

Reti di
Calcolatori

93315 - 12CFU

Disclaimer: questi appunti sono un mix, ho integrato
sia il file `appunti-pdfone.pdf` che il libro
“Reti di calcolatori e Internet – JFK, KWR” quindi ci
sono blocchi di testo presi da essi.

Sono gratuitamente pubblicati su
csunibo.github.io/reti-di-calcolatori/appunti

Se trovate errori fatemelo pure sapere scrivendo a
✉ alberto.zuccari@studio.unibo.it oppure su
.telegram [@b3rt0nes](https://t.me/b3rt0nes)

0 SOMMARIO

1	Reti di calcolatori e internet	7
1.1	Mezzi fisici di trasmissione	8
1.2	Commutazione di pacchetto	8
1.3	Commutazione di circuito.....	9
1.4	Comm. Pacchetto Vs Circuito	9
1.5	Reti di reti	9
1.6	Ritardi e perdite.....	10
1.6.1	Ritardo di accodamento e perdita dei pacchetti	10
1.7	Traceroute	10
1.7.1	Troughput	11
1.8	Architettura a livelli	11
1.8.1	Modello OSI-----	11
1.9	Reti sotto attacco	12
2	Livello di applicazione.....	12
2.1	Introduzione	12
2.1.1	Processi	13
2.1.2	Socket	13
2.1.3	Servizi di trasporto disponibili per le applicazioni-----	13
2.2	Web e http.....	13
2.2.1	HTTP non persistente	14
2.2.2	HTTP persistente	14
2.2.3	HTTP request message-----	15
2.2.4	HTTP response message-----	15
2.2.5	I Cookies	16
2.2.6	Web caches (proxy server)-----	16
2.3	Electronic mail	17
2.3.1	STMP	17
2.3.2	Protocolli di accesso mail-----	17
2.4	DNS	18
2.4.1	Risoluzione di un nome	18
2.4.2	Caching-----	19
2.4.3	Resource record -----	19
2.4.4	Formato dei messaggi DNS	19
2.4.5	Attaccare un DNS-----	20
2.5	Client-server Vs P2P.....	20

2.6	Servizi con architettura P2P.....	21
2.6.1	Distribuzione di file: BitTorrent -----	21
2.6.2	Streaming video -----	21
2.6.3	CDN (Content Distribution Networks)-----	22
2.7	Programmazione Socket.....	22
2.7.1	Programmazione con UDP -----	22
3	Livello di trasporto.....	25
3.1	Servizi di livello trasporto	25
3.2	Multiplexing e demultiplexing.....	25
3.2.1	Mux e demux non orientati alla connessione -----	26
3.2.2	Mux e demux orientati alla connessione -----	26
3.3	Trasporto non orientato alla connessione: UDP	27
3.3.1	Struttura del segmento UDP-----	27
3.3.2	Checksum UDP -----	28
3.4	Principi di trasferimento affidabile dei dati.....	28
3.4.1	Canale perfettamente affidabile: rdt1.0-----	28
3.4.2	Canale con errori sui bit: rdt2.0-----	29
3.4.3	Canale con errori sui bit: rdt2.1-----	30
3.4.4	Canale con perdite & errori sui bit: rdt3.0 -----	30
3.4.5	Trasferimento affidabile con pipeline-----	31
3.4.6	GBN -----	31
3.4.7	Ripetizione selettiva (SR) -----	32
3.5	Trasporto connection-oriented: TCP	33
3.5.2	Struttura dei segmenti TCP: -----	33
3.5.3	RTT e timeout -----	34
3.5.4	Trasferimento affidabile dei dati -----	34
3.5.5	Controllo di flusso-----	35
3.5.6	Gestione della connessione-----	35
3.6	Principi di controllo delle congestioni	35
3.6.1	Due mittenti e un router con buffer illimitati -----	36
3.6.2	Due mittenti e un router con buffer limitato-----	36
3.6.3	Quattro mittenti, router con buffer finiti e percorsi composti da più collegamenti-----	36
3.7	Controllo di congestione TCP	37
3.7.1	Fairness -----	38
3.7.2	Notifica esplicita di congestione (ECN)	38
4	Livello di rete: il piano dei dati	39

