua comunque il rilevamento della portante*** ma per cui **la collisione è rilevata in più tempo, indirettamente, tramite ACK dal destinatario**.

Come funziona CSMA/CA? Effettuo anzitutto rilevamento della portante.
Se il canale è libero per un tempo **DIFS** allora trasmetto, se è occupato:
1) ***vado in binary exponential backoff***
2) **il countdown** di attesa, dopo aver aspettato DIFS, **scende solo quando il canale è libero**
3) Quando il countdown arriva a 0 trasmetto il frame per intero. Il destinatario fa CRC e dopo un tempo di **SIFS** manda responso. *Se ACK allora ok, se NAK collisione e quindi aumento binary exponential backoff* e torno al punto 2).

Un altro metodo per evitare le collisioni è **RTS** (**Request to send**, ***prenotazione esplicita***).
Un nodo ***può inviare all'AP un frame RTS di richiesta di trasmissione usando CSMA/CA***. Quando l'AP riceve il frame ***risponde dopo tempo SIFS con un frame*** **CTS** (**clear to send**) che viene chiaramente ricevuto da tutti i nodi nel collegamento.
In questo modo ai nodi che non avevano inviato la richiesta RTS, CTS starà a significare che devono differire la trasmissione affinché non vi siano collisioni.
Al nodo che invece aveva inviato RTS, sarà permesso di trasmettere.

RTS è opzionale e viene utilizzata solo quando un nodo deve inviare un frame di grande dimensioni. Le collisioni di RTS non comportano gravi danni poiché sono di piccola dimensione (*il problema di non rilevare subito le collisioni sta nel fatto che il nodo continua a trasmettere il frame grande anche quando una collisione già c'è stata, quindi spreco di risorse*).

PDU 802.11



Informazioni di controllo

Durata: ***necessaria per l'opzione RTS e CTS***, serve ad indicare il tempo che il mittente ha per trasmettere

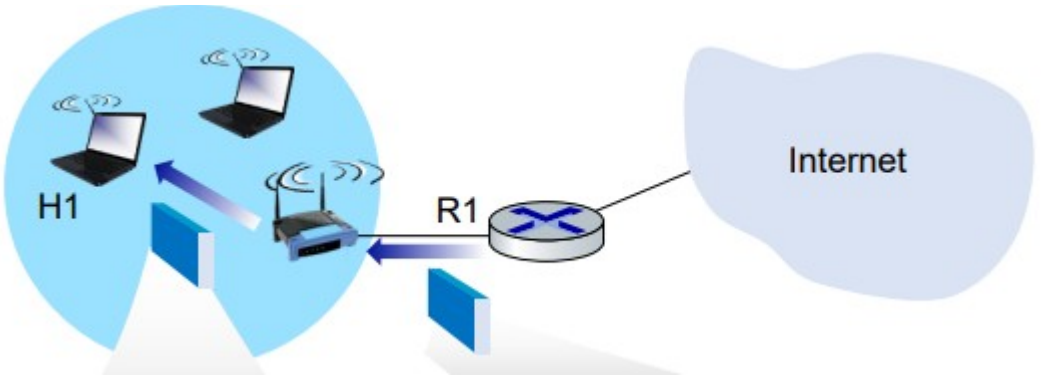
CRC

Numero di Sequenza: necessario per ***implementare il trasferimento affidabile a livello di collegamento*** (Wireless di sua natura poco affidabile, così non spreco risorse end to end

Payload

4 Indirizzi: Si hanno 4 indirizzi per via del fatto che con WiFi si distinguono mittente originale da quello attuale a livello di collegamento e lo stesso per il destinatario. In particolare:

- **Indirizzo 1** == ***MAC dell'host wireless o AP che deve ricevere il frame*** (non è detto che sia il destinatario finale)
- **Indirizzo 2** == ***MAC del dispositivo che ha appena inoltrato il pacchetto*** (può anche non essere il mittente iniziale)
- **Indirizzo 3** == ***MAC del router di bordo della sottorete collegato all'AP*** (necessario per accedere ad altre sottoreti)
- **Indirizzo 4** == ***usato solo nella modalità ad hoc***



Immaginiamo che il router R1 debba inviare il datagramma ad H1.
I dati saranno incapsulati in un frame il cui MAC address destinatario sarà H1 e come MAC address mittente l'interfaccia del router di questa sottorete.

