I dati da mandare vengono raggruppati in **pacchetti**. Quelli che si occupano di trasmetterli si chiamano **packet switches**. La maggior parte di essi usano la **store and forward** transmissio, cioè devono aspettare di ricever l'intero pacchetto prima di processarlo. Pericò i pacchetti in uscita si potrebero impilare nell'**output buffer** (o queue). Se l'output buffer è pieno, avviene un **packet loss**. Inoltre, ogni router ha una **forwarding table** per sapere dove mandare un pacchetto in base al suo indirizzo nell'header. Opposto al packet switching è il **circuit switching**, dove come nei telefoni prima viene stabilita una connessione, e poi vengono liberamente mandati i dati. Per stabilire più circuiti su uno stesso link si può usare la **FDM** (frequency division multiplexing), in cui a un circuito viene attribuita solo una frazione della bandwidth del link(proprio come nella radio), oppure la **TDM** (time division multiplexing), in cui il tempo in un link viene diviso in segmenti, e ogni circuito può usare tutta la bandwidth ma solo nei suoi segmenti. I due problemi principali del circuit switching sono i **periodi di silenzio** e le grandi difficoltà di implementazione. Il vantaggio è che permettono scambio di dati con minore ritardo.
Ogni nodo, prima di ricevere e mandare pacchetti, ha vari tipi di delay: **processing delay**: il tempo che ci vuole ad analizzare l'header di un pacchetto, a controllare eventuali errori. Di solito intorno al microsecondo; **Queueing delay**: dipende da quanti pacchetti devono essere trasmessi. Varia da 0 a millisedondi; **Transimission delay**: Il tempo che ci vuole a trasmettere un pacchetto attraverso il link. Se L è il numero di bit da mandare, R è la capacità del link, allora li tempo è L/R. Di solito varia da qualche microsecondo a qualche millisecondo; **Propagation delay**: è il tempo che ci mette il pacchetto ad arrivare a destinazione. Se d è la distanza e s la velocità del link, il tempo è d/s. s dipende dal materiale del link, ma spesso è vicino alla velocità della luce.
**Traffic Intensity**: (L*a)/R, dove L è il numero di bit di ogni pacchetto, a è la media di quanti pacchetti al secondo arrivano, R è la velocità di trasmissione di un pacchetto. Se la traffic intensity è maggiore di 1, allora la queue d'attesa diventerà infinita. Tuttavia è da notare che è una funzione esponenziale: più è vicina a 1, più la lunghezza della queue cresce esponenzialmente. Quindi in ogni network dovrebbe sempre essre strettamente minore di 1, ma più bassa è meglio è. **Traceroute**: un semplice programma che manda N pacchetti speciali a una destinazione verso cui deve fare N - 1 **hop**. Ogni router che riceve il suo pacchetto speciale rimanda indietro un messaggio di successo (anche la destinazione). Traceroute misura il tempo che ci mette ogni messaggio a tornare. Ripete l'esperimento 3 volte. **Throughput**: velocità a cui un file viene mandato da sorgente a destinazione. Se non c'è traffico, è uguale alla velocità di trasmissione del link più lento attraversato.
**Protocol Stack**: insieme dei protocolli di ogni layer. Il protocol stack di internet è composto da 5 layer: **Application**: un protocollo applicativo è distribuito su vari end systems, con un'applicazione su ogni end system. I pacchetti scambiati nel protocollo applicativo si chiamano messaggi. **Transport**: i protocolli di trasporto si occupano di trasportare messaggi fra due applicazioni. I pacchetti scambiati si chiamano segmenti. **Network**: IP protocol e vari altri routing protocol. Si occumano di prendere i segmenti e farli arrivare a destinazione. I pacchetti scambiati prendono il nome di datagram. **Link**: Si occupa di far muovere i datagram tra i vari nodi della rete. Alcuni esempi sono Wi-Fi e Ethernet. Può essere che un datagram utilizza diversi link protocol nel suo viaggio. I pacchetti scambiati sono i frame. **Physical**: dipendono dal mezzo usato per trasmettere. Si occupano di mandare bit per bit i frame attraverso un link.
**Encapsulation**: ogni layer prende il pacchetto del layer superiore e o incapsula aggiungendo informazioni. Il pacchetto incapsulato si chiama **payload**, mentre le informazioni aggiunte sono l'header. Gli host implementano tutti e 5 i layer. I router solo i primi 3. I link switches solo i primi 2. Nel tardo 1970, nel modello OSI, c'erano altri due layer, presentation e session layer; il primo si occupava di descrivere i dati trasferiti(compressione...), mentre il secondo si occupava di delimitare e sincronizzare lo scambio di dati. Oggi questi servizi sono implementabili a livello applicativo.
**Socket**: interfaccia software che permette ai processi di mandare e ricevere messaggi. In pratica è l'interfaccia fra l'pplication layer e il transport layer. Il programmatore ha il totale controllo della parte applicativa, ma della parte di trasporto può decidere solo il tipo di protocollo e alcuni parametri. Le architetture applicative sono distinte in due grosse categorie: **Client-server architecture**: ci sta un host, il server, sempre attivo, che soddisfa le richieste di vari host temporanei, i client. Se un solo server non basta, si possono impiegare vari datacenter. Due client non possono comunicare fra loro senza passare attraverso il server. Il server ha IP fisso. **Peer-to-peer architecture**: l'applicazione usa comunicazione diretta tra coppie di host temporanei chiamati peer. Nesun server. I vantaggi principali del P2P sono la self-scalability, la failure prevention e i bassissimi costi. Tuttavia nel futuro ci saranno vari problemi: non sono ISP-friendly (richiedono anche molto upstream), non è detto che siano sicuri, e non è detto che gli utenti vogliano condividere.
**UDP** sta per User Datagram Protocol. Fornisce i servizi di multiplexing/demultiplexing e di integrity checking. **Demultiplexing**: il transport layer si occupa, quando l'host riceve un messaggio, di inviarlo al processo applicativo giusto dell'host. E' in grado di farlo grazie alla presenza dei **socket**. **Multiplexing**: il transport layer raccoglie tutti i messaggi uscenti dai vari socket dell'host e li raggruppa in vari grandi segment. Un messaggio UDP è quindi formato dalla source port number e dalla destination port number (in caso ci sia bisogno di una risposta), e da altri due valori. Dopo viene passato al network layer, che ci scrive source IP e destination IP. Importante: un socket UDP viene identificato solamente dalla coppia destination IP e destination port number.
UDP è il protocollo di trasporto più semplice che esista. Viene usato spesso da **DNS**. Non ci sta nessun tipo di handshake. Se dopo un certo tempo non ricevo nessuna risposta, allora viene mandata un altra richiesta.
Vantaggi principali di UDP sopra a TCP: 1) Controllo sui dati inviati: UDP aggiunge solo un piccolo header ai dati e passa tutto al network layer. Non ci sta nessun tipo di congestion control, quindi nessun eventuale attesa. Inoltre, non è detto che debba avere come condizione l'integrità dei dati. 2) Nessuna connessione: TCP usa un three-way handshake prima di inviare dati, che introduce notevoli ritardi. 3) Nessuno stato di connessione: TCP salva molti dati sullo stato di una connessione, UDP invece zero. 4) Piccolo overhead dei pacchetti: un pacchetto TCP ha 20 byte di overhead, mentre un UDP solo 8 byte.
L'header UDP è composto da 4 campi, ognuno largo 2 byte. Oltre a source e destination port, ci stanno length e checksum. La **checksum** è calcolata come il complemento a uno della somma di tutte le parole a 16 bit del messaggio. Il ricevente, se somma la checksum a tutte le parole del messaggio, dovrebbe ottenere tutti uni. L'error checking è stato introdotto poichè non è detto che tutti i link implementino error checking fra di loro, e ci potrebbero anche essere altri generi di errori. UDP non offre nessuna maniera di correzione dell'errore.
**HyperText Transfer Protocol**: Una **Web Page** (documento) è formata da **oggetti**. I client sono i **Web browsers**, invece i server sono i **Web server**. HTTP usa TCP per mandare messaggi, quindi non si deve preoccupare di eventuali errori. Siccome HTTP manda i file richiesti senza salvarsi nulla del client, prende il nome di **stateless protocol**. **Round Trip Time**: tempo che passa da quando il client manda un pacchetto al server a quando una risposta torna al client. HTTP può essere configurato per usare la **non persistent connection**, cioè creo una nuova connessione TCP per ogni file che devo mandare. Potrebbe essere lento poichè una web page è formata da vari file, ogni file ci mette 2 RTT + tempo di trasmissione del file. Tuttavia, di default usa la **persistent connection**, cioè dopo aver risposto a ua richiesta, il server non chiude la connnessione, ma può rispondere ad altre richieste. Non solo. Le richieste possono esere fatte tutte di seguito, senza aspettare le risposte (**pipelining**). Una request in HTTP è composta da una o più linee codificate in ASCII, ognuna seguita da un carriage return e da un line feed. La prima linea è la **request line**, formata da 3 campi: il campo del metodo("GET, POST, HEAD, PUT, DELETE"), il campo URL, e il campo della versione ("HTTP/1.1"). Le linee successive sono le **header lines** ("host, connection, User-Agent, Accept-Language") per fornire varie informazioni. Poi segue una riga vuota, e poi ci sta l'entity body, che contiene informazioni nel caso del POST. Una risposta invece è formata da una **status line**, composta da 3 campi: versione HTTP, codice di status e status message. Dopo ci sono le header lines, con campi come ("Date, Server, Last-Modified, Content-length, Content-Type"). Alcuni status code: 200 OK (Tutto a posto); 301 MOVED PERMANENTLY (la risorsa si trova nell'URL specificato dal campo "Location"); 400 BAD REQUEST; 404 NOT FOUND; 505 HTTP VERSION NOT SUPPORTED. I **cookie** invece non sono altro che del testo che il browser salva accanto al nome dell'host qunado riceve "Set-cookie:" nell'header della risposta. Vengono rimandati quando ci si riconnette a quell'host. Le **Web Cache** sono dei proxy server installate dalle ISP per tentare di ridurre il traffico (e ravvelocizzarlo motlo, se si fanno molte cache hit). Per capire se il file della cache è vecchio, la cache può usare il **conditional GET**, con il campo "If-modified-since". Se la copia non è vecchia, il server risponderà con 304 NOT MODIFIED.
**File Transfer Protocol**: molto simile a HTTP, anche lui usa TCP. Una grande differenza è che usa una doppia connessione TCP, una control connnection e una data connnection. I file vengono inviati nella data connnection, mentre i comandi nella control. La data connection viene usata per inviare un solo file, poi viene chiusa. Nel caso, ne viene aperta un altra. Un'altra differenza è che il server mantiene uno **state** per ogni client, in cui si salva la cartella su cui sta, i suoi permessi etc. Come in HTTP, gli header FTP sono formattati in ASCII. I comandi più comuni sono: "USER, PASS, LIST, RETR, STOR". A ogni comando, segue una risposta. Niente pipeline (?).
**SMTP**: si occupa di trasferire le mail dal server del sender al server del ricevente. E' molto vecchio, ha alcune cose strande, come per esempio il body della mail deve essere per forza codificato in 7-bit ASCII, e quindi non posso mandare audio, video etc. SMTP trasferisce le mail tra due server senza server intermediari, direttamente dal mittente al destinatario. Se il server del destinatario è irraggiungibile, allora SMTP riprova dopo un tot di tempo. Ovviamente, usa TCP. La porta usata è la porta 25. Una volta stabilita la connessione, ci sta un po' di handshaking, e dopo invia il messaggio. I comandi che si scambiano tra server sono del genere ("HELO, MAIL FROM, RCPT TO, DATA, QUIT"). Una linea che consiste di un solo punto corrisponde alla fine della mail. Il server risponde a ogni comando. SMTP usa persistent connection, e quindi se deve inviare varie mail lo fa attraverso una singola connessione TCP. Le differenze principali con HTTP: HTTP è principalmente un **pull protocol**, usato per scaricare risorse dai server. Invece, SMTP è principalmente un **push protocol**, usato per mandare dati ai server. Inoltre, HTTP non forza il body del messaggio a essere codificato in 7-bit ASCII. Una terza differenza sta nel fatto che ogni oggetto di una pagina Web viene gestito con un messaggio HTTP diverso (uno per il testo, uno per le immaggini, etc.) **Mail formats**: Tutte le mail, all'inizio del messaggio, hanno alcune header lines, come ("To:, From:, Subject:"), e dopo una riga bianca, contengono il vero e proprio messaggio. Queste linee sono diverse dai comandi SMTP usati per l'handshaking. Perchè usiamo dei mail server gestiti dagli ISP? Poichè se avessimo un mail server sul nostro computer, significherebbe che il nostro computer dovrebbe rimanere sempre acceso, e soprattutto, connesso a Internet. Quindi serve anche un protocollo per scaricare mail da un mail server. **POP3**: inizia quando il client apre una connessione TCP sul mail server sulla porta 110. Tre fasi: autorizzazione (invio di user e password), transaction (scarico messaggi, ed eventualmente eliminazione dal server e altre azioni varie), e update (chiude la connessione, elimina le mail marcate da eliminare dall'utente). A ogni comando segue una risposta, che può essere ("+OK, -ERR"). I comandi possono essere ("user, pass, list, retr, dele") etc. POP3 tiene uno **state** per ogni connessione, cioè quali messaggi sono stati marcati per l'eliminazione. Si può configurare che il server cancella un messaggio dopo che è stato scaricato ("Download-and-delete"), oppure no ("Download-and-keep") **IMAP**: un server IMAP associa a ogni mail una cartella, all'inizio tutte nella **inbox**, e poi possono essere spostate dall'utente. Ha anche funzioni di ricerca. Al contrario di POP3, IMAP si tiene lo **state** anche quando la connessione viene chiusa, come le cartelle e il loro nome. Un altra feature è che un user agente può richiedere di scaricare solo una parte della mail (come l'header) nel caso di connessioni lente. **Web-based Email**: il più diffuso oggi, sfrutta il protocollo HTTP.
**Domain Name System**: è sia un sistema distribuito di database, sia il nome di un application layer protocol. DNS usa UDP e va sulla porta 53, serve a tradurre hostname in indirizzi IP, ed è costantemente utilizzato da altri application layer protocol come HTTP e FTP. Inoltre, offre anche i servizi di *host aliasing* (vari nomi per un singolo host), *mail server aliasing*, *load distribution* (quando un'azienda ha vari server con vari IP, i server DNS restituiscono gli IP a rotazione per distribuire il carico sui vari server). **Root DNS server**: ce ne sono solo 13, la maggior parte negli Stati Uniti. **Top-level Domain server**: sono responsabili per i domain principali come .com, .net, .edu, .it etc. **Authoritative DNS Server**: un server DNS specializzato solo per gli host di una o più compagnie. Dopodichè, ci sono i **local DNS server**, che non sono per forza nella gerarchia DNS: ogni ISP ne ha uno, e servono per collegarsi ai DNS più grossi. Le query a un server DNS possono essere **recursive** (il local DNS interroga il root, che interroga il top-level domain server, che interroga l'authoritative) oppure **iterative** (il local DNS interroga il root, poi, dopo aver ricevuto una risposta, interroga il top-level domain etc..) **DNS Caching**: Spesso vengono fatte le stesse identiche query ai local DNS, che quindi si salvano i record. A volte si salvano anche gli IP dei server top-level domain, per evitare di passare attraverso il root. I server DNS si salvano i dati in forma di **Resource Records (RR)**, che hanno la forma (Name, Value, Type, TTL). TTL non è altro che il tempo di vita di un record (in una DNS cache, dopo quanto tempo deve essere rimosso). Se Type=A, allora Name è l'hostname e Value è l'indirizzo IP. Se Type=NS, allora Name è un domain ("foo.com") e Value è l'hostname dell'Authoritative DNS che conosce il suo IP. Se Type=CNAME, allora Value è il vero hostname di un alias di nome Name. Se Type=MX, allora Value è il vero hostname di un server mail che ha come alias Name. **DNS Messages**: sono formati da un header, di 12 byte, con la query, varie flags, e la lunghezza dei vari pezzi del body. Nel body, ci sono le questions (hostname e Type), le answers (Type, Value, TTL), le authority, e le additional informations.
**BitTorrent**: L'insieme di peer coinvolto nella distribuzione di un particolare file prende il nome di torrent. I peer in un torrent stanno condividendo chunk di quel file di dimensioni prefissate (di solito 256 KB). Mentre un peer scarica chunk, allo stesso tempo condivide quelli che ha. Ogni torrent ha un'infrastruttura chiamata **tracker**. Quando un peer si unisce a un torrent, si registra sul tracker, e periodicamente informa al tracker che ci sta ancora. In questo modo, un tracker ha la lista di tutti i peer del torrent. Appena un peer si unisce, il tracker manda una lista di una parte degli IP di altri peer (di solito 50). Il peer prova a formare connessioni TCP con questi altri 50. Alcuni di questi se ne andranno, ma altri si connetteranno al nuovo peer. Periodicamente, un peer chiede ai suoi **neighboring peers** la lista dei chunk che posseggono. Dopodichè, inizia a fare richieste ai suoi vicini, partendo dai chunk più rari. Inoltre, deve iniziare a condividere i chunk che ha. Una richiesta di un chunk in BitTorrent viene soddisfatta se il peer richiedente è **unchocked**, cioè è uno di quei peer che ti stanno inviando più dati. Inoltre, ogni peer, ogni 30 secondi, sceglie a caso un altro peer e soddisfa le sue richieste. Questo peer preso a caso diventa **optimistically unchocked**. Se questo nuovo peer ha dei file che piacciono al peer originario, questi due potrebbero iniziare a scambiarsi dati velocemente e diventare unchocked. Questo peer preso a caso diventa **optimistically unchocked**. Se questo nuovo peer ha dei file che piacciono al peer originario, questi due potrebbero iniziare a scambiarsi dati velocemente e diventare unchocked. Tuttavia, Bittorrent non è un puro P2P, in quanto si basa sul server tracker.
**Stop&Wait**: Un protocollo Stop&Wait ha un sistema di reliable data transfer basato sull'error checking. In pratica, mando un messaggio e aspetto una risposta di tipo `ACK` o `NAK`. In base a quello, mando un altro messaggio o ripeto il precedente. Il ricevente invece ha un singolo stato, riceve un pacchetto e manda conferma negativa o positiva. Inoltre, ci deve essere anche un sistema decente di controllo di errori. Tuttavia dobbiamo aggiungere un campo `sequence number`, da 1 bit, per indicare se il pacchetto mandato è uno nuovo oppure è una retrasmissione. Questo risolve il problema di capire che fare se arriva un pacchetto di conferma `ACK` corrotto: semplicemente rimando il pacchetto di nuovo.
**Alternating bit protocol**: uguale allo stop and wait, solo che mittente, dopo un certo tempo che non ha avuto nessuna risposta, rimanda il pacchetto.
Tuttavia, il metodo stop and wait è molto lento, poichè gli host stanno moltissimo tempo ad aspettare senza fare nulla. La soluzione è inviare pacchetti anche se non si è ricevuto nessuno acknowledgment.
**Go-Back-N**: è uno sliding window protocol, cioè ammette l'attesa di al massimo $N$ acknowledgments. In pratica posso la mia window di pacchetti che non so se sono arrivati bene oppure no è lunga $N$. Questo limite esiste per il congestion e il flow control. E' necessario un field con il sequence number del pacchetto. Il mittente, ogni volta che manda un pacchetto, deve controllare se la window è piena. Inoltre, si deve tenere conto di fino a quale `ACK` ha ricevuto (**cumulative acknowledgment**, cioè se ricevo un `ACK 30` vuol dire che il destinatario a ricevuto bene tutti i pacchetti fino al 30). Inoltre, deve avere timer che se scade, rimanda tutti i pacchetti di cui non ha avuto `ACK`. Appena ne ricevo uno, resetto il timer. Il ricevente, se riceve pacchetti in un ordine sbagliato, li butta. Non si salva pacchetti che avrebbe dovuto ricevere dopo, perchè tanto se ne manca qualcuno prima, vuol dire che anche tutti quelli dopo verranno rimandati
Tuttavia, Go-Back-N ha qualche difetto: se la window size e grossa, e invio molti pacchetti nella pipeline, mi basta che ci sia un solo errore e li devo rimandare tutti quanti.
**Selective Repeat**: tecnica uguale a Go-Back-N, solo che rimanda solo i pacchetti (potenzialmente) coinvolti nell'errore. Una grossa differenza è che il receiver deve fare l'acknowledgment di ogni singolo pacchetto arrivato, non è più cumulative. Un'altra differenza è che i pacchetti non ordinati non vengono buttati, ma salvati per essere usati quando arrivano tutti quelli mancanti (e viene fatto l'acknowledgment). Importante! la window size deve essere larga al massimo quanto la metà del massimo sequence number, altrimenti si incorre in problemi in quanto non si capisce se un pacchetto è nuovo o è una retrasmissione.
Un problema che capita nella rete è che potrebbe arrivare un pacchetto vecchissimo con lo stesso sequence number di uno nuovo che sta arrivando. Come faccio a scartarlo? Per esempio, TCP decide che un pacchetto può vivere al massimo 3 minuti nella rete.
**Demultiplexing**: uguale a UDP; l'unica differenza è che un socket TCP viene identificato univocamente da quattro valori: source IP e port number, destination IP e port number. Inoltre, la porta 12000 (di solito) è la **welcoming port**, da dove iniziano tutte le connessioni TCP. Una volta connesso a quel socket, ci si mette d'accordo su quale porta usare.
**TCP**: offre un servizio **full-duplex**, cioè se A manda dati a B, anche B può mandare dati ad A. Inoltre, è **point-to-point**, cioè una connessione è solo fra una singola sorgente e una singola destinazione.
Il segmento TCP è strutturato in questo modo: 32-bit: sequence number e acknowledgment number. 16-bit: source port, destination port, receive window (flow control), checksum, urgent data(pointer all'ultimo byte dell'urgent data). 4-bit: header length. 1-bit: ACK (se sto mandando un acknowledgment), RST, SYN, FIN (tutti e tre flag di connessione, PSH(il receiver deve mandare subito i dati all'application layer), URG(flag per dati urgenti).
TCP usa i sequence e acknowledgment number non in base al numero del pacchetto, ma in base al numero del byte. Gli acknowledgment number sono un pochino strani: rappresentano il byte che mi aspetto che arrivi dall'altro host. Inoltre, gli acknowledgments sono **cumulative**. Tuttavia, se arrivano pacchetti in un ordine sbagliato, TCP non definisce nessuna politica precisa: si possono sia buttare sia tenere aspettando che arrivino quelli mancanti (molto spesso la seconda politica è quella più diffusa)
Importante! TCP non usa **NAK**!!
TCP usa i timeout per rimandare i pacchetti, e la durata viene decisa attraverso la stima del **round-trip-time**: $\texttt{EstimatedRTT} = (1-\alpha)\cdot \texttt{EstimatedRTT} + \alpha \cdot \texttt{SampleRTT}$. Di solito vale che $\alpha = 0.125$. Inoltre, abbiamo anche una stima della differenza tra `EstimatedRTT` e `SampleRTT`, che si calcola come $\texttt{DevRTT} = (1-\beta)\cdot \texttt{DevRTT} + \beta \cdot |\texttt{SampleRTT} - \texttt{EstimatedRTT}|$. Alla fine, il timeout interval si calcola come $\texttt{TimeoutInterval} = \texttt{EstimatedRTT} + 4\cdot \texttt{DevRTT}$. All'inizio, di solito, è settato a 1.
Modifiche possibili al TCP: **doubling timeout interval**: dopo un evento di timeout expiration, subito dopo aver rimandato il pacchetto con il seq number più piccolo non ancora confermato, il timeout interval viene raddoppiato, per evitare congestione. Infatti, nella maggior parte dei casi il timeout avviene a causa della congestione. **Fast retransmit**: se un sender riceve tre acknowledgment dello stesso dato già inviato, allora rimanda tutto senza aspettare il timeout. Questo succede perchè siccome TCP non ha `NAK`, vuol dire che c'è stato un errore e quel tipo particolare di dato non è proprio arrivato.
Fondamentalmente, TCP è un Go-Back-N protocol, ma con alcune grosse differenze: non è detto che butta i pacchetti non in ordine; se un accade un timeout expiration TCP rimanda solo quel pacchetto invece che tutti quanti quelli di cui non ha ricevuto un `ACK`
**Flow Control**: ogni host che intrattiene una connessione TCP si salva dei buffer per i dati della connessione. Il buffer del ricevente serve per esempio a tenersi i dati mentre l'application layer li legge. Il flow control serve a non riempire quel buffer, mandando i dati alla stessa velocità con cui l'applicazione li legge. Il sender si tiene una **receive window**, che è uguale allo spazio libero