4.1.1	Piano dei dati Vs Piano di controllo	39
4.2	Modelli di servizio a livello rete	39
4.3	Cosa si trova all'interno di un router?	40
4.3.1	Struttura delle porte di input	40
4.3.2	Struttura di commutazione	41
4.3.3	Accodamento in ingresso	41
4.3.4	Accodamento in uscita	42
4.3.5	Scheduling	42
4.4	Internet Protocol (IP)	42
4.4.1	Indirizzamento IPv4	43
4.4.2	CIDR (Classless InterDomain Routing)	44
4.4.3	DHCP (Dynamic Host Configuration Protocol)	44
4.4.4	NAT (Network Address Translation)	44
4.4.5	IPv6	46
4.4.6	Inoltro generalizzato e SDN	47
5	Livello di rete: il piano di controllo	49
5.1	Protocolli di routing	49
5.1.1	Algoritmi link-state	50
5.1.2	Algoritmi Distance vector	51
5.1.3	Instradamento interno ai sistemi autonomi	51
5.2	intra-AS: OSPF	52
5.3	inter-AS: BGP	52
5.4	SDN Control plane	53
5.4.1	Traffic engineering	53
5.5	ICMP (Internet Control Message Protocol)	53
5.6	SNMP (Simple Network Management Protocol)	53
6	Crittografia	54
6.1	Crittografia con chiave simmetrica	54
6.2	Crittografia con chiave pubblica	55
6.2.1	RSA (Rivest Shamir Adleman)	55
6.2.2	Autenticazione nelle reti	56
6.2.3	Firma digitale	56
6.2.4	Message digest	57
6.2.5	Cifratura ibrida	57
6.2.6	E-mail sicura	57
6.3	SSL-TLS	58

6.4	IP Sec.....	59
6.5	Firewall	59
6.5.1	Stateless firewall-----	59
6.5.2	Statefull firewall -----	59
6.5.3	Application gateway firewall -----	59
6.5.4	DMZ (Zona Demilitarizzata)-----	59
7	Wireless	60
7.1	Proprietà delle RF (Radio Frequenze).....	60
7.2	Propagazione delle RF	61
7.3	Trasmissione wireless.....	63
7.3.1	Guadagno di segnale (misurato in decibel, dB)-----	63
7.3.2	Perdita del segnale: (dB)-----	63
7.3.3	Effetti propagazione del segnale -----	63
7.3.4	Propagazione multipercorso-----	64
7.4	Progettazione del sistema (dal punto di vista energetico).....	66
7.4.1	mW – dBm: tabella di conversione -----	67
7.5	Antenne	67
7.5.1	Antenne omnidirezionali -----	68
7.5.2	Antenne semi-direzionali-----	69
7.5.3	Antenne Molto-Direzionali (Highly-directional) -----	69
7.5.4	Antenne settorizzate -----	70

1 RETI DI CALCOLATORI E INTERNET

Internet

è una rete di calcolatori. Tutti i dispositivi della rete sono detti **host** o **sistemi periferici**.

Gli host sono connessi tra loro tramite una **rete di collegamenti** e **commutatori di pacchetti**.

Collegamenti diversi trasmettono a velocità diverse, questa velocità è detta **velocità di trasmissione** ed è misurata in *bit per secondo (bps)*.

Quando un host deve mandare informazioni lo fa sotto forma di **pacchetti**

Commutatore di pacchetto

È quel dispositivo che prende i pacchetti in entrata, li lavora, e li invia. I principali sono:

- Router
- Comutatore a livello collegamento

Gli host accedono ad internet tramite gli **ISP** (Internet Service Provider)

Provider

È un insieme di commutatori di pacchetto e di collegamenti.

Ci sono vari protocolli che regolano lo scambio dei pacchetti nella rete:

- **TCP (Transmission Control Protocol)**
- **IP (Internet Protocol)**

Gli host connessi ad internet forniscono delle **API**, ovvero un insieme di regole da seguire per trasmettere correttamente.

Ogni attività su internet deve seguire un determinato **protocollo**

Un **protocollo** definisce il formato e l'ordine dei messaggi scambiati tra due o più entità in comunicazione.

Gli host sono divisi in:

Client

→ host che richiedono servizi

Server

→ host che forniscono servizi

Una **rete di accesso** è una rete che connette un sistema al suo **Edge router** (è il primo router incontrato nel percorso mittente – destinatario)

Le due tipologie più diffuse sono:

- DSL → ovvero attraverso la rete telefonica
- Via cavo → attraverso l'infrastruttura delle televisioni

1.1 MEZZI FISICI DI TRASMISSIONE

I mezzi fisici di trasmissione si possono dividere in 2 macrocategorie

- Vincolanti (il segnale è mantenuto in un mezzo fisico)
- Non Vincolanti (il segnale si propaga nell'atmosfera)

- DOPPINO DI RAME INTRECCIATO → due fili di rame distinti disposti a spirale (vengono disposti a spirale per evitare interferenze)
- CAVO COASSIALE → due conduttori di rame concentrici
- FIBRA OTTICA → tubo che conduce impulsi luminosi, 1 impulso = 1 bit; è immune all'interferenza.
- CANALI RADIO → Segnale viene trasportato all'interno dello spettro elettromagnetico
 - Dipende da *ambiente* (per riflessioni su superfici o interferenza) e *distanza* (per la perdita di segnale)
- CANALI RADIO SATELLITARI → onde trasmesse attraverso un satellite / ° \

1.2 COMMUTAZIONE DI PACCHETTO

Come vengono mandati i messaggi?