Il frame arriva all'AP della sottorete con H1, e dovrà mandarlo ad H1. Dovrà creare quindi un frame di tipo 802.11 populando opportunamente i 4 indirizzi. Sarà messo come indirizzo 1 (che coincide con il destinatario in questo caso) H1 e come indirizzo 2 il proprio indirizzo MAC e come indirizzo 3 l'indirizzo del router R1.

Quando H1 riceve il datagramma tira fuori il payload e dovrà inviare un datagramma di risposta. Dovrebbe destinare il datagramma all'interfaccia del router, ma in realtà gli indirizzi di 802.11 saranno popolati come segue: l'AP sarà messo come indirizzo 1 poiché è lui che riceve il frame, metteremo nell'indirizzo 2 come colui che trasmette che è anche mittente H1, come indirizzo 3 ci si mette il router.

Mobilità all'interno della stessa sottorete

L'IP non cambia ma problema per lo switch, che dovrebbe aggiornarsi continuamente (gli switch con l'autoapprendimento non sono pensati per cambiamenti rapidi di posizione). ***L'idea è che ogni volta che l'host si sposta il nuovo AP invia un frame Ethernet broadcast con mittente il nodo che si è spostato affinché lo switch apprenda la nuova posizione per raggiungerlo***.

24.

Bluetooth

Si tratta del più semplice approccio Wireless per evitare l'utilizzo dei fili.
Raggio di copertura limitato e tasso trasmissivo di al più 3 Mbps.
Bluetooth utilizzato per le **WPAN** **Wireless Personal Area Networks** (cuffie, tastiere, mouse…) da non confondersi con le **Body Area Network** (accessori anche per la salute che utilizzano protocolli più veloci e sicuri).

Bluetooth lavora sulla frequenza ISM da ***2.4-2.5 GHz***. ***Si tratta di una banda molto affollata***, utilizzata da vecchi implementazioni di WiFi, telefoni cordless e che **può anche avere interferenze** a causa di rumori ambientali quali forni a microonde e alcuni motori.
Per questa ragione grande attenzione nel protocollo di accesso multiplo per Bluetooth per evitare interferenze.

Il protocollo di accesso multiplo per Bluetooth unisce i due protocolli di suddivisione del canale TDM e FDM, con slot temporali di tendenzialmente 625 microsecondi e banda divisa in 79 diversi canali.
Inoltre Bluetooth utilizza anche il protocollo a rotazione di Polling per regolare la trasmissione da parte dei nodi connessi.

Infatti Bluetooth è un'architettura wireless che **opera in modalità ad hoc**, per cui i nodi comunicano tra loro senza l'ausilio di una stazione base.
Esiste in particolare tra i nodi un **nodo master**, che via Polling determina quali nodi possono trasmettere e gestisce la comunicazione sincronizzandoli.
Per evitare le interferenze infatti Bluetooth utilizza il metodo **FHSS** **Frequency Hopping Spread Spectrum**, ***per cui un nodo ad ogni slot temporale trasmette in un canale di frequenza diverso***. In questo modo se in un canale c'è interferenza non si spreca tempo a trasmettere ancora in quel canale ma si passa ad un altro.

La rete Bluetooth è detta **WPAN** o **Piconet** poiché presenta pochi host contemporaneamente connessi (al più 8) e copre corto raggio.
Si possono avere anche al più **255 dispositivi in parked mode**, addormentati, che non dialogano nella rete ma vengono risvegliati dal master in caso di necessità.