del buffer dove salva i dati il ricevente. La receive window non deve mai arrivare a 0. Per fare questo, basta essere sicuri che `LastByteSent - LastByteAcked` $\leq$ `ReceiveWindow`. Nel caso la ReceiveWindow dovesse arrivare a 0, allora il sender è autorizzato a inviare pacchetti grossi un byte per ricevere `ACK` per sapere se si è liberata.
**TCP connection**: 1) Il client manda un segment vuoto con la flag SYN attiva, e un valore nel sequence number random. 2) Il server riceve, crea i buffer e le variabili, e poi manda un segmento vuoto al client con la flag SYN attiva e il sequence number ricevuto più uno come acknowledgment number. Nel suo sequence number viene messo un valore a caso scelto dal server. Questo si chiama **SYNACK segment**, poichè significa che il server accetta la connessione. 3) Quando riceve il SYNACK, il client crea buffer e variabili. Poi rimanda al server un altro segment al server, con il sequence number scelto dal server più uno nell'acknowledgment number, e con la flag SYN a 0, dato che la connessione è ormai attiva. Questo terzo segment può già contenere dati. Questo processo si chiama **three-way handshake**. Per spegnere una connessione, il client (o il server) deve mandare un segment con il FIN a 1, e attendere l'`ACK` del server. Poi anche il server risponde con un segment con il FIN a 1, e aspettare l'`ACK` del client. A questo punto, tutte le risorse possono essere deallocate.
Gli stati che una connessione TCP (client) può avere sono: CLOSED, SYN_SENT, ESTABLISHED, FIN_WAIT_1 (aspetto ack del mio FIN), FIN_WAIT_2 (aspetto il suo FIN), TIME_WAIT ((ri)mando ack del suo FIN ricevuto). TIME_WAIT no è infinito, si disattiva da solo dopo circa trenta secondi o un minuto.
Gli stati che un server può avere sono: CLOSED(server spento), LISTEN (aspetto connessioni), SYN_RCVD, ESTABLISHED, CLOSE_WAIT(ho ricevuto un FIN e mando il mio), LAST_ACK (aspetto `ACK` del mio FIN).
**Principi di congestion control**: Scenario 1: la congestione avviene nel caso il throughput del sender è vicino a R/2, dove R è la capacità del link. Si incorre in delay molto alti a causa delle queue nel router. Scenario 2: alcuni pacchetti devono essere reinviati dal sender, e nel router il buffer è finito. Se il sender spara moltissimi pacchetti, il router ne perde molti a causa del suo buffer limitato, e il sender ne deve ritrasmetter altrettanti. Quindi si perde tempo a mandare in giro copie inutili di pacchetti. Scenario 3: un router deve essere condiviso da due connessioni. Magari però una di queste due connessioni è più vicina, e manda dati più velocemente. Piano piano, il router si riempie di dati di questa stessa. Tutti i pacchetti dell'altra, che arrivano da lontano, vengono buttati. Tutti la strada che hanno fatto fino a quel momento è stata inutile. La seconda connessione non parlerà mai.
Ci sono due modi principali di gestire il congestion control: End-to-End, come quello usato dal TCP, dove il network layer non da nessuna informazione riguardo la congestione, e se la devono sbrigare gli host tra di loro; Network-assited, dove il network layer comunica a ogni host lo stato della congestione.
**TCP congestion control**: TCP ha una variabile, la **congestion window**, messa in modo tale che `LastByteSent - LastByteAcked` $\leq$ `min{CongestionWindow, ReceiveWindow}`. Fatta in questo modo, l'output rate del sender può essere approssimato come (CongestionWindow) / RTT byte/sec. Come fa tuttavia ad accorgersi di un **packet loss**? Semplice: o scade un timeout, oppure riceve tre `ACK` duplicati.
**Slow start**: all'inizio di una connessione, vale che CongestionWindow = 1 MSS (dove MSS è la grandezza massima di un pacchetto). Ogni `ACK` ricevuto, la CongestionWindow viene aumentata di 1 MSS. In questo modo, ogni RTT, la grandezza di CongestionWindow raddoppia. Tre modi in cui smette di crescere, il primo: appena avviene un packet loss, la CongestionWindow viene resettata a 1 e ricomincia il processo, viene tuttavia salvato il valore (CongestionWindow / 2) in `sstresh`; secondo modo: quando CongestionWindow è ad un valore maggiore o uguale a sstresh, slow start finisce e si passa a congestion avoidance; terzo modo: il sender riceve tre `ACK` duplicati, e deve fare una fast retransmit, e passa allo stato Fast Recovery.
**Congestion avoidance**: In questo stato, ogni acknowledgment fa aumentare solo di 1 la CongestionWindow. Quando mi accorgo di un packet loss, resetto CongestionWindow a 1 e riaggiorno sstresh alla metà del valore di CongestionWindow precedente, e passo allo stato Slow Start. Se invece riceve 3 `ACK` duplicati, allora dimezzo il valore di CongestionWindow e aggiorno sstresh, ed entro nello stato Fast Recovery.
**Fast Recovery**: il valore di CongestionWindow aumenta di 1 MSS per ogni `ACK` duplicato ricevuto. Quando arriva il primo nuovo `ACK`, allora abbassa CongestionWindow e ritorna nello stato Congestion Avoidance. Se ottengo un packet loss per timeout, allora resetto CongestionWindow a 1, riaggiorno sstresh e ritorno a Slow Start. Da ricordare che Fast Recovery non è strettamente necessaria al funzionamento di TCP.
Assumendo che i packet loss vengono indicati da 3 `ACK` duplicati invece che da un timeout, il TCP congestion control viene definito una **additive-increase, multiplicative-decrease (AIMD)** forma di congestion control
Le connessioni TCP sono "fair", nel senso che si autobilanciano. Due connessioni TCP sullo stesso link prima o poi raggiungeranno la stessa velocità di throughput.
Spesso un trucco che viene usato per mandare più velocemente dati è usare multiple connessioni TCP parallele: questo gioca a vantaggio di chi manda i dati perchè le connessioni TCP si autobilanciano, quindi se ne ho più di una posso "imbrogliare"