1. La sorgente prende il messaggio da spedire
2. Il messaggio viene suddiviso in "pacchetti" di lunghezza L
3. I pacchetti vengono spediti ad una velocità R

Quindi il tempo di trasmissione è L/R (L pacchetti ogni R)

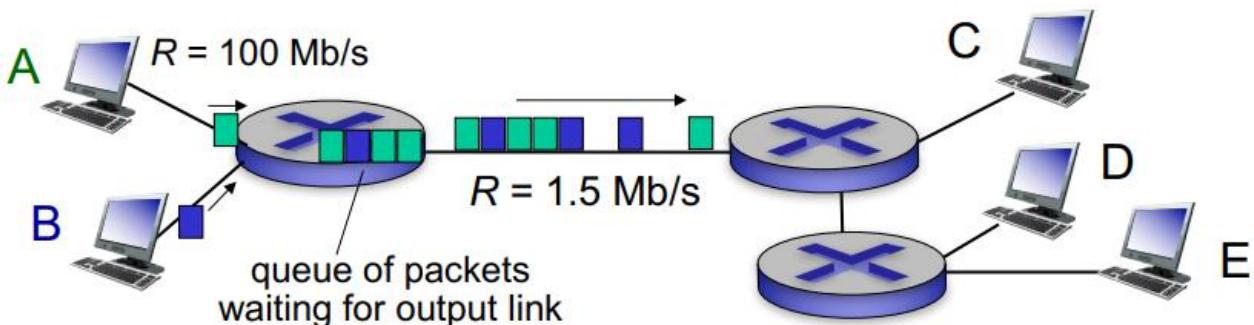
Store-and-forward, è una tecnica che utilizza il commutatore, in pratica il commutatore deve ricevere tutto il pacchetto prima di spedirne un altro.

Buffer di output, è una struttura dati del commutatore che contiene i pacchetti da mandare.

Se il pacchetto arriva al commutatore e trova il canale occupato si mette *in coda nel buffer*.

Oltre al ritardo dello S&F abbiamo il ritardo dovuto dal buffer

Dato che la dimensione del buffer è limitata, se la coda del buffer è piena può verificarsi la **perdita di un pacchetto**.



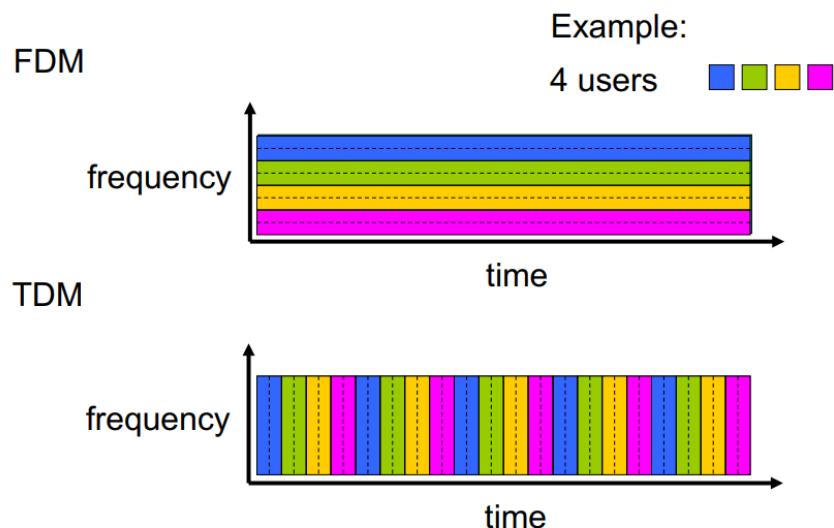
Come vedere dove mandare il pacchetto?

Ogni pacchetto ha, nella intestazione, l'indirizzo della sua destinazione; quando il pacchetto arriva al router, quello controlla la sua **tavella di instradamento (forwarding table)** che lo *instrada* per l'appunto con i collegamenti in uscita.