Infine bluetooth è una rete ad hoc **bootstrapping**, *per cui cioè i nodi si "autoassemblano" nella piconet seguendo due fasi*:
- **Neighbor Discovery**: il master invia un frame ***inquiry*** ai vicini
- **Paging**: se un vicino si vuole connettere invia un ACK al master. Il master allora *invia un frame al nodo in questione contenente informazioni legate all'impostazione del clock (sincronizzazione TDM), il pattern dei canali che dovrà seguire (FDM) e un indirizzo su cui sostenere la comunicazione*.

Beacon == **dispositivi che inviano frequentemente un messaggio, per esempio con un identificatore**.
Possono servire a vari scopi, es. stupido beacon trasmette appena torno a casa e fa accendere ad Alexa le luci.

Reti Cellulari 4G e 5G

Ultima tecnologia che copre ampio raggio, pensata per dispositivi mobili.
Vediamo le somiglianze e differenze con l'Internet cablato:
- **Somiglianze**: Si hanno dei provider che coprono la rete locale e una fetta del nucleo della rete (come per gli ISP). *Devono essere interconnessi tra loro affinché vi sia copertura globale* (rete di reti, come ISP). La stazione di testa, tipicamente vere e proprie stazioni radio, sono collegate all'Internet cablato. ***Si utilizzano protocolli utilizzati anche nell'Internet cablato*** come DNS, UDP, TCP, IP, NAT. ***Vi è separazione tra piano di controllo e dati, usano tunneling***.

- **Differenze**: **molto più improntato sul concetto di mobilità**.
Esistono protocolli dedicati alla gestione del fatto che si tratta di un collegamento Wireless.
Ogni dispositivo mobile presenta una **scheda SIM**, a livello circuitale *garantisce autenticazione e quindi la differenziamo dall'IP e dal MAC*.
Forte impronta di business, mi devo abbonare ad un operatore di telefonia mobile per avere accesso alla rete mobile. Se mi trovo sotto il raggio di copertura del mio operatore **home network**, altrimenti **visited network**.

Componenti 4G e 5G

Si ha una **stazione base**, detta ***eNode-B***, *contenente la stazione radio che emette e riceve i segnali*. I dispositivi connessi nel raggio di copertura sono detti

User Equipments UE. *Ognuno di essi è autenticato da una* **Scheda SIM**, che memorizza anche un identificativo a 64 bit **IMSI** che **non solo identifica univocamente il dispositivo ma indica anche qual è la home network** di qeust'ultimo. Si ha poi un **nucleo EPC** *a cui sono connesse tutte le celle dello stesso operatore*.

Nel nucleo EPC troviamo:

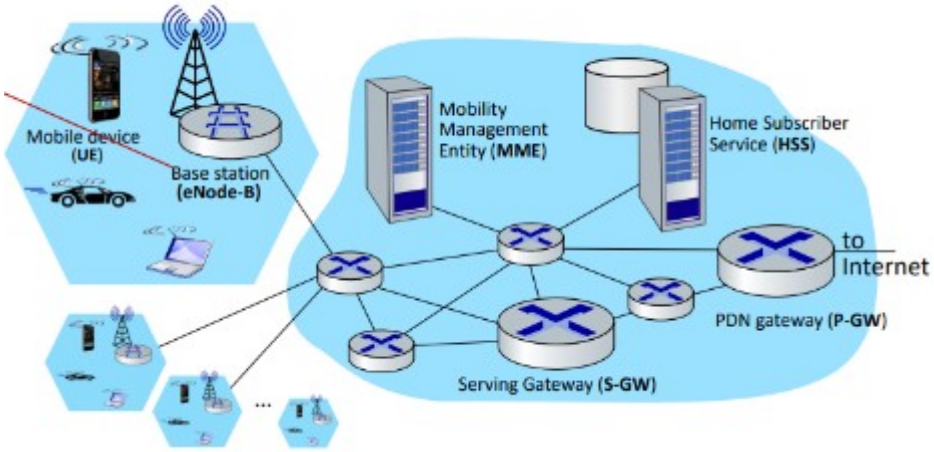
-**P-GW e S-GW**: si tratta di ***due router gateway***, dedicati quindi all'interconnessione del mio provider con il resto di Internet. Perché due? *Immaginiamo che un mittente invii a un dispositivo mobile sotto il mio dominio un pacchetto, e che nel mentre il mio dispositivo si sta spostando tra le varie celle con le stazioni base. Se cambia cella l'indirizzamento basato su destinazione non funziona più*!