# CAP 4

**Forwarding** Ogni **router** ha la sua forwarding table. Il servizio network di internet prende il nome di **Best-Effort Service**. Ci sono due modi nel network layer di passarsi dati: sia with connections che connectionless (la stessa differenza che ci sta nel transport layer tra le connessioni TCP e UDP connectionless). Per le connessioni servono processi di handshaking etc. Per implementarle, serve una **Virtual Circuit Network**. Ci sta un path per arrivare da ogni router a un altro. Un pacchetto che appartiene a una Virtual Circuit ha un VC number nel suo header. La forwarding table di ogni router consiste in una tabella a cui a ogni possibile VC number dei pacchetti in arrivo corrisponde un'interfaccia di uscita (output link) e un VC number di output. Infatti, il VC number cambia nel corso del viaggio di un pacchetto. Questo è perchè altrimenti ci si impiegherebbe troppo tempo a mettersi d'accordo in tutta la rete su quali VC number usare. Inoltre, ogni router di una VC network deve tenersi le varie connection state information. Per formare una connessione in una VC network si deve passare attraverso 3 fasi: VC setup (comunico a chi mi voglio connettere; aspetto che la VC network crei un path per me), Data Transfer, VC Teardown (comunico che termino la connessione; la VC network aggiorna tutte le forwarding tables coinvolte di eliminare il path). L'unica grande differenza dal sistema di connessioni TCP è che mentre nel TCP solo due end system sono al cosciente di una particolare connessione, nelle VC tutti i router coinvolti in un path hanno le informazioni di quella connessione. I messaggi usati per configurare una VC si chiamano signaling messages, e i protocolli si chiamano signaling protocols.
**Datagram Network**: Niente path, niente connessioni. Un end system mette l'indirizzo di un certo pacchetto nel suo header e poi lo butta nel network, e non deve più preoccuparsene. Si è sviluppata così tanto per la sua semplicità di implementazione, e per la facilità con cui diverse tecnologie possono essere collegate fra loro.
**Longest Prefix Matching**: Nei router la forwarding table è formata da vari prefix, e a ognuno corrisponde un'interfaccia di uscita. Quando arriva un pacchetto, il router ne esamina l'indirizzo bit per bit, e lo manda attraverso l'interfaccia a cui corrisponde il longest prefix match.
Un router è composto da varie input ports, da varie output ports (che possono essere anche collegate se il router è bidirezionale), da una switching fabric, e dal routing processor. L'unica parte che esegue software è il routing processor, mentre tutto il resto fa parte del **forwarding**, implementato a livello hardware. Ogni input port ha la sua copia della forwarding table, e appena arriva un pacchetto fa un lookup hardware velocissimo (a volte anche usando cache ram) e capisce in quale output port va mandato. Dopo il pacchetto viene inoltrato nella switching fabric (se è libera). Il routing processor si occupa di aggiornare la forwarding table, con tutte le operazioni necessarie a farlo (calcolare percorsi etc) La **switching fabric** è fatta in 3 modi: via memory, dove cioè un pacchetto viene scritto in una memoria, e poi il routing processor lo manda in una output port (ma non deve fare calcoli che sono tutti già stati fatti nelle input port), via bus, dove cioè a ogni pacchetto viene messo un mini-header che indica quale output port usare, poi viene mandato su un bus che lo inoltra a tutte le output port, e solo quella giusta lo propaga (tuttavia sul bus ci può stare solo un pacchetto alla volta), e via interconnection network, dove cioè ci sta una rete interconnessa di bus con switch che si aprono e chiudono a seconda della strada che deve fare il pacchetto.
Sia le input che le output ports hanno dei buffer per salvarsi i pacchetti arrivati o da mandare. Se questi buffer si riempiono, si occorre in packet loss. La lunghezza di questi buffer di solito è c*rtt, dove c è la capacità del link e rtt è l'average round trip time. Ci sono vari algoritmi per decidere quale pacchetto servire nella queue dei pacchetti (non è sempre FIFO, a volte si usano complessi metodi per garantire uguale servizio a tutti i vari host comunicanti).
**Datagram**: composto da vari cambi, come version number, header length (per la lunghezza delle opzioni), type of service (real time oppure no), datagram length (lunghezza header + data), identifier, flags, fragmentation offset, time to live (se arriva a 0 il pacchetto viene buttato), protocol (6 per TCP, 17 per UDP), header checksum, source IP, destination IP, options (di varia lunghezza), e alla fine data, cioè il payload. Senza opzioni, l'header del datagram occupa 20 byte. **Fragmentation**: Siccome nel percorso che fa un pacchetto passa attraverso vari e diversi link protocol, potrebbe succedere che uno di essi abbia il MTU (maximum transmission unit) molto basso. Quindi, a volte un datagram è troppo grosso per essere mandato. Viene diviso in due o più datagram più piccoli, e mandato. Vengono riassemblaty all'end system di destinazione. Ogni datagram inviato ha un identifier number, quindi se due pacchetti hanno lo stesso identifier vuol dire che sono parte di uno frammentato. L'ordine dei frammenti viene ripristinato grazie al fragmentation offset. L'ultimo frammento ha una flag posta a 0, mentre tutti gli altri ce l'hanno settata a 1.
**IP addressing**: ogni interfaccia (cioè uscita ad un link) di ogni host e di ogni router deve avere un IP unico e globale. Un network che connette degli host ad un router, o dei router tra di loro, si chiama **subnet**. Secondo la strategia Classless Interdomain Routing (CIDR), un IP ha la forma a.b.c.d\x, dove x indica il numero di bit del prefisso, e questo prefisso indica la subnet della destinazione voluta. In questo modo, i router qunado devono propagare un pacchetto guardano solo il prefisso; il resto dei bit vengono guardati quando si deve decidere in quale host della subnet tocca mandare il pacchetto. L'unico indirizzo speciale è 255.255.255.255, che indica come destinazione tutti gli host della subnet corrente. Si occupa di gestire gli indirizzi IP la Internet Corporation for Assigned Names and Numbers (ICANN)
**DHCP**: Il protocollo Dynamic Host Configuration Protocol serve a comunicare a un host che si connette a un router varie informazioni sulla rete, tra cui il suo indirizzo IP (che può essere fisso o temporaneo), l'indirizzo del DNS locale, la subnet mask etc. E' un protocollo plug and play, che funziona in automatico, fatto apposta per le situazioni in cui ci sono molti utenti che vanno e vengono, in una rete dinamica. Ci sono quattro step: DHCP server discovery (l'host manda un DHCP discover message con un pacchetto UDP alla porta 67, all'indirizzo 255.255.255.255), DHCP server offer (il server DHCP risponde con un DHCP offer message all'indirizzo 255.255.255.255, con dentro un IP valido e l'IP address lease time, cioè il tempo per cui è valido), DHCP request (l'host decide se l'IP ricevuto va bene, poichè potrebbero anche essere più di uno, e risponde all'offerta con un DHCP request message), e infine DHCP ACK (il server risponde alla DHCP request con un DHCP ACK message). Ci sono anche altri modi, per comunicare ad esempio che voglio continuare ad usare il mio IP oltre il lease time consentito.
**NAT**: Netword Address Translation: se una subnet diventa molto più grande del previsto, finiscono presto gli IP disponibili. Allora, si usa una NAT: in pratica tutta la subnet, per internet, prende un'unico indirizzo IP che punta al router. Dentro la subnet, si usano degli indirizzi IP privati, che non hanno significato all'esterno della rete domestica. Ora, il router NAT quando deve mandare un pacchetto all'esterno, usa una porta diversa per ogni porta uscente da ogni host, e si salva quale porta ha usato. Quando riceve un pacchetto dall'esterno, controlla la porta a cui è indirizzato e lo rimanda all'host e alla porta corrispondenti ai dati che si era salvato. Questo porta alcuni vantaggi come l'enorme numero di host che si possono connettere (65,000, come i numeri delle porte), ma anche vari svantaggi, soprattutto il fatto che gli host sono meno visibili nella rete e non possono comunicare attraverso porte prefissate, quindi gli vengono impediti una serie di servizi come connessioni TCP (che devono essere ottenute attraverso NAT traversal) oppure l'impossibilità di funzionare come server. Per aggirare questo problema si può usare **UPnP** (universal plug and play), in cui chiedo al NAT di mappare una precisa porta pubblica per una certa porta di un certo host.
**ICMP**: Internet Control Message Protocol: sono pacchetti speciali portati dentro l'IP payload. In pratica nei layer è un pochino sopra IP. I messaggi ICMP hanno un campo tipo e un campo codice, e contengono l'header e i primi 8 byte del datagram che ha generato il messaggio ICMP. Per esempio Traceroute funziona mandando messaggi UDP con porte sbagliate e aspettando di ricevere messaggi ICMP di errore.
**IPv6**: Principali differenze: gli indirizzi passano da 32 bit a 128 bit; aggiunti gli anycast address, in cui si richiede che un messaggio venga mandato a uno qualunque di un gruppo di host; header fisso a 40 byte. Inoltre, adesso il formato del pacchetto IPv6 diventa fatto dai seguenti campi: version, traffic class (come type of service in IPv4), flow label, payload length, next header (come il protcol field in IPv4), hop limit (se arriva a 0 il pacchetto viene buttato), source address, destination address, e data, cioè il payload. Non ci sta più la frammentazione: se un pacchetto è troppo grosso per essere inviato attaverso un certo link, allora viene rimandato un ICMP message con scritto "Packet too big" ed è proprio il source host a mandare datagram più piccoli. E' stato rimossa la checksum, dato che sia UDP che TCP la fanno per conto loro, e inoltre sarebbe dovuta stata ricalcolata a ogni router, dato che dipende dal campo time to live. Sono state eliminate le opzioni. Vengono trattate come un payload, e se ci sono indicate dal campo next-header.
**IPv4 to IPv6**: Ci sono due modi principali per passare da IPv4 a IPv6 dappertutto: uno è il dual-stack, in cui i router che sono doppiamente compatibili, se vedono che un hop va fatto attraverso un nodo IPv4, semplicemente convertono l'header (perdendo alcune informazioni come il flow), e poi l'IPv6 viene ripristinato come possibile da un nodo IPv6. L'altro è il tunneling, in cui se si deve passare attraverso un nodo IPv4 si incapsula il messaggio IPv6 dentro un IPv4, e viene decapsulato appena possibile da un nodo IPv6. Per capire quali destinazioni accettano IPv6 e quali no, ce lo dice il DNS.
Di solito un host è connesso a un solo router, detto il **default router** ( o anche detto il first-hop router). Allora, questo router, per mandare pacchetti ad altri router, deve usare algoritmi di instradamento. Possono avere varie caratteristiche: possono essere **globali** (calcolo del cammino su tutta la rete, detti algoritmi link-state), **decentralizzati** (calcolo del cammino solo nella mia parte del grafo, poi parlo con i vicini, detti algoritmi distance-vector), **statici** (i cammini cambiano molto raramente), **dinamici** (varia sia il grafo sia il volume del traffico), **sensibili al carico** (i costi dei collegamenti variano dinamicamente in base al traffico), **insensibili** (non cambiano i costi).
**Algoritmo d'instradamento link-state (LS)**: ogni nodo conosce tutto il grafo, questo lo otteniamo facendo scambiare tra i nodi dei pacchetti di stato, che contengono collegamenti e costi vari. Ottenuto tramite un algoritmo di link-state broadcast. In pratica viene usato l'algoritmo di Dijstra. I percorsi vengono ricostruiti in base ai predecessori di ogni nodo. La complessità è $O(n^2)$, se si usa un heap il tempo minimo scende a logaritmico.
**Algoritmo d'instradamento distance-vector (DV)**: è un algoritmo **distribuito**, **iterativo**, **asincrono**. Ogni nodo mantiene le seguenti informazioni: il costo del cammino per ciascun vicino, il vettore distanza, e il vettore distanza di ciascuno dei suoi vicini. Quando un nodo riceve un nuovo vettore distanza da qualcuno dei suoi vicini, lo salva e usa la formula di Bellman-Ford per aggiornare il proprio vettore distanza $D_x(y) = min_v\{c(x,v) + D_v(y)\}$ per ciascun nodo y in N. I vettori distanza vengono scambiati tra i nodi all'inizio e ogni qual volta dovesse cambiare un collegamento. **Reverse poisoning**: se un nodo $z$ deve passare attraverso $y$ per raggiungere $x$, allora nel distance vector che passa a $y$ mentirà dicendo che la sua distanza a $x$ è infinita. In questo modo, $y$ non tenterà mai di passare attraverso $z$ per arrivare a $x$. Questo serve a prevenire **routing loop**, che avviene quando viene aumentato il costo di un collegamento (finchè non vengono riaggiornate le forwarding table, un pacchetto potrebbe rimbalzare all'infinito tra due nodi). Tuttavia, il reverse poisoning riesce a risolvere routing loops in cui sono coinvolti due soli nodi; per quelli che coinvolgono più nodi non c'è soluzione al count to infinity problem.
**Count-to-infinity**: si verifica quando in una rete del tipo A-B-C viene perso il collegamento con A. C, che non è ancora al corrente della situazione, sostiene di poter arrivare ad A in due passi (attraverso B). B, siccome non p più arrivare ad A, allora decide di mandarlo a C credendo che lui ci possa arrivare, e così via.
**LS vs DV routing algorithm**: Inanzitutto, nel LS la complessità dei messaggi passati è enorme, in quanto devo passare informazioni su tutto il grafo ($O(N \cdot E)$ messaggi passati in giro). Tuttavia, nel DV la velocità di convergenza è pessima: mentre nel LS richiede tempo $O(N^2)$ per eseguire Dijkstra, in DV non è garantito che si raggiunga questa convergenza (risultato ottimale), e in caso favorevole ci si arriva molto lentamente. Per quanto riguarda la robustezza, vince LS poichè viene influenzato meno dagli errori (in DV potrebbe passare molto tempo prima che la rete si accorga di un router malfunzionante). Questi due algoritmi sono i più usati, ci sono altri algoritmi di routing del tipo **circuit switched routing** ma poco usati e in casi molto particolari.
**Autonomous systems**: gli algoritmi di routing visti finora funzionano solo per reti piccole e limitate (e soprattuto prive di una disposizione gerarchica); nella realtà i router vengono raggruppati in autonomous system, all'interno dei quali vengono eseguiti questi algoritmi di routing per distribuire i pacchetti. L'algoritmo che gestisce il traffico dentro un AS è chiamato **intra-AS routing protocol** (simile ai due visti prima). Ogni AS possiede alcuni router capaci di comunicare con altri AS, e questi prendono il nome di **gateway routers**. Invece, il protocollo che si occupa di ottenere informazioni dagli AS interconnessi e propagare l'informazione a tutti i router interni all'AS prende il nome di **inter-AS routing protocol**. Affinchè due AS possano comunicare, devono usare lo stesso protocollo. In internet, tutti gli AS usano lo stesso inter-AS protocol, chiamato BGP4.
**Hot potato routing**: Un router dentro un AS, quando deve mandare un pacchetto all'esterno dell'AS (e quindi deve scegliere come destinazione uno dei vari gateway router dell'AS), semplicemente lo manda al gateway router più vicino (cioè quello per il quale il pacchetto impiega meno tempo).
**Routing Information Protocol**: RIP è uno dei primi protocolli intra-AS, ed è molto simile all'algoritmo distance vector. Invece delle distanze usa il numero di hop che il sorgente deve fare per arrivare a destinazione. Gli hop non sono contati attraverso i router, ma attraverso il numero di sottoreti attraversate (destinazoine compresa). Il massimo numero di hop consentito è 15. Ogni 30 secondi, i router si scambiano aggiornamenti sullo stato della rete, chiamati RIP response messages (o bRIP advertisements). Ogni router tiene la sua **routing table**, che contiene sia la forwarding table, che il distance vector. Per ogni destinazione, oltre al numero di hop necessari, nella routing table c'è anche scritto qual è il primo router a cui va mandato un pacchetto indirizzato a una certa destinazione. Se dopo 180 secondi non ho più ricevuto notizie da un router, lo considero irraggiungibile. Importante: RIP in realtà è implementato nell'application layer protocol: i RIP advertisements vengono trasportati attraverso UDP nella porta 520 (quindi dietro ogni router che esegue RIP ci deve essere una macchina UNIX che esegue il processo **routed**).
**OSPF**: considerato il successore di RIP, è simile a un algoritmo Link-State. Ogni router esegue Dijkstra imponendo se stesso come radice. Inoltre, ogni router manda informazioni a tutti gli altri router dell'AS (flooding), invece che solo ai vicini. Manda informazioni ogni qual volta c'è stato un cambiamento in un link, oppure periodicamente ogni 30 minuti. I messaggi OSPF (**OSPF advertisements**) sono trasportati direttamente da IP (quindi il protocollo deve implementare da solo reliable transfer etc). Importante: i costi dei collegamenti in un AS vengono settati a mano dal network administrator (spesso sono inversamente proporzionali alla capacità del link, in modo che un link lento ha un peso molto alto). I vantaggi principali di OSPF sono: Sicurezza (gli advertisements sono autenticabili, attraverso password semplici oppure anche attraverso hash MD5 sia del pacchetto che della password); Multiple Same Cost Paths (se per arrivare a una certa destinazine ci sono vari percorsi di costo uguale, OSPF permette di distribuire i pacchetti attraverso i vari percorsi); Integrated Support for Unicast and Multicast routing; Hierarchy Support (un AS puo' venire diviso in aree, ognuna che usa localmente la sua versione di OSPF, e un'area particolare, la backbone, collega tutte le aree fra loro).
**Border Gateway Protocol** (BGP4): In questo protocollo coppie di router si scambiano informazioni attraverso connessioni TCP semipermanenti sulla porta 179. Questi due router prendono il nome di **BGP peers**, mentre la connessione si chiama **BGP session**. Se la connessione avviene tra due router dello stesso AS, allora è una internal BGP session (iBGP); altrimenti, è una external BGP session (eBGP). Un AS puo' fare l'advertising delle sue subnet attraverso il più lungo prefisso comune. In BGP, ogni AS è identificato da un unico autonomous system number (ASN). Quando un router fa un advertisement BGP, non manda solo i prefissi raggiungibili dal suo AS, ma manda anche una serie di attributi, che insieme ai prefissi prendono il nome di **route**. I più importanti attributi sono: AS-PATH (contiene tutti gli AS attraverso cui l'advertisement è passato. Usato per prevenire looping); NEXT-HOP (interfaccia del router che inizia l'AS-PATH. Usato per configurare correttamente le forwarding table). **Import policy**: quando una particolare route arriva ad un router, il router puo' decidere se scartarla o no, oppure settare certi attributi. Questo poichè spesso un certo router non è autorizzato a mandare messaggi che passano attraverso alcuni particolari AS (che appartengono a compagnie diverse). **Route selection**: se ci sono più route possibili, allora BGP sceglie in base ai seguenti criteri, in ordine: Local Preference (settato dal network administrator), shortest AS-PATH, closest NEXT-HOP (intra-AS hot potato), BGP identifiers.
**Broadcast Routing**: un singolo nodo source vuole mandare un certo pacchetto a tutta la rete. L'approccio più semplice (N-way unicast) consiste nel mandare una copia del pacchetto a ogni destinazione. Non solo è molto inefficiente (intaso molto il primo link che attraverso), ma è anche difficile e richiederebbe protocolli aggiuntivi (come fa il sorgente a conoscere tutte le destinazioni?). Un altro approccio è uncontrolled flooding, in cui quando un router riceve un pacchetto broadcast, manda una copia del pacchetto a tutti i suoi vicini (tranne a quello da cui è arrivato il pacchetto); tuttavia ci sta il problema enorme del **broadcast storm**: non solo verranno creati trilioni di pacchetti, ma se nel grafo c'è un ciclo allora i pacchetti non smetteranno mai di girare. Invece, nel controlled flooding, si usa un approccio chiamato **reverse path forwarding**: quando un router riceve un pacchetto broadcast con un indirizzo di sorgente, trasmette il pacchetto a tutti i vicini se e solo se il pacchetto è arrivato dal link che sta sul percorso più corto dal router in questione alla sorgente. Previene looping e broadcast storm. Un altro approccio un po' più debole è il **sequence number controlled flooding**: il sorgente scrive il suo indirizzo e un broadcast sequence number nel pacchetto, e lo manda a tutti i vicini. Ogni router a questo punto si tiene una lista, per ogni sorgente, dei pacchetti broadcast arrivati, così evita di mandare due volte un pacchetto che è già arrivato in precedenza. Al posto del flooding, ci sta un'altra tecnica molto elegante: lo **Spanning-Tree Broadcast**: nel viene costruito un minimum spanning tree. Ogni nodo manda il pacchetto broadcast solo attraverso i link che fanno parte del minimum spanning tree. In questo modo, ogni nodo riceve il pacchetto broadcast una e una sola volta. Come si costruisce questo spanning tree? **center-based approach**: viene preso un nodoo centrale, e tutti gli altri nodi mandano un pacchetto verso questo nodo centrale. Il pacchetto si ferma quando arriva al nodo centrale, o quando incontra unnodo che sta già sullo spanning tree, e determina su quale ramo dello spanning tree il nodo che ha inviato il pacchetto sta.
**Multicast Routing**: un pacchetto viene mandato da una sorgente a un subset dei nodi della rete. I subset sono gestiti come multicast group, dove ogni gruppo ha un suo preciso indirizzo IP, in modo tale che se voglio mandare un pacchetto a tutto il gruppo devo solo specificare quell'indirizzo, e non tutti gli indirizzi singoli di tutte le destinazioni. **Internet Group Management Protocol**: in pratica fa comunicare un host con il suo first-hop router, nel caso l'host voglia far parte di un certo multicast group. Ha solo tre tipi di messaggi: membership_query (il router chiede agli host su un'interfaccia di quali gruppi fanno parte); membership_report (risposta al membership query da parte dell'host; puo' anche essere usato dall'host quando l'host entra in un group e vuole notificarlo al router); leave_group (è opzionale, in quanto il router capisce se un host non sta più in un gruppo se non risponde più al membership_query dopo un certo timeout). E' un esempio di soft-state (stato che termina dopo un certo timeout). Tuttavia, per mandare messaggi tra i multicast, servono altri algoritmi, i **multicast routing algorithms** (poichè IGMP fa parlare solo l'host con il suo first-hop router). Il punto è che si deve costruire un albero nella rete che collega tutti gli host di un certo multicast group (questo albero potrebbe comprendere anche router di passaggio che non fanno parte del gruppo). Ci sono due approcci fondamentali: Multicast routing con group-shared tree, dove viene costruito un solo albero da un nodo centrale nella stessa maniera con cui viene costruito uno spanning tree (ma in cui rimane il problema di come scegliere il nodo centrale), e Multicast routing con source-based tree, dove ogni sorgente ha un suo albero che costruisce traimite RPF (reverse path forwarding), anche se è necessaria una tecnica chiamata pruning per evitare che nodi che non fanno parte del multicast ricevano troppi pacchetti RPF per la costruzione dell'albero. Tuttavia nell'applicazione reale l'algoritmo più usato è il Protocol Independent Multicast (PIM) routing protocol, in cui fa differenza in base alla vicinanza degli utenti di un certo gruppo. In modalità densa, PIM usa flood-and-prune RPF. In modalità sparse, PIM crea vari rendezvous points per creare l'albero.