1.3 COMMUTAZIONE DI CIRCUITO

Un circuito all'interno di un collegamento è implementato tramite **FDM** (Frequency Division Multiplexing) o **TDM** (Time Division Multiplexing)

- ➔ **FDM** il collegamento dedica una banda della frequenza per ogni connessione
 - ➔ **TDM** il tempo viene diviso in frame della stessa durata e suddivisi per le connessioni



1.4 COMM. PACCHETTO Vs CIRCUITO

Secondo alcuni, la commutazione di pacchetto non è adatta a servizi in tempo reale a causa dei suoi ritardi end-to-end variabili.

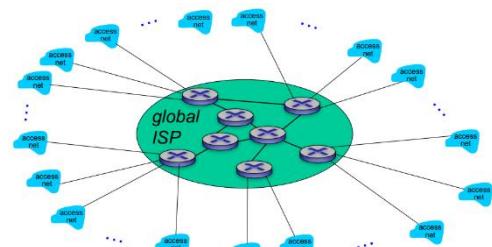
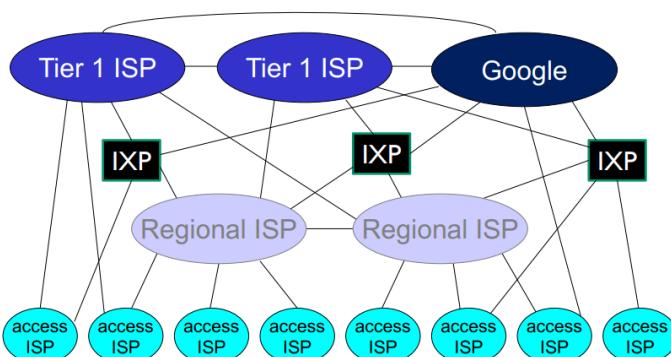
Per altri, la commutazione di pacchetto offre una migliore condivisione della larghezza di banda ed è anche più semplice, più efficiente e meno costosa.

1.5 RETI DI RETI

I sistemi periferici si connettono ad internet tramite gli ISP, però per completare il quadro generale gli ISP devono essere interconnessi tra loro.

Abbiamo perciò 5 strutture di rete (una più avanzata dell'altra)

1. Interconnette tutti gli access ISP con un unico ISP globale →
 2. Gerarchia a 2 livelli: i provider globali in alto e quelli di accesso alla base, ci sono poi i provider regionali che fanno da tramite



3. Per la struttura 3 ci sono dei provider regionali più importanti di altri (come una gerarchia interna alla regione)
 4. **Internet effettivamente:**
aggiungiamo i **POP** (Point of presence), un gruppo di router vicini tra loro nella rete del provider.
 5. Si aggiungono i **Content provider networks** (coloro che forniscono i contenuti)

1.6 RITARDI E PERDITE

Un pacchetto durante il suo percorso da mittente a destinatario può subire vari ritardi:

Ritardo di elaborazione	d_{elab}	tempo per esaminare l'intestazione del pacchetto e decidere dove mandarlo
Ritardo di accodamento	d_{acc}	tempo che il pacchetto passa nel buffer aspettando la trasmissione
Ritardo di trasmissione	d_{trans}	tempo impiegato per trasmettere tutti i bit L/R
Ritardo di propagazione	d_{prop}	tempo impiegato per propagarsi fino al router, è dato da d/v (dove d = distanza e v = velocità)

Il ritardo totale della trasmissione d_{nodo} è dato da:

$$d_{nodo} = d_{elab} + d_{acc} + d_{trans} + d_{prop}$$

1.6.1 Ritardo di accodamento e perdita dei pacchetti

Il ritardo di accodamento varia da pacchetto a pacchetto

Siano: a = velocità media di arrivo dei pacchetti

L = Lunghezza del pacchetto

R = velocità di trasmissione

La velocità media di arrivo dei bit in coda è $a \cdot L$

L'intensità di traffico è data dal rapporto tra la velocità media di arrivo dei bit in coda e la velocità di trasmissione $\Rightarrow a \cdot L/R$

- Se $La/R > 1 \Rightarrow$ ritardo di accodamento infinito
- Se $La/R \approx 1 \Rightarrow$ poco ritardo di accodamento
- Se $La/R \leq 1 \Rightarrow$ abbastanza ritardo di accodamento

Le code hanno una capacità finita, quindi se non si riescono a smaltire i pacchetti e la coda si satura avremo una perdita di pacchetti.

1.7 TRACEROUTE

È un metodo usato per calcolare quanto ritardo ha fatto il pacchetto.

Quando l'utente immette l'indirizzo di destinazione, il modulo dell'utente invia un certo numero di pacchetti speciali alla destinazione.