Si adotta quindi una strategia di **tunneling**. Il pacchetto, **arrivato al P-GW**, **viene inoltrato attraverso un tunnel** (incapsulato in un altro pacchetto che manipola l'instradamento) **verso l'S-GW**. **Dall'S-GW si costruisce un altro tunnel verso la stazione base in cui il mio dispositivo si trova attualmente**

-**HSS**: ***Home subscriber service***, *si tratta di un database contenente i dati relativi agli UE abbonati a quello specifico provider di telefonia mobile*.

-**MME**: ***Mobility Management Entity***, come suggerisce il nome si occupa principalmente del servizio sulla mobilità. **L'MME** traccia il percorso del dispositivo tra le varie celle del provider e ne gestisce l'handover, occupandosi del **setup dei tunnel dal dispositivo al P-GW**.

Ripetizione Tunneling: a livello logico P-GW e S-GW sono direttamente connessi (più a basso livello questo collegamento diretto è realizzato tramite tunneling tra vari router con destinazione finale S-GW). Quando S-GW riceve il pacchetto fa il decapsulamento e vede il destinatario, l'MMI ha già fatto il setup del tunnel verso quel destinatario che si trova in una certa stazione base: ancora una volta incapsulamento tunneling verso quella specifica stazione base (che a sua volta decapsula, vede il cellulare di destinazione e glielo manda).



Nel 4G (anche detto LTE) vi è **netta separazione tra piano di dati e piano di controllo**.
- **Piano di controllo**: si hanno nuovi protocolli specifici per la gestione della mobilità, sicurezza e autenticazione
- **Piano di dati**: nuovi protocolli a livello fisico e di collegamento, uso estensivo di tunnel per gestire la mobilità.

Tra i vari protocolli LTE se ne hanno 3 specifici e importanti a livello di collegamento:
- **Packet Data Convergence**: per la compressione degli header e la cifratura;
- **Radio Link Control Protocol** (***RLC***): per la frammentazione/riassemblaggio dei frame e il trasferimento affidabile
- **Medium Access**: richiesta e uso di slot per la trasmissione radio (OFDM, tecnica simile a Bluetooth).

Passaggio al 5G anche per consentire scenari di affidabilità massima e latenza bassa (immagina un chirurgo che opera a distanza).

Come per bluetooth gli UE possono entrare in **sleep mode** per preservare la batteria.

Mobilità

Dal punto di vista della rete si ha ***mobilità nulla*** se il dispositivo si sposta tra reti di accesso diverse ma è spento, ***bassa mobilità*** se è acceso e si sposta lungo la stessa rete di accesso, ***media mobilità*** se si sposta tra reti di accesso diverse ma sotto il dominio dello stesso operatore e ***alta mobilità*** se si sposta tra reti di accesso di diversi operatori.

Ma se quindi un dispositivo si sposta da una rete all'altra, come gestisco la cosa? Due approcci:
1) **Lascio che i router gestiscano la situazione**, ipotizzando che le tabelle vengano aggiornate di volta in volta che il nodo mobile con identificativo fisso cambia sottorete. ***Questa soluzione non è scalabile per miliardi di dispositivi, quindi impraticabile***.
2) **Lascio che gli end system gestiscano la situazione**.
Quest'ultima in quanto scalabile è la soluzione effettivamente applicata per implementare la mobilità. Esistono due approcci:
- **Instradamento indiretto**: *l'invio dei pacchetti da corrispondente a dispositivo mobile* ***è intermediato dalla Home Network*** *del dispositivo mobile*
- **Instradamento Diretto**: il corrisponende ottiene l'indirizzo attuale del dispositivo mobile nella rete visitata e invia i pacchetti direttamente al dispositivo ***usando il classico approccio di inoltro basato su destinazione***.