# cap 8

**Ciphers**: ci sono due tipi di crittografia: i block ciphers (PGP, SSL, IPSec) e stream ciphers (wireless LAN).
**Block Ciphers**: usati da PGP, SSL, IPSec, etc. Funzionano prendendo il messaggio in blocchi da $k$ bit, e ogni blocco viene criptato indipendentemente. Non si possono usare delle full-table block ciphers, dato che sarebbero grandi $2^k$ (e di solito k è 64), percio' si usano delle funzioni che simulano le permutazioni random (spesso dividono i k bit in sottogruppi dove usano tabelle, e poi permutano quelli). Esempi sono DES e AES. Tuttavia c'è un grave problema: se per caso mi capitasse di mandare più volte un blocco uguale, si capirebbe moltissimo e ci sarebbe un grosso rischio che mi beccano la chiave. Per fare questo, uso il Cipher Block Chaining (CBC): all'inizio mando una sequenza random chiamata Inizializaiton vector, in clear text, che chiamo $c(0)$. Poi, il primo messaggio $m(1)$ lo mando come $c(1) = K(m(1) \oplus c(0))$. Per un futuro messaggio $m(i)$, lo mando come $c(i) = K(m(i) \oplus c(i-1))$, dove $K$ è l'algoritmo usato per criptare (con una data chiave apposita). In pratica uso il messaggio precedente come numero random per fare lo scramble dei messaggi (in realtà, avrei potuto fornire un numero random in cleartext per ogni messaggio mandato; tuttavia avrei dovuto usare quasi il doppio dei bit).
**RSA**: scegli due numeri primi molto grandi $p$ e $q$. Trova $n = pq$ e $z = (p-1)(q-1)$. Scegli $e \mid MCD(e,z) = 1$. Scegli $d \mid (ed-1) \bmod z = 1$. A questo punto, la chiave pubblica è $(n,e)$; la chiave privata è $(n,d)$. Se un altro vuole mandarmi un messaggio $m$, mi manda il messaggio $c = m^e \bmod n$. Quando lo ricevo, per decriptarlo, lo ottengo con $m = c^d \bmod n$.
**Cryptographic hash function**: è una funzione che preso un input $m$ restituisce una fixed-size string $H(m)$. Inoltre, deve essere computazionalmente complicato trovare due messaggi $x$ e $y$ tali che $H(x) = H(y)$. Esempi sono MD5 oppure SHA-1. Questi servono ad assicurare la provenienza di un messaggio. Infatti, poniamo caso che mittente e ricevente si mettano d'accordo su una authentication key $s$. Il mittente, mandando un messaggio $m$, manda anche $H(m+s)$, che prende il nome di **message authentication code** (MAC). Il ricevente, per essere sicuro della provenienza, calcola $H(m+s)$ e controlla che corrisponde.