Durante il loro percorso questi pacchetti passeranno attraverso una serie di router. Quando un router riceve uno di questi pacchetti speciali spedisce un messaggio indietro al mittente. Questo messaggio contiene *nome* e *indirizzo* del router.

In questo modo il computer riesce a calcolare i ritardi e a ricostruire il percorso che i pacchetti hanno fatto.

Esempio:

Nome Router	Indirizzo IP Router	Ritardo A/R
1. cs-gw	(128.119.240.254)	01 ms 01 ms 02 ms
2. border1-rt-fa5-1-0.gw.umass.edu	(128.119.3.145)	01 ms 01 ms 02 ms
3. cht-vbns.gw.umass.edu	(128.119.3.130)	06 ms 05 ms 05 ms
4. jn1-at1-0-0-19.wor.vbns.net	(204.147.132.129)	16 ms 11 ms 13 ms
5. jn1-so7-0-0-0.wae.vbns.net	(204.147.136.136)	21 ms 18 ms 18 ms

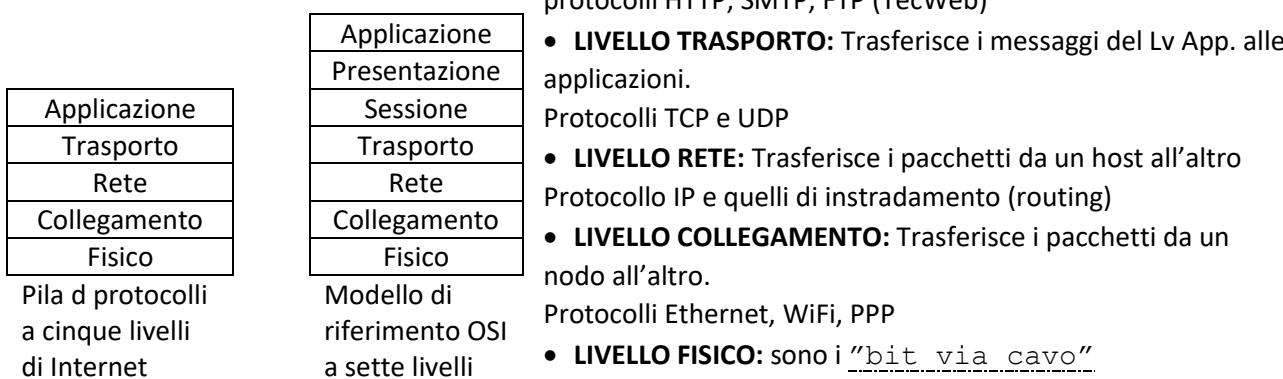
1.7.1 Troughput

È la velocità con cui i bit sono trasferiti dal mittente al destinatario. Può essere:

- Istantaneo: velocità in un dato punto del tempo
- Medio: velocità in un certo periodo di tempo. Sarà $\frac{F}{T}$ con F = bit e T = secondi

1.8 ARCHITETTURA A LIVELLI

Come quando viaggiamo in aereo ci sono degli step da seguire per viaggiare da un aeroporto all'altro anche con i *viaggi* dei segnali bisogna seguire degli step, si crea quindi uno stack di livelli ai quali lavorare.