Come visto in 4G/5G esiste una differenza sostanziale tra **home network** e **visited network**. La prima rappresenta la rete del carrier (operatore) a cui il dispositivo è abbonato, l'HSS memorizza le informazioni circa l'identità del dispositivo e dei servizi abilitati.
La seconda rappresenta una qualsiasi altra rete, diversa da quella domestica (home) a cui mi associo.

Affrontare la mobilità

Un dispositivo mobile è provvisto di un **identificativo IMSI a 64 bit univoco e un IP fisso legato alla sua home network**. Quando il dispositivo cambia rete di accesso allora si porta dietro l'IMSI ma cambia IP.

Quando ciò accade, **il dispositivo nella visited network si associa al Mobility Manager della rete visitata**, questo poi **registra la posizione del dispositivo mobile nell'HSS della home network (IMSI fornisce anche la home network!)**.
In questo modo il Mobility Manager della rete visitata sa del dispositivo mobile mentre l'HSS sa la posizione attuale del dispositivo.

E se voglio ora inviare un pacchetto al dispositivo mobile?

Instradamento indiretto

Il pacchetto è inoltrato all'IP permanente del destinatario con instradamento classico basato su destinazione, giungendo quindi alla home network.
Il pacchetto in particolare giunge al router gateway, per cui è già stato impostato un tunnel (grazie alle informazioni presenti nell'HSS della home network e il lavoro svolto dall'MME) **verso l'S-GW della rete visitata che potrà** inoltrare il pacchetto al dispositivo mobile.

Il dispositivo potrà a questo punto rispondere in due modi: o viene inviata la risposta direttamente al mittente oppure attraverso la rete domestica.
Questo metodo di instradamento è detto **triangolare**, poiché si passa prima per la home network per poi giungere alla vera destinazione.

Contro: il routing indiretto è **inefficiente se il corrispondente e il dispositivo mobile si trovano sulla stessa rete** (dovranno comunque dialogare passando per il gateway).

Pro: il dispositivo mobile nello spostarsi tra visited networks risulterà **trasparente** al corrispondente! (nel registrarsi ad una nuova rete visitata ci pensano l'HSS e l'MME a gestire il tutto). **Quindi connessioni (come TCP) in corso tra corrispondente e dispositivo mobile possono essere mantenute**!

Quindi 1) **dispositivo nella nuova stazione base fornisce IMSI**
2) **Configurazione piano di controllo**: **MME locale scrive in HSS home del dispositivo il fatto che è nella visited network**
3) **Configurazione piano di dati**: **MME configura tunnel di inoltro da RGW**

home a SGW locale e da SGW locale a dispositivo mobile, i dati possono essere trasmessi con trasparenza.

Routing diretto

Vediamo ora l'approccio del **routing diretto**.
Ciò che accade sostanzialmente è che invece di passare per dei tunnel il corrispondente si fa comunicare di volta in volta la posizione del dispositivo mobile a cui deve inviare pacchetti.

Questo approccio non ha i contro del routing indiretto, ma presenta l'enorme **difetto per cui il dispositivo mobile non risulterà trasparente al corrispondente** (quindi non posso mantenere connessioni). Il corrispondente infatti deve tenersi aggiornato chiedendo di volta in volta la posizione del dispositivo mobile in caso questo passi ad una nuova visited network.

Per connettersi a una BS questa invia in broadcast un segnale di sincronizzazione primario ogni 5ms su tutte le frequenze.
Il mobile node trova un segnale di sincronizzazione primario individuando informazioni trasmesse dalla BS come larghezza di banda, configurazioni…
Il mobile sceglie tra le varie BS quella a cui associarsi preferendo ad esempio quella del suo operatore
Altri passaggi per l'autenticazione e la configurazionje del piano di dati.

Il vantaggio nell'avere un tunnel da PGW a SGW e da SGW a dispositivo mobile è che se il dispositivo cambia semplicemente stazione base sotto il dominio dello stesso operatore allora mi basta cambiare il tunnel SGW – dispositivo mobile, il primo lo lascio inalterato! (arrivo sempre al nucleo della stessa visited network)