**Digital Signature**: Supponiamo che voglio firmare digitalmente un documento $m$. Allora, diffondo anche la versione criptata con la mia chiave privata, cioè $K^-(m)$. Per chi volesse verificare che sono stato proprio io a firmare il documento, gli basta decriptare con la chiave pubblica e verificare che $K^+(K^-(m)) = m$. L'unico problema è che queste operazioni sono lunghe e dispendiose; infatti di solito, invece che diffondere tutto il messaggio criptato, diffondo solo un hash criptato del messaggio. Allora al ricevente basta verificare che l'hash del messaggio originale $m$ corrisponda alla decriptazione dell'hash criptato che ho diffuso. Uno dei problemi che potrebbe però sorgere è che non si è sicuri a chi appartenga una certa chiave pubblica. Per questoa esiste un'ente chiamato Certification Authority, che verifica e attesta che una certa chiave pubblica appartiene a una certa persona/compagnia., producendo un certificato (ovviamente firmato digitalmente)
**Authentication Protocol**: ap4.0: Mi voglio autenticare a un server. Dichiaro la mia identità. Il server mi manda un nonce(per evitare playback attacks), cioè un numero unico per me (o comunque non usato per molto tempo). Io rispondo rimandando questo numero criptato con una password segreta che conosciamo solo io e il server. A questo punto ho completato l'autenticazione. ap5.0: al posto di una semplice password, viene usata chiave pubblica e chiave privata. Quando ricevo la nouce, la cripto con la mia chiave privata. Poi il server mi chiede la mia chiave pubblica. Io gliela mando, e il server dovrebbe riotttenere la nounce originaria. Ancora più sicuro di prima; tuttavia, ci potrebbe comunque essere il problema di man in the middle attack.
**Secure e-mail**: per avere la massima confidentiality, potremmo semplicemente usare RSA (invece che metterci d'accordo su una shared key in AES), ma è inefficiente per messaggi lunghi. Allora si fa così: cripto il mio messaggio in AES con una symmetric key, e mando il messaggio. Dopo, mando la symmetric key criptata in RSA con la sua chiave pubblica. Il ricevente si deve prima ricavare la symmetric key con la sua chiave privata, e poi può decriptare il messaggio originario. Per verificare anche l'autenticità del mittente (cioè voglio esssere sicuro che chi me l'ha mandato sia corretto) e anche l'integrità del messaggio, combino al messaggio un hash del messaggio firmato digitalmente, e ripeto la procedura della simmetric key.
**Pretty Good Privacy**: usa MD5 oppure HASH per l'hash; usa CAST, 3-DES o IDEA per la symmetric key encryption, RSA per la public key encryption. Quando PGP viene installato, viene creata una coppia di chiavi. Per accedere alla chiave privata, serve una password. Tuttavia, per la certificazione delle chiavi pubbliche, PGP non usa CA, ma usa una rete di fiducia (web of trust): cioè gli utenti possono firmarsi a vicenda le chiavi (possibilmente incontrandosi fisicamente).
**Secure Sockets Layer** (SSL, TLS): Fino ad ora abbiamo visto come mettere in sicurezza il livello applicativo. Ora vediamo il livello di trasporto (TCP). Tutte le transazioni elettroniche o quantomeno importanti avvengono attraverso SSL (si nota dal fatto che l'url inizia con "https://"). SSL offre i seguenti servizi: confidentiality, data integrity, server autentication, client autentication. Tecnicamente, SSL risiede nell'application layer; ma per lo sviluppatore è buono considerarlo parte del transport layer. Un messaggio SSL è formato da 3 campi in chiaro: Type (handshake oppure data), Version, Length; e da 2 campi criptati (Data e MAC). Prima di comunicare, client e server devono superare la fase handshake, che serve ad autenticare tutti e due e a scambiarsi la master key (da dove saranno ricavate le varie symmetric key); funziona in questo modo: il client manda al server una lista di algoritmi crittografici supportati; il server ne sceglie uno simmetrico (come AES), uno public key (RSA), e un MAC algorithm, manda le scelte al client insieme a un certificato della sua public key e un server nonce (che serve a evitare playback attacks); il client verifica il certificato, si salva la public key del server, genera una Pre-Master-Secret (PMS), la cripta con la chiave pubblica del server e gliela manda; usando la stessa funzione di derivazione definita da SSL, client e server estraggono dal PMS e dal nonce la Master-Secret (MS), e da questa MS ricavano quattro chiavi: due di criptazione (una da client a server, e una da server a client) e due di MAC (sempre una da client a server e una da server a client); infine client e server si scambiano un MAC di tutti gli handshake messages (per evitare che qualcuno ha combinato qualcosa durante l'handshake). Importante: siccome SSL deve mandare il MAC di ogni messaggio, lo stream di byte viene diviso in record, e ogni record ha il suo MAC. Importante: in SSL, nella creazione del MAC, non vengono usati solo il messaggio e la chiave, ma viene usato anche il sequence number, un numero che l'host aumenta ogni volta che manda un record (questo serve a evitare che un terzo possa scambiare l'ordine dei pacchetti, dato che il sequence number del TCP è in chiaro); il sequence number dell' SSL non viene scambiato, ma viene usato per calcolare il MAC: in questo modo il ricevente di un messaggio può verificare che nessuno ha manomesso l'ordine dei record. Importante: per chiudere una connessione, lo si indica nel campo Type dell'SSL Record, non si usa il TCP FIN (in quanto un terzo potrebbe chiudere la connessione al posto nostro, truncation attack).
**Virtual Private Network** (VPN): una compagnia potrebbe volere i propri dati sicuri e costruirsi un proprio private network, ma molto spesso è troppo costoso, e quindi si ricorre a una VPN. Funziona in questo modo: quando due host si scambiano un pacchetto che passa solo per gli headquarters della compagnia, viene mandato attraverso il normale IPv4. Tuttavia, se deve passare attraverso internet, l'header del pacchetto è sempre IPv4 (in questo modo, i router lo possono trattare anche come un normale datagram), mentre nel payload ci sta un pacchetto IPSec, con il suo header in chiaro ma il payload criptato.
**IPSec**: ci sono due protocolli principali: Authentication Header (AH) e Encapsulation Security Payload (ESP). Sono simili in quanto tutti e due offrono source authentication e data integrity ma solo il secondo offre confidentiality (e infatti parleremo solo del secondo). Prima di scambiarsi pacchetti IPSec il mittente e il ricevente devono creare una connessione logica chiamata security association (SA), che è unidirezionale. Ogni host deve mantenere delle informazioni riguardo una SA: un 32-bit identifier chiamato Security Parameter Index (SPI); l'interfaccia di origine e di destinazione; il tipo di crittografia; la chiave di crittografia; il tipo di integrity check (HMAC con MD5 per esempio); la chiave di autenticazione. Un host si salva tutte le informazioni riguardo le sue SA nel Security Association Database (SAD). IPSec ha due forme differenti di pacchetto: una per il tunnel-mode e una per il transport-mode. La prima è molto più usata della seconda. Il datagram IPSec è formato in questo modo: si mette un ESP trailer (che contiene i campi padding, per far sì che la lughezza del messaggio sia un multiplo del block length, pad length, e next header, che indica il tipo di dato trasportato) alla fine del datagram IPv4 e si cripta tutto quanto con la chiave di crittografia. Poi si appende all'inizio l'ESP header (che contiene SPI per identificare a quale SA appartiene, e sequence number, per difendersi dai replay attacks), formando così l'enchilada. Si appende all'enchilada l'ESP MAC, ottenuto dall'algoritmo di autenticazione specificato dal SA applicato a tutta l'enchilada. Finalmente, si incapsula tutto quanto in un IPv4 datagram con i classici header. I nuovi header dichiareranno però che l'IP sorgente e di destinazioni non sono più quelli originari (che sono stati criptati) ma sono quelli di inizio e di fine del tunnel. Inoltre, nel campo protocol number viene scritto 50, a significare che è un IPSec datagram che usa il protocollo ESP. Ogni host mantiene pure una Security Policy Database, che indica quali tipi di datagram devono essere processati con IPSec, e per quelli che devono essere processati, quale SA deve essere usato; ma indica anche in quale caso un datagram in chiaro deve essere criptato prima di essere inoltrato nella rete. In sostanza, IPSec fornisce i seguenti servizi: confidentiality, source authentication, data integrity, replay-attack prevention. Se una VPN è molto piccola, le informazioni sulle SA le può mettere anche il network administrator. Tuttavia, se diventa troppo grande, è necessario un sistema automatizzato, che viene fornito dal Internet Key Exchange (IKE) protocol. Il suo funzionamento è molto simile a quello dello SSL handshake, solo che è diviso in due fasi: nella prima, i due host creano con Diffie-Hellman una IKE SA bidirezionale (differente dall'IPSec SA in quanto crea un canale autenticato e criptato per i due router), dove nel loro primo messaggio si scambiano chiavi per la criptazione e l'autenticazione dell'IKE SA (e si mandano anche la master secret); nella seconda fase, i due router si autenticano fra di loro firmando digitalmente, ma nessun altro sa le loro identità dato che le autenticazioni sono criptate, e si mettono d'accordo sugli algoritmi vari da usare.
**Firewall**: è una combinazione di hardware e software tale da isolare la rete interna di un'organizzazione dall'Internet intero, lasciando passare solo alcuni pacchetti. Ci sono tre tipi di firewall: **Traditional Packet Filters**: tutto il traffico che entra o che esce passa attraverso un router, e in questo router vengono filtrati i pacchetti, e ogni datagram viene esaminato in base a una serie di regole definite dal network administrator (IP source e destination, tipo di protocollo, TCP o UDP source e destination port, TCP SYN o ACKS, ICMP messages etc.). Queste regole sono implementate nei router attraverso access control lists, e ogni interfaccia di un router ha la sua list. Invece, gli **Stateful Packet Filters** saguono le connessioni TCP, ricordandosi di tutti i segnali SYN, ACK, e FIN, e tenendosi tutta la lista delle connessioni in una connection table; inoltre, nella access control list, ci sta una nuova colonna "check connection". L'ultima forma è l'**Application Gateway**: cioè un application specific server che va a controllare molto più nel dettaglio: non controlla l'header di un datagram, ma controlla l'application data. Ogni host può eseguire varie application gateways, ognuna con il suo processo server. Alcuni svantaggi sono che tutto va più lento (dato che tutti i dati devono passare via il gateway), e il software deve essere configurato singolarmente per istruirlo a contattare il gateway e a passargli tutti i dati.
**Intrusion Detection Systems** (IDS): è un dispositivo che avvisa il network administrator quando ossera pacchetti potenzialmente pericolosi (anche a livello di applicatoin data). Se invece li blocca proprio questi pacchetti pericolosi, allora si parla di Intrusion Prevention System (IPS). Una organizzazione può settare vari sensori IDS, ognuno con funzionalità diverse, che si dividono il lavoro, che inviano i loro dati a un processore IDS centrale, che colleziona tutto e manda gli avvisi. Si classificano o come signature-based systems oppure come anomaly-based systems. I primi si mantengono un largo database di attack signatures, cioè di caratteristiche che corrispondono ai pacchetti pericolosi, uno dei più famosi è Snort (solo che il problema principale è che possono sgamare solo attacchi già più o meno conosciuti, e non sono velocissimi) . I secondi, invece, controllano anomalie nelle statistiche del traffico (per esempio, strani port scans, oppure troppi pacchetti ICMP uno dietro l'altro). Loro possono sgamare anche attacchi nuovi e non documentati, ma sono molto difficili da calibrare.

## Altre cose

**ARP spoofing**: consiste nel mandare falsi pacchetti di risposta ARP in una LAN col fine di alterare le tabelle ARP degli host e ridirezionare i pacchetti nella rete, in modo da leggerne il contenuto. Alcune tecniche efficaci per contrastare questo tipo di attacco sono l'utilizzo di IPv6, IPSec, tabelle ARP statiche, insomma qualunque cosa che rende sicuro lo scambio di messaggi ARP (anche Snort).
**Bluetooth**: wireless technology in 2.4GHz, introdotto originariamente per sostituire i cavi. 1 MHz spaced channels, quindi velocità a 1 Mb/s. Usa il Frequency Hopping Spread Spectrum, dove la frequenza usata per trasmettere cambia a ogni pacchetto, in modo tale da ridurre interferenze e aggiungere sicurezza. Il tempo viene diviso in slots, e uno slot dura 625 nanosecondi. I pacchetti vengono inviati in 1, 3 oppure 5 slots. Il raggio di comunicazione è molto corto (10 - 100 metri). I dispositivi bluetooth sono organizzati in piconets, cluster fatti da un master e fino a 7 slaves, sincronizzati in base al master ID e al clock. Nel caso una piconet abbia più di 7 slave, alcuni vengono messi in low power mode (detta park mode). Una scatternet invece è un insieme di piconet, che abiliita alla multi-hop communication, e ad-hoc networks di dispositivi bluetooth; tuttavia sono molto complicate da formare, in quanto la device discovery è complicata.
Nel protocollo network è diviso in due parti, la prima è la data plane. (forwarding da un router all'altro) la seconda è il control plane (routing algorithms). Ci sono due modi di fare il control plane: una è di far calcolare i percorsi ai singoli router (come link state oppure DV), un altro modo è usare SDN, sistema centralizzato per calcolare percorsi, che poi manda le forwarding table ai router.
**802.11 advanced**: Rate adaptation: la base station cambia dinamicamente la trasmission rate per abbassare la Signal to Noise Ratio per ridurre il Bit Error Rate (se necessario); Power management: il nodo dice all'ap che si metterà in sleep fino all'arrivo del prossimo beacon frame (che di solito sono a distanza di 100ms). Il beacon frame contiene la lista dei frame che devono essere inviati dall'AP al nodo; se questa lista è vuota, allora il nodo può addormentarsi. Se un nodo va in sleep fino al prossimo beacon sleep, consuma circa l'1% di quanto consumerebbe normalmente.
**Laboratorio**: SDN è software defined networking. Le forwarding table date ai router non sono quelle normali ma si chiamano flow table e si possono usare per fare per esempio firewall, load balancing etc. L'idea su cui si basano è match-plus-action per cui per ogni pacchetto si vede se soddisfa determinati requisiti, e la parte action che dice cosa fare con i pacchetti matchati (forwarding, dropping). Lo standard per l'SDN è OpenFlow. I problemi dell'SDN sono single point of failure, security, latency to controller, scalability, old hardware, distributed system challenges (imperfect knowledge of network state e consistency issues between controllers), e infine la sicurezza . I controller è un programma software che manda e riceve OpenFlow dai network devices, riesce ad ottenere un livello di network convergence che prima non potevamo neanche immaginare. OpenFlow è uno standard per i protocolli di comunicazioni che abilita il control plane a interagire con il forwarding plane (e non è l'unico protocollo di comunicazioni del genere). La differenza tra SDN e il Network Functions Virtualizatoin (NFV) è che NFV si incentra a ottimizzare i singoli servizi network (DNS, caching...) staccandoli da proprietary hardware, in modo tale che esse possano essere eseguite in software per accelerare l'innovazione. SDN e NFV possono essere usate insieme, sono mutually beneficial. Invece, Network Virtualization (NV), si assicura che il network possa integrarsi con e supportare le richieste di architetture virtualizzate.