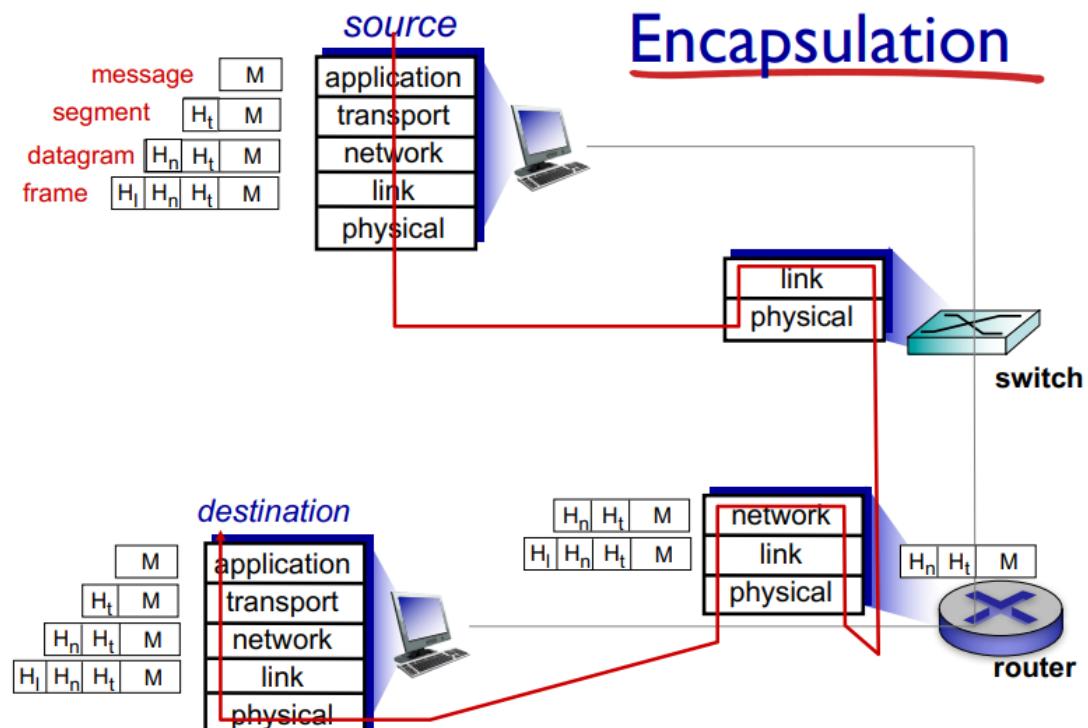


1.8.1 Modello OSI

È il modello ISO (International Standardization Organization)

OSI → Open System Interconnection

- **LIVELLO PRESENTAZIONE**: Interpreta il significato dei dati scambiati
- **LIVELLO SESSIONE**: fornisce la delimitazione e la sincronizzazione dello scambio dei dati.



1.9 RETI SOTTO ATTACCO

I **malware** sono contenuti che possono entrare nel nostro computer ed infettarlo. Si dividono in:

- **Virus** → interagiscono con l'utente
(link fake nelle mail)
- **Worm** → non interagiscono con l'utente
(pagine finte)

Botnet → è una rete che contiene solo computer infettati da malware, detti bot o zombie (è come un esercito di zombie)

Le botnet possono effettuare attacchi come quelli di **negazione al servizio (DOS – Denial Of Service)** ovvero un attacco che rende inutilizzabile dagli utenti in rete un host o un'infrastruttura.

Packet sniffer → è un ricevitore passivo che memorizza i pacchetti transiti su di una rete WiFi

IP Spoofing → è la capacità di trasmettere pacchetti con un IP di origine falso, è il modo in cui ci si spaccia per qualcun altro.

2 LIVELLO DI APPLICAZIONE

In questo capitolo vogliamo vedere:

- | | |
|---|---|
| <ul style="list-style-type: none">• Gli aspetti implementativi concettuali dei protocolli di rete per le applicazioni di rete.1. Modelli di servizio a livello trasporto2. Paradigma client – server3. Paradigma peer-to-peer4. Reti di distribuzione dei contenuti• Conoscere i protocolli esaminando i protocolli di livello applicazione più famosi | <ul style="list-style-type: none">1. http2. FTP3. SMTP/POP3/IMAP4. DNS• Creazione di applicazioni di rete1. Socket API |
|---|---|
-

2.1 INTRODUZIONE

Lo sviluppo di una **applicazione di rete** è costituito dalla compilazione di programmi che vengono eseguiti su *sistemi periferici* e che comunicano tra loro tramite la rete. Un'app deve poter essere eseguita da macchine diverse, ma non è necessario fare dei programmi per chi lavora sotto lo stack di internet.

Alla base distinguiamo **2 infrastrutture**.

- **client-server**: un *server* è un host sempre attivo che fornisce dei servizi; un *client* è un host che comunica con il server inviando richieste.
Il *server* è sempre connesso quindi ha un indirizzo IP sempre attivo e sempre uguale.
Il *client* può essere online e offline, può collegarsi da reti differenti, quindi ha un indirizzo IP che cambia nel tempo
- **P2P**: è basata sul principio per il quale un host può assumere il ruolo sia di client che di server.
Non abbiamo i server sempre online ⇒ indirizzi IP dinamici ⇒ più difficile gestione.
Però abbiamo un grande vantaggio in termini di “auto-scalabilità” (ogni nuovo dispositivo fornisce potenza di calcolo al sistema ⇒ + efficiente) e di sicurezza (essendo le app distribuite + difficili da attaccare).

2.1.1 Processi

Un **processo** è un programma che lavora all'interno di un host.

Ci sono *processi client* (processo che da inizio alla comunicazione) e *processi server* (processo che aspetta di essere contattato).

Processi in host diversi comunicano scambiandosi messaggi, se invece fanno parte dello stesso host viene effettuata una comunicazione inter-processo.