# CAP 5

I servizi che il link layer può offrire sono: Framing (incapsulamento del datagram in un frame), Link Access (un medium access protocol - MAC - specifica le regole con cui un frame viene trasmesso in un link), Reliable Delivery (garanzia del corretto trasporto dei bit; tuttavia, spesso i cavi hanno error rate talmente basso che non c'è bisogno di questa garanzia), Error detection e Error correction.
Il link layer è implementato spesso in un **network adapter**, anche chiamato **network interface card (NIC)**. Quindi la gran parte dei servizi del link layer sono implementati a livello hardware.
**Parity Checks**: la più semplice forma di parity check è il parity bit (conto il numero di uni e scrivo se è pari o no), che ha una probabilità di beccare l'errore attorno al 50%. Se invece uso il **two-dimensional parity**, divido il messaggio in righe e colonne, e per ogni colonna uso il parity bit. Se ci sta un solo errore, lo posso anche correggere, mentre due errori li posso solo detectare.
**Checksumming**: si fa il complemento a 1 della somma dei byte e si ottiene il checksum. Il ricevente somma tutti i byte al checksum e dovrebbe ottenere un numero con tutti 1. Tuttavia, è una forma abbastanza debole di error checking, infatti viene usata solo a livello software da TCP e cose varie. Il link layer puo' implementare a livello hardware cose come il **Cyclic Redundancy Check** (CRC): sia il sender che il receiver si accordano su $r+1$ bit chiamati il generatore, dove il primo bit è sempre 1. Quando il sender vuole mandare $d$ bits, prende altri $r$ bits in modo tale che il numero formato appendendo $R$ a $D$ sia divisibile per il generatore in aritmetica modulo 2. Il ricevente deve solo controllare che il nuero ricevuto (formato da $d+r$ bits) sia divisibile per il generatore. CRC è capace di detectare tutti gli errori che coinvolgono fino a $r$ bits; inoltre, c'è buona probabilità di trovare errori anche più lunghi.
Ci sono due tipi di link: i **point-to-point link** (una singola sorgente, una sola destinazione), e i **broadcast link** (molte sorgenti e destinazioni connesse allo stesso broadcast channel, come Wi-Fi o Ethernet). Questi ultimi si chiamano broadcast perchè ogni pacchetto viene inviato a tutti i nodi.
**Channel Partitioning Protocols**: TDM (time-division multiplexing), in pratica vengono assegnati N time slots, uno per ogni host, e ogni host puo' inviare pacchetti nel suo slot, molto fair (tutti gli host ottengono bitrate di R/N), ma se ci sta un solo host che vuole parlare è molto svantaggiato; FDM (frequency-division multiplexing), il canale viene diviso in piccole frequenze, in modo che ogni host ha R/N bitrate, ma stessi vantaggi e svantaggi di TDM; CDMA (code division multiple access), ogni nodo ha un codice con cui cripta i propri pacchetti.
**Random Access Protocols**: Tutti i node mandano alla massima bitrate. Se due pacchetti collidono, allora i due nodi coinvolti li riinviano aspettando un delay random. **Slotted Aloha**: Tutti i frame sono composti da L bits. Il tempo è diviso in slots da L/R secondi (cioè giusto il tempo per trasmettere un frame). I nodi iniziano a trasmettere frame solo all'inizio di uno slot.. Se due pacchetti di due nodi collidono, i due nodi se ne accorgono prima che lo slot finisca. Ogni nodo coinvolto in una collisione manda il pacchetto incriminato nel prossimo slot con probabilità $p$. E così via per tanti slot, fino a che non lo manda senza collisioni. Il vantaggio principale è che è un protocollo molto semplice e veloce (quando ci sono pochi nodi). Tuttavia, se i nodi sono tanti, l'efficienza si riduce drasticamente, in quanto uno slot è buono solo se ci scrive un nodo solo (quindi probabilità di uno slot buono $= Np(1-p)^{N-1}$). Si dimostra che l'efficienza massima è del 37%. **Aloha**: Come prima, solo che non ci sono gli slot. In pratica mando un frame appena posso. Se ho una collisione, con probabilità $p$ lo mando, altrimenti aspetto un frame transmission time e riprovo con probabilità $p$. Siccome per una trasmissione per essere buona devono aver finito tutti di trasmettere (con prob $(1-p)^{N-1}$), e nessuno deve iniziare a trasmettere durante la trasmissione (con prob $(1-p)^{N-1}$), allora la probabilità di una trasmissione buona è $p(1-p)^{2(N-1)}$, e l'efficienza è esattamente la metà di slotted aloha. **CSMA**: (Carrier Sense Multiple Access): un nodo trasmette se e solo se, ascoltando il canale, nessun altro sta trasmettendo. Tuttavia si incorre lo stesso in collisioni per il channel propagation delay (il temp o che ci mette un segnale per arrivare da un nodo all'altro). **Collision Detection**: Se un nodo CSMA, mentre trasmette, si accorge di una collisione (cioè riceve un segnale da un altro nodo che sta trasmettendo), allora smette di trasmettere, e aspetta un intervallo random per rimandare tutto il frame. Per scegliere l'intervallo si usa il binary exponential backoff (se un particolare frame ha colliso $n$ volte, allora deve aspettare un intervallo preso a caso tra $\{0 \dots 2^n - 1\}$, moltiplicato per un valore che per esempio in Ethernet è 512 bit times). L'efficienza del Collision Detection è data dalla formula $1/(1+5(d/t))$, dove $d$ è il propagation delay e $t$ è il tempo che ci vuole a mandare un frame.
**Taking Turns Protocol**: Sia ALOHA che CSMA hanno la proprietà che se ci sta un solo nodo nel canale, quello raggiunge una velocità di R bits, ma non garantiscono che se ci sono tanti nodi la velocità media sarà di R/N bits. Vediamo se alcuni protocolli ha possono garantire: **Polling Protocol**: un nodo diventa il master node, che dice a ogni altro nodo, a turno, che ha la possibilità di mandare al massimo un certo numero di frame. Quando quel nodo ha finito (e lo si capisce dall'assenza di segnale), il master da il permesso a un altro nodo, e così via. Svantaggi: polling delay (il tempo che si deve aspettare perchè il master dia i permessi, e quindi se c'è un solo nodo attivo non riesce ad ottenere R bits di velocità), e il problema che se per qualche motivo il master fallisce la rete non funziona più. **Token-Passing Protocol**: ci sta un frame particolare, il token, che serve a decidere chi parla. Quando un nodo ha finito di trasmettere, passa il token al prossimo nodo. Ci sta un numero massimo di frame che un nodo puo' mandare prima di passare il token. Se un nodo fallisce, cade la rete. Se un nodo si rifiuta di passare il token, si perde il token e ci stanno delle procedure per rimetterlo in circolazione.
**Indirizzi link-layer**: un indirizzo link-layer è anche chiamato un LAN address, un Physical address o un MAC address. Ogni adapter (network interface) di un host o un router ha un suo MAC address (proprio come avrebbe un suo IP address). Tuttavia, i link switches non hanno un loro address. Per la maggior parte delle LAN il MAC address è lungo 6 byte, spesso espresso in esadecimale. Inoltre, ogni adapter ha un unico MAC address (responsabilità della IEEE e dei manufattori di adapters. Nella maggior parte dei casi, un indirizzo MAC è fisso, non cambia in base al luogo. Spesso un frame viene mandato in broadcast, e ogni adpater per capire se è destinato a lui oppure no confronta il MAC address del frame con il suo proprio MAC address. L'indirizzo usato per mandare un frame a tutti quanti nella LAN è FF-FF-FF-FF-FF-FF. Il MAC address è stato introdotto per rendere il protocollo layer indipendente dal protocollo network (e rendere più veloci le cose) **Address Resolution Protocol** (ARP) : Serve a ottenere il MAC address di un'interfaccia sapendo il suo IP. Simile al DNS, ma funziona solo localmente in una subnet al posto che in tutto il mondo. In pratica ogni router e host ha una sua ARP table, dove a ogni IP corrisponde un MAC address e un TTL (time to live, di solito 20 minuti). Se devo mandare un pacchetto a un'indirizzo noto, bene; altrimenti, mando nella subnet un ARP packet, un pacchetto speciale che viene mandato in broadcast, in cui chiedo chi ha questo particolare IP. L'host che ha quel particolare IP risponde con un altro ARP packet (questa volta non in broadcast, ma indirizzato proprio a me), e io aggiorno la mia ARP table. Il protocollo ARP funziona tutto in automatico, non ha bisogno di manutenzione da parte del network administrator. Se voglio mandare un frame fuori dalla mia subnet, allora il MAC address che dovro' usare è quello del mio first-hop-router, e poi lui penserà a fare il resto del lavoro per me.
**Ethernet**: All'inizio di ethernet, tutti gli host e router sono connessi a un hub con twisted-pair copper wire, usando CSMA/CD come multiple access protocol. In realtà, è un protocollo sia link che fisico. Un hub è un dispositivo che appena riceve un bit da un interfaccia, lo ritrasmette con più potenza a tutte le altre interfacce (quindi fa da broadcast LAN, tutti ricevono tutti i pacchetti). Un frame ethernet è composto da: Data field (da 46 a 1500 byte, se va oltre deve essere frammentato da IP, altrimenti deve essere riempito con roba inutile), Source address e Destination address (6+6 byte), Type Field (2 byte, per capire il tipo di network layer protocol da usare, come per esempio IP oppure anche ARP, anche se non sarebbe proprio un network protocol), Cyclic Redundancy Check (4 byte), Preamble (8 byte, i primi 7 sono 10101010, mentre l'ultimo è 10101011, per sincronizzare gli orologi tra sender e receiver). Ethernet è connectionless, e non offre reliable transport. Nei primi anni 2000 gli hub sono stati sostituiti con i **switch**. I switch sono trasparenti agli host, nel senso che non vengono percepiti nella rete, ma fanno il loro lavoro in silenzio. Ogni switch ha la sua switch table, dove a ogni MAC address corrisponde una certa interfaccia e l'istante in cui quella tupla è stata inserita nella tabella. Non è detto che la tabella contenga tutti gli indirizzi della sottorete. Grazie alla tabella un switch fa il filtering (drop dei pacchetti non necessari) e il forwarding (mandare i pacchetti all'interfaccia giusta). Quando arriva un pacchetto al switch, ci sono tre casi: il switch non conosce l'indirizzo MAC di destinazione (in questo caso manda il pacchetto a tutte le interfacce tranne quella da cui è provenuto il pacchetto); il pacchetto arriva dalla stessa interfaccia attraverso cui il switch avrebbe dovuto fare il forwarding (in questo caso il switch droppa il pacchetto); il pacchetto arriva da un'interfaccia diversa (in questo caso il switch fa il forwarding del pacchetto sull'interfaccia giusta presa dalla switch table). Una caratteristica importante dei switch è che non devono essere configurati in nessun modo, in quanto la switch table si costruisce da sola: all'inizio è vuota, e si riempie salvandosi da quale interfaccia è arrivato un pacchetto con un certo MAC address; un'entrata nella tabella viene cancellata se passa troppo tempo dall'ultimo pacchetto arrivato. Quindi, i switch sono plug-and-play, in quanto funzionano interamente da soli. Inoltre, sono full-duplex, in quanto ogni interfaccia puo' ricevere e inviare allo stesso tempo. Importante: ogni switch ha un buffer per salvarsi i frame da mandare, proprio come i router. I vantaggi dei switch: nessuna collisione (i switch si salvano i frame nei buffer e mandano un solo frame alla volta su un'interfaccia, quindi non ci sarebbe neanche bisogno di un multiple access control protocol), link eterogenei (si possono collegare tecnologie diverse, non è necessaria una unica tecnologia di broadcast), più sicuri (se si rompe un cavo si rompe solo quella particolare connessione invece che tutta la subnet, e inoltre eventuali ospiti indesiderati ricevono molti meno pacchetti in broadcast). In fin dei conti, in molti casi un switch potrebbe sostituire un router, in quanto è plug-and-play ed è più veloce a fare forwarding; tuttavia, un router puo' calcolare i percorsi più corti per i pacchetti ed è invulnerabile ad attacchi di broadcast storm. Ci sono un altro tipo di dispositivi usati da Ethernet: i **bridges**, che sono come dei switch ma con due porte sole (o coswitchesmunque molto poche): hanno principalmente la funzione di filtering (cioè non fanno passare un pacchetto il cui MAC address non sta nella zona in cui sta andando), e quindi riescono a dividere la subnet in collision domains; usano bridge tables, costruite e usate nello stesso modo delle switch tables (self-learning). Tuttavia funzionano bene solo quando non ci sono dei cicli nel grafo, quindi devono essere organizzati secondo uno spanning tree, ma questo comporta anche uno svantaggio in quanto non si possono usare path alternativi per ravvelocizzare lo scambio di pacchetti. Sono molto veloci ma non offrono protezione da broadcast storms. Per il resto tutte le altre caratteristiche sono uguali ai switch.

# CAP 6

**Wireless Things**: gli wireless hosts si connettono attraverso wireless links alle base stations. Se sono in infrastructure mode, allora tutti i servizi di network comuni (come DNS, routing...) vengono formiti dalla base station; altrimenti si dice che sono ad hoc networks. Quando un host mobile cambia base station, fa un processo chiamato handoff. Sostituendo un link fisico con uno wireless, ottengo le seguenti modifiche: decrescita della potenza del segnale, interferenza da sorgenti estranee, multipath propagation (parti delle onde elettromagnetiche rimbalzano su alcune superfici prendendo percorsi diversi e arrivando in tempi diversi; peggio ancora se nell'ambiente ci sono oggetti che si muovono). Per misurare la qualità di un segnale wireless si usa la Signal-To-Noise-Ratio (SNR), misurata in dB, che sostanzialmente venti volte il log-10 del rapporto fra l'ampiezza del segnale e l'ampiezza del noise. Dato un certo SNR, aumentare la velocità di trasmissione di bit aumenta il Bit-Error-Rate (BER), la probabilità che un bit venga sbagliato. Per questo spesso il segnale viene modulato e la velocità di trasmissione è dinamica, a differenza dei link wired. Due problemi dei collegamenti wireless sono: hidden terminal problem (un terminale riceve due segnali broadcast da due terminali solo che uno dei due è nascosto all'altro da un ostacolo e non vengono rilevate collisioni) e il fading (un terminale riceve due broadcast da due terminali diversi che sono pero' troppo lontani per accorgersi della loro collisione)
**Code Division Multiple Access** (CDMA): appartiene alla famiglia dei channel partitioning protocols. In pratica ogni bit mandato viene codificato con un codice che cambia molto più velocemente del bitrate (a una velocità chiamata chipping rate). Un bit viene mandato in un certo intervallo di tempo chiamato bit slot, e questo viene diviso in $M$ mini bit slots. Il codice è lungo $M$, e si ripete a ogni bit. Il ricevente, per riottenere la sequenza di bit originaria, rimoltipica tutti i mini bit slots ricevuti per il codice, li somma tutti e li divide per $M$; facendo questo per ogni bit slot dovrebbe ottenere 0 o 1. Incredibilmente questa cosa funziona anche se ci sono vari mittenti e si sommano i messaggi trasmessi. Infatti, se i codici vengono scelti bene (ovviamente diversi per ogni mittente), il ricevente puo' fare lo stesso processo e riottenere le varie sequenze originarie di bit usando i vari codici.
**Carrier Sense Multiple Access with Collision Avoidance** (CSMA/CA) : appartiene alla famiglia dei random access protocols. Come in Ethernet, ogni station sta a sentire il canale prima di trasmettere. Non si usa la collision detection (smetto di trasmettere se capisco che qualcun altro sta trasmettendo), poichè è molto complicato capire se un altro sta trasmettendo (per la scarsa potenza del segnale, e per problemi come hidden terminal problem e fading); infatti qui, quando inizio a mandare un frame, non interrompo più, ma lo invio per intero. Funziona così: all'inzio, se il canale è idle (nessuno sta parlando) aspetto un intervallo di tempo chiamato Distributed Inter-frame Space; altrimenti, scelgo un random backoff value usando la binary exponential backoff (come in Ethernet) e faccio decrescere questo backoff solo mentre il canale è idle; quando il backoff arriva a zero, mando la mia frame e aspetto il mio ack (tutte le station, quando ricevono un frame che supera il controllo CRC, aspettano un piccolo intervallo chiamato Short Inter-frame Spacing e mandano indietro un ack frame); appena arriva l'ack, ricomincio dal punto 2 (scelgo un backoff etc.); se l'ack non arriva proprio, ritorno al punto 2 aumentando l'intervallo di scelta del backoff. La grande differenza dal CSMA/CD è che il backoff time decresce solo quando il canale è idle, invece che continuamente. Questo accade perchè come è detto prima è molto difficile accorgersi delle collisioni, e quindi è molto meglio provare direttamente a evitarle.
**IEEE 802.11** wireless LAN (WiFi): ci sono tre versioni principali di 802.11 (a, b, g; la prima a 11 Mbps, le altre due a 54 Mbps). Inoltre, una nuova, la 802.11n, usa varie antenne sia sul mittente che sul ricevente, per mandare segnali contemporaneamente. Una basic service set (BSS) contiene una o più wireless stations e una base station centrale, chiamata access point (AP). Prima di cominciare a trasmettere dati, una wireless station si deve connettere ad un AP. Un AP ha un Service Set Identified (SSID), un nome formato da una o due parole, e un channel number (siccome 802.11 funziona nella banda 2.4 - 2.485 GHz, ci sono 85 MHz su cui scegliere un canale; 802.11 definisce 11 canali parzialmente sovrapposti; due canali non si sovrappongono se sono separati da almeno quattro canali. Ogni AP, inoltre, manda periodicamente un beacon frame, che contiene lo SSID e il MAC address dell'AP. Per capire quali AP sono accessibili in un certo posto, una wireless station fa lo scan degli 11 canali aspettando questi benedetti beacon frames con le dita incrociate (attenzione: non è detto che due AP diversi trasmettano su due canali diversi), e questo processo si chiama passive scanning. Invece, una wireless station puo' anche mandare un probe broadcast a cui tutti gli AP rispondono; questo invece prende il nome di active scanning. Dopo aver scelto un AP a cui collegarsi, mando una association request, a cui l'AP risponde con una association response. Una volta associato, mando un DHCP discovery message per ottenere il mio IP e per collegarmi finalmente alla subnet corrispondente. Se necessaria autenticazione, questa avviene attraverso o una whitelist di MAC addresses, oppure attraverso username/password (a volte il processo di autenticazione è decentralizzato a un authentication server che funziona per vari AP). Una WiFi jungle è un posto fisico dove una wireless station riceve segnale abbastanza potente da almeno due AP.
**802.11 MAC protocol** Una volta che la wireless station è associata con un AP, può iniziare a mandare e ricevere i frame a e da l'accesso point. Siccome diverse stazioni potrebbero voler trasmettere dati simultaneamente, un protocollo ad accesso multiplo è necessario per gestire il traffico. (Una stazione può essere o una stazione wireless o un AP). Trasportati dal successo di Ethernet e del suo random access protocol, gli sviluppatori dell'802.11 optarono nuovamente per un random access protocol denominato CSMA with collision avoidance, o più brevemente CSMA/CA. In quest'ultimo, similarmente a CSMA/CD per Ethernet, ogni stazione ascolta il canale prima di trasmettere e si astiene dal farlo se rileva che il canale è occupato. Di per sè il protocollo MAC 802.11 non implementa alcuna rilevazione di collisioni. Ci sono due importanti ragioni per questo: in primis perchè la possibilità di rilevare collisioni richiede la capacità d'inviare e ricevere contemporaneamente. Siccome la potenza del segnale ricevuto è nettamente inferiore alla potenza del segnale trasmesso, risulta molto costoso costruire hardware che rilevano la collisione. In secundis è molto importante il fatto che l'adattatore, anche nel caso in cui potesse trasmettere e ricevere simultanemante, non potrebbe rilevare tutte le collisioni a causa del problema del termianle nascostio de dell'attenuazione del segnale. Prima di considerare la collision avoidance, abbiamo bisdogno di esaminare il link-layer acknoledgemenet scheme necessario per ovviare al problema della dispersione dei frame nella rete wireless. Difatti quando la stazione di destinazione riceve un frame che passa il controllo CRC, attende per un breve periodo di tempo, noto come Short inter-frame Spacing (SIFS), dopo il quale invia al mittente un frame di conferma di avvenuta ricezione. Se la stazione trasmittente non riceverà questo riscontro entro un arco di tempo stabilito, presupporrà un errore e ritrasmetterà il frame, utilizzando ancora il protocollo CSMA/CA. Se il frame di acknoledgement non viene ricevuto dopo alcuni tentativi di ritrasmissione, la stazione che sta trasmettendo il frame abbandona e lo scarta. Supponiamo di avere una stazione che abbia un frame da trasmettere: 1. inizialmente la stazione ascolta il canale inattivo, trasmette il suo frame dopo un piccolo periodo di tempo noto come Distributed Inter-Frame Space (DIFS); 2. Altrimenti la stazione sceglie un valore di backoff randomicamente ricorrendo a binary exponential backoff e decrementa questo valore quando il canale viene percepito inattivo. Mentre il canale viene percipito occupato il suddetto valore viene congelato; 3. quando il valore raggiunge lo zero, cosa che può capitare mentre il canale è inattivo, la stazione trasmette l'intero frame e poi attende un acknowledgment; 4. se viene ricevuto l'ack, la stazione trasmittente sa che il frame è stato correttamente ricevuto. Se la medesima stazione ha un altro frame da inviare, avvia il protocollo CSMA/CA dal secondo passaggio. Altrimenti se l'ack non viene ricevuto, la stazione trasmittente rientra nella fase di backoff al secondo passaggio con un valore randomico estratto da un intervallo più grande. **Dealing whit Hidden Terminals: RTS e CTS**: Il protocollo MAC 802.11 implementa anche un elegante schema di prenotazione che aiuta a evitare collisioni anche in presenza di terminali nascosti. Quest'ultimi potrebbero rivelarsi problematici poichè supponendo cheuna stazione s1 stia trasmettendo un frame e nel mezzo della sua trasmissione, la stazione s2 vuole inviare un frame as AP. S2, non potendo ascoltare la trasmissione da s1, in primis, aspetterà un intervallo DIFS per poi trasmettere il frame, generando una collisione. Così facendo il canale verrà dunque sprecato l'intero periodo della trasmisisone di s1 come durante quello di s2. Per risolvere questo problema, ilprotocollo IEEE 802.11 permette alla stazione di ricorrere ad un piccolo Request to Send (RTS) control frame e anche ad un piccolo Clear to Send (CTS) control frame per riservare l'accesso al canale. Quando il trasmittente vuole inviare il DATA frame, innanzitutto invia il frame RTS all'AP, indicando il tempo totale richiesto per la trasmissione dello stesso DATA frame e del frame d'ACK. Quando l'AP riceve il frame RTS, risponderà diffondendo in broadcast il frame CTS. Questo avviene per due motivi: comunica al trasmittente il permesso esplicito di inviare e comunica alle altre stazioni di non trasmettere durante il periodo di tempo riservato. L'uso dei frame di controllo RTS E CTS può incrementare le performance in due modi importanti: 1. il problema della stazione nascosta viene mitigato dal momento che un long DATA frame viene trasmesso solo dopo che il canale è stato riservato; 2. siccome questi due frame sono di piccola dimensione, un'eventuale collisione riguardante essi durerebbe solo per la durata del breve RTS o CTS frame. UNa volta che i due frame vengono trasmessi correttamente i successivi DATA e ACK frames dovrebbero essere trasmessi senza collisione. Nonostante l' RTS/CTS exchange può aiutare a ridurre le collisioni, introduce un ritardo e consuma le risorse del canale. Per questo motivo, l'RTS/CTS exchange è usato solamente per riservare il canale per la trasmissione di un long DATA frame.

# CAP 6.5

**IEEE 802.11 frame**: Nonostante il frame 802.11 condivida delle peculiarità con quello Ethernet, esso contiene anche un numero di campi specifici per l'utilizzo nelle reti wireless. Esaminiamo ora i campi nel frame come alcuni dei più importanti sotto-campi del campo di controllo. Campi Payload e CRC: Il cuore del frame è il campo payload che tipicamente consiste di un daatgramma IP o di un pacchetto ARP che, in genere, non supera i 1500 byte. Inoltre, come in Ethernet, i frame 802.11 hanno un campo CRC di 32 bit che permette al ricevente il rilevamento di eventuali errori nei bit. **Campi indirizzo**: Probabilmente la differenza più sostanziale risiede nel fatto he il frame 802.11 contiene quattro campi indirizzo, ciascuno dei quali può contenere un indirizzo MAC di 6 byte. I primi tre indirizzi sono necessari per scopi d'interconnessione, in particolar modo per trasportare datagrammi del livello di rete da una stazione wireless, attraverso un AP, all'interfaccia router. Invece il quarto indirizzo viene utilizzato nelle reti ad hoc, ma non in quelle con infrastruttura. Siccome fino ad ora abbiamo preso in considerazione infrastrutture network, ci foccalizzeremo sui primi tre campi indirizzo. Lo standard 802.11 definisce questi campi nel seguente modo: 1. l'indirizzo 2 è l'indirizzo MAC della stazione che trasmette il frame. Ergo, se una stazione wireless trasmette un frame, l'indirizzo MAC della medesimia stazione verrà inserito nel campo Indirizzo 2. La stessa cosa vale per gli AP; 2. l'indirizzo 1 è l'indirizzo MAC della stazione wireless che sta ricevendo il frame. Ergo, se una stazione wireless mobile trasmette un frame, il campo Indirizzo 1 contiene l'indirizzo MAC dell'AP di destinazione. La stessa cosa vale per gli AP invertendo le due entità di rete; 3. per comprendere al meglio il campo Indirizzo 3, ricordiamo che il BSS, costituito dall'AP e dalle stazioni wireless, fa parte di una sottorete e quest'ultima connette le altre sottoreti attraverso un'interfaccia del router. L'indirizzo 3 contiene l'indirizzo di questa interfaccia. **Sequence number, duration, and frame control fields**: Il campo numero di sequenza nel pacchetto 802.11 ha esattamente la stessa funzione a livello di collegamento che aveva a livello di trasporto ovvero permette al ricevente di riconoscere un frame appena trasmesso da un altro ritrasmesso. Inoltre 802.11 permette alla stazione trasmittente di riservare il canale per un periodo di tempo in cui avviene la trasmissione del suo frame e la ricezione del frame di conferma. Questo valore è incluso nel campo durata, per i frame DATA, RTS E CTS. Infine il campo di controllo del frame è articolato in sottocampi. I campi tipo e sottotipo sono utilizzati per distinguere frame di associazione, RTS, CTS, ACK e DATA. I campi 'verso AP' e 'da AP' definiscono lla funzione dei diversi campi indirizzo. Il significato di questi cambia a seconda che sia utilizzata una rete ad hoc o con infrastruttura e, in quest'ultimo caso, se il frame è inviatoo da una stazione o da un AP.

**Mobility in the same IP subnnet**: Per incrementare la copertura di una LAN wireless, aziende e università dispongono spesso di vari BSS all'interno di una stessa sottorete IP. Ciò naturalmente fa sorgere il problema della mobilità all'interno dei BSS. Come vedremo, la mobilità può essere trattata in modo semplice quando i BSS fanno parte della stessa sottorete. Quando una stazione di muove da una sottorete all'altra, è necessario un protocollo più sofisticato (IPmobile).

**Accesso cellulare a internet**: Nella trattazione di questo argomento introduciamo lo standard global system for mobile communication (GSM). Spesso òe tecnologie cellulari sono classificate per 'generazioni'. Quelle più datate sono state progettate per il traffico telefonico. I sistemi di prima generazione (1G), erano analoghi ai sistemi FDMA progettati solamente per le comunicazioni audio. Questi sistemi sono stati sostituiti dai 2G digitali, progettati anch'essi per la fonia. Più tardi furono estesi a 2.5G per supportare i traffico dati. Attualmente i sistemi 3G supportano sia la voce che i dati, con una maggiore enfasi sulla capacità di trasporto dati e collegamenti di accesso radio. Dopodichè abbiamo i sistemi 4G che sono all'avanguardia, basati su una tecnologia LTE, nucleo di rete completamente basato su IP e forniscono voce e dati a velocità di Megabit.

**Architettura della rete cellulare 2G: connessioni voce alla rete telefonica**: Il termine cellulare si riferisce al fatto che un'area geografica è suddivisa in aree di copertura dette celle. Anche GSM ha una sua nomenclatura particolare. Ogni cella contiene una stazione di base che prende il nome di base transceiver station (BTS) che scambia segnali con le stazioni mobili. L'area di copertura di una cella dipende da molti fattori, inclusi la potenza di trasmissione del BTS e quella della stazione mobile, la presenza di palazzi nella cella, e l'altezza dell'antenna della stazione base. Lo standard GSM per i sistemi cellulari 2G usa una combinazione di FDM e TDM per l'interfaccia aerea. In questi sistemi ibridi FDM/TDM, il canale è ripartito in sottobande di frequenza all'interno delle quali il tempo è suddiviso in frame e slot. Quidni, in questi sistemi, se il canale è suddiviso in F sottobande e il tempo in T slot, i canali potranno supportare F x T chiamate simultanee. Anche le reti di accesso HFC usano un approccio ibrido. Una base station controller (BSC) nelle reti GSM serve tipicamente alcune decine di BTS. Il ruolo dei BSC è di allocare canali radio BTS agli utenti mobili, eseguendo la procedura di -paging-, che consiste nel trovare la cella in cui l'utente mobile risiede, ed eseguire l'handoff dell'utente mobile. Il BSC insieme alle BTS che controlla costituiscono il -base station system- (BSS) di una rete GSM. Il -mobile switching center- (MSC) gioca un ruolo fondamentale nell'autorizzazione ed identificazione degli utenti, nello stabilire o terminare una chiamata e nell'handoff. (Tipicamente, un singolo MSC contiene fino a 5 BSC, pari circa a 200.000 utenti per MSC).

**3G core network**: Il nucleo della rete dati cellulare 3G connette reti di accesso radio a Internet. La core network interopera con le componenti della rete cellulare esistente per la voce, in particolare l'MSC. Alla luce dell'infrastruttura già esistente in 2G i progettisti del servizio dati 3G hanno scelte il seguente approccio: lasciare invariato il nucle GSM esistente per la fonia, integrando parallelamente funzionalità di trasmissione dati. Alternativamente l'approccio che portava all'integrazione di nuovi servizi dati direttamente dal nucleo della rete voce cellulare già esistente avrebbe comportato le stesse sfide illustrate precedentemente per passare da IPv4 a IPv6. Nella core network 3G ci sono due tipi di nodi: -Serving GPRS Suppport Nodes (SGSN)- e -Gateway GPRS Support Nodes- (GGSN). GPRS è un acronimo che sta per Generalized Racket Radio Service ovvero un servizio implementato in 2G. Qui discutiamo la versione evoluta di quest'ultimo implementata nelle reti 3G. Un SGSN ha il compito di consegnare datagrammi a/da nodi mobili nella rete di accesso radio a cui l'SGSN è collegato. Quest'ultimo interagisce con l'MSC della rete voce cellulare di quell'area, fornendo agli utenti autorizzazioni e handoff, memorizzando le informazioni di posizione (cella) dei nodi mobili attivi e inoltrando i datagrammi sempre tra i nodi mobili di una rete di accesso radio e un GGSN, che agisce come gateway connettendo più SGSN a Internet. Quindi GGSN è la parte terminale dell'infrastruttura 3G.

**Rete di accesso radio 3G: il confine wireless**: Le 3G -radio access network- è la rete wireless di primo hop vista da un utente 3G. Il -Radio Network Controller- (RNC) controlla tipicamente alcune stazioni base simili ai BTS incontrati nei sistemi 2G che nel gergo UMTS vengono chiamati "Node B". Ogni collegamento wireless all'interno di una cella opera tra i nodi modibili e una BTS, come nelle reti 2G. L'RNC è connesso sia alla rete voce cellulare a commutazione di circuito tramite un MSC sia a Internet a commutazione di pacchetto tramite SGSN. Ergo sebbene i servizi voce e i servizi dati 3G usino nuclei di reti differenti, hanno in comune la rete di accesso radio di primo ed ultimo hop. Un cambiamento significativo avviene in UMTS dove si arriva ad usare una tecnica CDMA nota come Direct Sequence Wideband CDMA (DS-WCDMA) all'interno degli slot TDMA al posto dello schema GSM FDM/TDM. Questo cambiamento richiede una nuova rete di accesso cellulare 3G che operi in parallelo con la rete radio BSS 2G. Il servizio dati associato alle specifiche WCDMA è noto come HSPA (high speed packet access) e prospetta una velocità nell'ordine di Megabit.

**Gestione della mobilià –> Principi** : L'utente fisicamente mobile presenta una serie di sfide a livello di rete, a seconda di come si sposta tra i punti di accesso alla rete. A un'estrmità dello spettro l'utente potrebbe trasferire il suo portatile con una scheda di rete wireless all'interno di un edificio. Questo non è un utente mobile sia dal punto di vista di rete che di collegamento, se si associa allo stesso AP indipendentemente dalla posizione. All'altra estremità dello spettro consideriamo un utente che viaggia, passando attraverso diverse reti di accesso wireless. Questo utente è mobile. Ovviamente tra i due esempi c'è un'infinità di casistiche di utenti mobili.

L'importanza che l'indirizzo del nodo mobile resti invariato è relativa alle applicazioni in esecuzione: se l'entità mobile è in grado di mantenere il proprio indirizzo IP quando si sposta, la mobilità diventa trasparente dal punto di vista dell'applicazione. Questa trasparenza è molto importante infatti se un'applicazione non si deve preoccupare circa un potenziale cambio di indirizzo IP, mentre lo stesso codice applicativo serve connessioni mobili e non mobili.

Nel caso in cui un'infrastruttura di rete venisse meno, le reti ad hoc forniscono precisamente le funzionalità necessarie.

In una rete, il luogo permanente in cui risiede un nodo mobile è detto -rete di appartenenza- (home network) e le entità che gestisconola mobilità per conto del nodo mobile all'interno di questa sono conosciute come -agenti domestici- (home agent). La rete in cui il nodo mobile viene a trovarsi occasionalmente costituisce la -rete ospitante- (foreign network) mentre l'entità al suo interno che si occupa della mobilità è detta -agente ospitante- (foreign agent). Il -corrispondente- è l'entità che desidera comunicare con il nodo mobile.

**Indirizzamento/ Addressing**: L'approccio è di portare le funzionalità di mobilità dal nucleo della rete alla sua periferia. Una modalità intuitiva di questo approccio utilizza la rete di appartenenza del nodo mobile. Certamente è necessario un protocollo tra il nodo mobile e l'agente domestico per aggiornare la posizione/localizzazione del nodo stesso. Osserviamo attentamente l'agente ospitante. L'approccio concettualmente più semplice è porre l'agente ospitante nei router agli estrmi della rete visitata. Un compito di questo agente è la definizione di un -indirizzo di mediazione-, detto -COA- (care-of-address) per il nodo mobile. La parte direte del COA sarà quella della rete ospitante. Gli indirizzi associati alla modalità mobile sono quindi due: quello -permanente- e il COA.

**Routing to a mobile node** : Per indirizzare e inviare il datagramma al nodo mobile esistono due approcci: l'instradamento diretto e indiretto.

**Indirect routing to a mobile node** : Nell'approccio dell'-instradamento indiretto-, il corrispondente non fa altro che indirizzare il datagramma all'indirizzo permanente del nodoe inviarlo nella rete, completamente inconsapevole dell'effetiva localizzazione del nodo. Dunque la mobilità è cristallina al corrispondente.

Alcuni datagrammi sono inizialmente instradati alla rete di appartenenza del nodo mobile. Focalizziamo l'attenzione sull'agente domestico che, oltre alla responsabilità di interagire con l'agente ospitante per monitorare il COA del nodo mobile, ha un'altra funzione molto importqante: il controllo dei datagrammi in entrata indirizzati ai nodi che fnno parte della rete di appartenenza, ma che si trovano al momento in una rete esterna. L'agente intercetta questi datagrammi e li invia al nodo mobile con un processo articolato in due passi. Il datagramma è innanzitutto inviato all'agente ospitante del nodo mobile, utilizzando il COA di quest'ultimo e poi viene inoltrato al nodo mobile stesso. Nella fase di re-instradamento l'agente domastico avrà bisogno di indirizzare il datagramma utilizzando il COA del nodo mobile mantenendo intatto il datagramma. Ciò si può ottente attraverso l'-incapsulamento-, da parte dell'agente domestico, dell'intero datagramma originale in uno nuovo di dimensioni maggiori. Quest'ultimo è indirizzato e inviato al COA del nodo. L'agente ospitante, che conosce il COA, riceverà e aprirà il datagramma più grande per recuperare quello incapsulato, e lo invierà al nodo mobile. Lo standard IP mobile utilizza l'approccio dell'instradamento indiretto.

**Direct routing to a mobile node** : L'instradamento indiretto comporta un'efficienza conosciuta come -problema dell'instradamento triangolare- (triangle routing problem), in quanto i datagrammi indirizzati al nodo mobile devono prima essere instradatiall'agente domestico e poi alla rete ospitante, anche in presenza di percorsi più efficienti tra corrispondente e nodo mobile. L'-instradamento diretto- supera l'inefficienza insita nell'instradamento triangolare, ma al costo di una maggiore complessità. In questo approccio un -agente corrispondente-, nella rete del corrispondente, ottiene innanzitutto il COA del nodo mobile. Ciò può essere fatto con una richiesta presso l'agente domestico, assumendo che il nodo mobile disponga di un valore aggiornato del suo COAe che questo sia noto all'agente domestico. E' anche possibile che il corrispondente svolga la funzione di agente corrispondente, così come il nodo mobile potrebbe svolgere la funzione di agente ospitante. L'agente corrispondente, poi, invia tramite un tunnel i datagrammi direttamente al COA del nodo mobile, in modo analogo a come faceva l'agente domestico. Così facendo, l'instradamento diretto risolve il problema dell'instradamento triangolare ma, al contempo, introduce due ulteriori, fondamentali problemi: 1. la necessità di un -protocollo di localizzazione dell'utente mobile- tramite il quale l'agente corrispondente possa interrogare l'agente domestico oer ottenere il COA del nodo mobile; 2. come avviene l'invio dei dati all nuova rete quando il nodo mobile si sposta da una rete a un'altra? Per risolvere questo problema nell'instradamento triangolare bastava aggironare il COA presso l'agente domenstico. Al contrario con l'instradamento diretto, l'agente domestico è interrogato dall'agente corrispondente solo una volta, all'inizio della sessione. Così, aggiornare il COA nell'agente domestico non risolverà il problema dell'instradamento dei dati verso la nuova rete del nodo mobile. Una soluzione potrebbe essre un nuovo protocollo per avvisare il corrispondente del cambio di COA. Un'altra soluzione, come nel caso delle GSM, agisce non seguente modo: si supponga che, in una determinata sessione, siano inviati dati a un nodo mobile nella rete ospitante, nella quale si trova inizialmente e identifichiamo l'agente in quella rete come -agente di appoggio- (anchor foreign agent). Quando il nodo si sposta in una nuova rete, registra il nuovo COA presso il nuovo agente ospitante, che fornirà all'agente di appoggio il nuovo COA del nodo mobile. Quando l'agente di appoggio riceverà il datagramma incapsulato per il nodo non più presente, lo re-incapsulerà e lo invierà a quel nodo utilizzando il nuovo COA. Se infuturo il nodo si sposterà ancora in un'altra rete, l'agente della nuova rete ospitante dovrà contattare l'agente di appoggio per impostare l'invio dei datagrammi verso quella nuova rete.

**Mobile IP**: il mobile IP standard consiste in tre parti: Agent Discovery, Registration with home agent, Indirect routing of datagrams. Agent Discovery: quando un mobile IP node arriva in un nuovo network, deve capire l'identità del corrispondente foreign o home agent; infatti è proprio la scoperta di un nuovo foreign agnet che fa capire al mobile node di essere entrato in un nuovo network. Questo processo, chiamato Agent Discovery, è diviso in due parti. Nella prima, chiamata Agent Advertisement, l'agente manda periodicamente un pacchetto ICMP con il campo "type" settato a 9 (router discovery) su tutti i link su cui è connesso. Questo router discovery message contiene l'IP address del router e contiene anche tutte le informazioni necessarie al mobile node, tra cui: Home agent bit (per indicare se l'agent è un home agent per il network in cui risiede), Foreign agent bit (il viceversa), Registration required bit (indica che il mobile node si deve registrare con un foreign agent, e non può assumere se stesso la funzione di foreign agent), encapsulation bit (indicano se viene usata una forma diversa da IP-in-IP encapsulation), Care-of Address fields (una lista di care-of addresses creata dal foreign agent, da cui il mobile node ne deve scegliere uno). Invece, nell'agent solicitation, un mobile node manda un messaggio di sollecitazione a un agent (cioè un ICMP message con campo "type" a 10). L'agente che lo riceve a questo punto manda in unicast al mobile node un agent advertisement (spiegato prima). Una volta che un mobile IP node ha ottenuto un care-of address, questo address deve essere registrato con l'home agent, direttamente oppure attraverso un foreign agent. Nel caso venisse registrato attraverso un foreign agent, vengono compiuti quattro passi: Primo, il mobile node manda un mobile IP registration message al foreign agent, usando un UDP datagram mandato alla porta 434, contenente un care-of address(COA), l'address dell'home agent(HA), l'address permanente del mobile node(MA), la requested lifetime della registrazione, e un numero a 64 bit che rappresenta la registration identification, che serve a collegare una certa received registration reply con una certa registration request. Secondo: il foreign agent manda un mobile IP registration message all'home agent, con lo stesso formato e contenuto di quello ricevuto dal mobile node (con aggiunta l'encapsulation format), e si segna il permanent address del mobile node. Terzo: l'home agent controlla la correttezza e l'autenticità della registrazione; nel caso di esito positivo, si segna il permanent address del mobile node e il suo care-of address; nel futuro, tutti i datagram che arrivano all'home agent indirizzati al mobile node verranno rimandati in tunnel al care-of address; inoltre, rimanda indietro una registration reply. Quarto: il foreign agent riceve la registration reply e la rimanda al mobile node. A questo punto la registrazione è completa (attenzione: la lifetime specificata nella registration reply è più piccola della lifetime richiesta). Un foreign agent non si deve occupare di deregistrare un COA quando un mobile node lascia il network; infatti questo avverà automaticamente appena lì mmobile node entra in un nuovo network e registra un nuovo COA.

**GSM**: come il mobile IP, GSM adotta un routing approach indiretto, prima mandando la chiamata del corrispondente al mobile user's home network, e da lì viene mandata al visited network. In GSM, l'home network del mobile user prende il nome di home public land mobile network (home PLMN). Questo home network è in pratica il provider telefonico a cui il mobile user è iscritto. Il visited PLMN non è altro che il network in cui il mobile user risiede correntemente. Le responsabilità dell'home network e del visited sono molto diverse, in quanto: l'home network mantiene un database chiamato home location register (HLR), che contiene i numeri di telefono e le informazioni degli iscritti (tra cui dove si trovano gli iscritti); inoltre, un switch speciale chiamato Gateway Mobile services Switching Center (GMSC, conosciuto anche come home MSC) viene contattato quando viene fatta una chiamata a un mobile user. Invece, il visited network tieneun visitor location register (VLR), dove ci sono le informazioni di ogni mobile router che è correntemente nel visited network. In pratica, il visited network è il network di un provider a cui un mobile user non è iscritto.

**GSM Calls** (cosa succede se chiamo un cellulare): un utente digita il numero di un cellulare. Le cifre non significano quasi nulla (tranne le prime che identificano l'home network). La chiamata viene instradata dal chiamante attraverso il PSTN all'home MSC dell'home network del ricevente della chiamata. L'home MSC, ricevuta la chiamata, interroga il HLR per capire dove cazzo sta sto cellulare de merda. HLR, se sei sculato, te risponne con un mobile station roaming number (MSNR), un numero assegnato a ogni cellulare de merda che entra in un cazzo di visited network, e ha lo stesso utilizzo di un fottuto COA. Se invece c'hai la puzza ar culo e non c'hai sto MSNR, sto HLR te ritorna l'indirizzo del VLR del visited network dove sta sto cellulare de merda. Allora te t'attacchi ar cazzo, devi contatta sto VLR e aspetta che te risponne e te da lui sto cazzo de MSNR. Na volta che c'hai sto MSRN, l'home MSC può fa sta chiamata de merda, dal chiamante all'home MSC, dall'home MSC al visited MSC, e da là alla base station dove sta sto czzo de cellulare de merda. Se ti stai chiedendo come fa l'home MSC a sape in quale VLR sta il cellulare de merda, devi sape che ogni volta che un cellulare de merda entra in un visited network, il VLR dice all'HLR del cellulare de ripijasse e che er cellulare ha imboccato da lui.

**GSM Handoff**: un handoff si verifica quando un mobile user cambia base station durante una chiamata. Potrebbe anche accadere perchè una base station è troppo sovraccarica oppure viene staccata dalla rete, non è detto che sia il cellulare a spostarsi. Un cellulare periodicamente misura la potenza del segnale non solo della base station associata, ma anche delle adiacenti, e manda tutte queste misurazioni alla base station una o due volte al secondo. Quindi l'handoff viene iniziato dalla base station associata in base a queste misurazioni. Il processo è lungo, e funziona così: la vecchia base station (BS) informa il visited MSC e il nuovo BS che sta per avvenire un handoff; il visited MSC inizia un path setup per la nuova BS, e avvisa il nuovo BS dell'handoff; il nuovo BS alloca e attiva un radio channel per il cellulare; il nuovo BS risponde al visited MSC e al vecchio BS che il path è stato stabilito e che bisognerebbe avvisare il cellulare; il cellulare viene informato del fatto che deve eseguire un handoff; il cellulare e il nuovo BS si scambiano dei messaggi per attivare bene il nuovo canale con il nuovo BS; il cellulare manda un handoff complete message al nuovo BS, che viene inoltrato al visited MSC, il quale fa il rerouting della chiamata in corso al nuovo BS; le risorse allocate dal vecchio BS vengono rilasciate; fine. Tutto questo però ipotizzanto che l'handoff avviene dentro lo stesso MSC. Se invece il cellulare si sposta a n MSC diverso, succede che quello su cui stava prima diventa un anchor MSC, e d'ora in poi continuerà a occuparsi lui di fare il rerouting della chiamata dall'home MSC al nuovo MSC in cui il cellulare è entrato. Quindi, si otterrà che la chiamata attraverserà un massimo di tre MSC: home, anchor, e visited (quello dove sta il cellulare correntemente).

**Mobility impact on higher protocols**: mentre molti protocolli non vengono influenzati dal fatto che la connessione è wireless oppure no, alcuni come TCP ne soffrono molto. Infatti, molti pacchetti vengono persi per bit errors o per handoff delays, mentre invece TCP interpreta ogni pacchetto perso come congestion, e riduce la congestion window quando non è necessario. Per parare a questo problema ci sono tre modi: Local recovery (protocolli per recuperare bit errors), TCP sender awareness of wireless links , oppure Split-connection approaches (la connessione tra due end points è divisa in due transport layer connections, una dal mobile host alla base station, e una dalla base station all'altro end point, in modo tale da poter usare protocolli di trasporto speciali come un UDP modificato per il tratto wireless). Inoltre, le connessioni mobili influenzano molto il livello applicativo in quanto soffrono di una bandwidth ridotta.