2.1.2 Socket

I processi mandano/ricevono messaggi dai **socket**, sono simili a delle porte nello stack tra il livello applicazione e quelli sottostanti.

Ogni porta socket ha un suo identificatore costituito da un IP a 32 bit e da un numero di porta.

2.1.3 Servizi di trasporto disponibili per le applicazioni

Ogni applicazione ha bisogno di servizi a livello trasporto affinché resti efficiente.

- **Trasferimento dati affidabile:** alcune app possono sopportare che non tutti i pacchetti arrivino (tipo streaming audio/video) mentre ad altre va bene solo se il 100% dei pacchetti arriva (trasmissioni finanziarie)
- **Throughput:** alcune app hanno bisogno di un minimo di portata per funzionare; altre app (dette elastiche) prendono quello che c'è facendoselo bastare
- **Sicurezza:** alcune app hanno bisogno di un trasporto sicuro e crittografato e protezione da attacchi esterni.
- **Timing:** alcune app hanno bisogno di una bassa latenza (videogame o videoconferenze)

Durante lo sviluppo bisogna tener conto di queste necessità e quindi utilizzare il protocollo di livello trasporto adeguato:

- **TCP:** trasporto affidabile, controllo del flusso e congestione, ma non da garanzie su tempismo, portata e sicurezza
- **UDP:** servizio di trasferimento non affidabile e senza garanzia.

2.2 WEB E HTTP

Una pagina web è formata da **oggetti**, gli oggetti possono essere file HTML, immagini JPEG, applicazioni JAVA, file audio/video...

Ogni oggetto è identificato da un URL (Uniform Resource Locator), che contiene il nome dell'host e il percorso dell'oggetto.

Il servizio web è implementato col **protocollo HTTP**, basato sull'architettura *client-server*.

Il client è il browser che richiede la pagina web e riceve gli oggetti.

Il server è un Web server che invia al client gli oggetti con protocollo HTTP

HTTP è **stateless** quindi non mantiene alcuna informazione riguardo le richieste passate, e utilizza esclusivamente TCP.

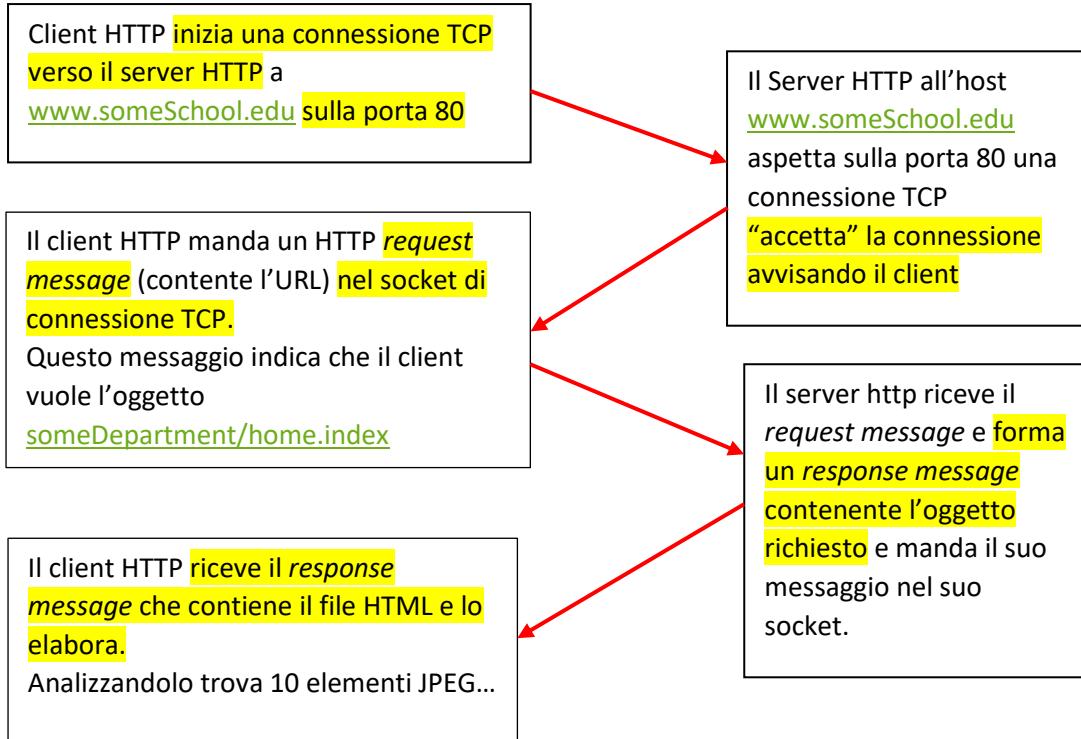
Ci sono 2 tipi di connessioni HTTP:

- **Non persistente**
Per ogni connessione instaurata viene inviato un solo oggetto
Quindi per inviare n oggetti serviranno n connessioni
- **Persistente**
Con una singola connessione possono essere inviati molti oggetti

2.2.1 HTTP non persistente

Esempio di come HTTP non persistente lavora:

tempo



Analizziamo ora il **tempo di risposta (RTT)**, ovvero il tempo impiegato per fare: **Client-Server-Client**

Vengono utilizzati, un RTT per fare 1. 2. 3. e un RTT per fare 3 4 5, in più ci aggiungiamo il tempo necessario al trasferimento del file

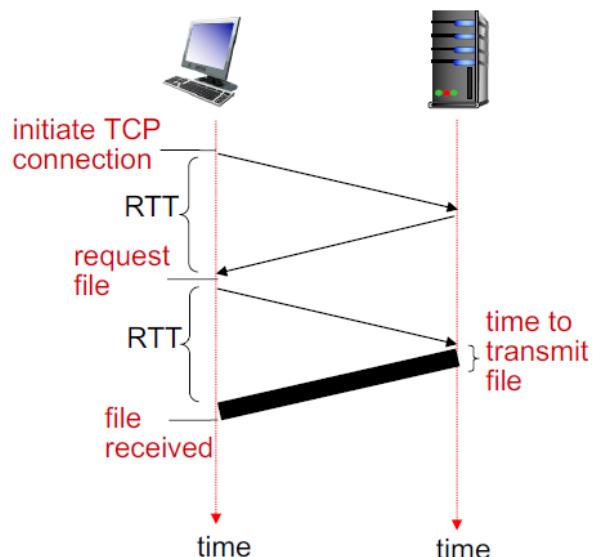
⇒ **2 RTT + tempo trasmissione file**

Fare tutto questo per ogni oggetto sovraccaricherebbe il Sistema operativo, l'unica alternativa è il *pipeling*, ovvero aprire connessioni parallele per essere più efficienti.

2.2.2 HTTP persistente

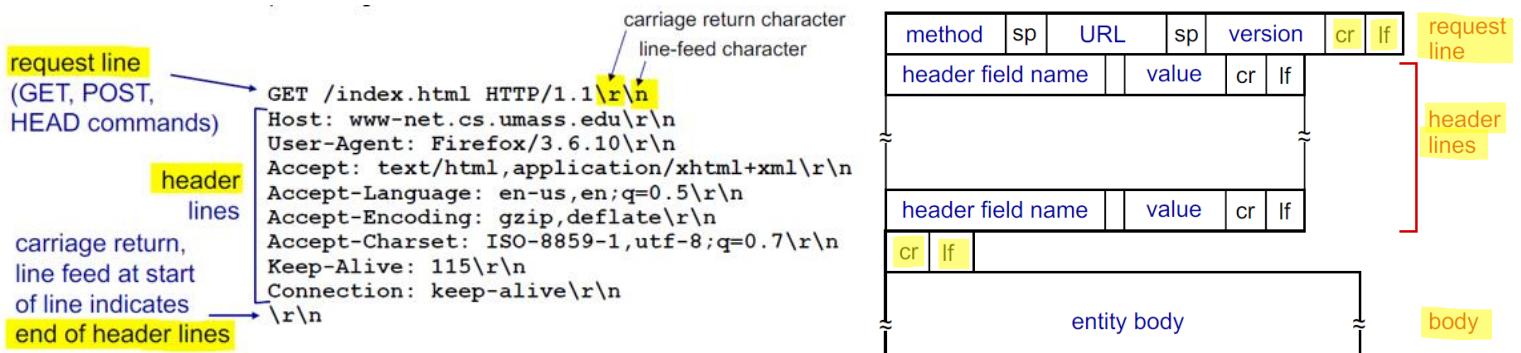
In questo caso il server lascia aperta la connessione dopo aver mandato la risposta, il client manda richieste appena incontra degli oggetti che non possiede.

Nel caso ottimo si avrà $RTT = 1$ per ottenere tutti gli n oggetti



2.2.3 HTTP request message

Vediamo la sintassi di un HTTP request message:



Nella request line:

- il campo *method* indica quale comando scegliamo tra GET, POST, HEAD, URL, PUT, DELETE.
- Il campo *URL* indica l'URL richiesto
- Il campo *version* indica la versione dell'HTTP

Ogni riga è interrotta da \r \n (ritorno a capo e nuova linea)

La riga successiva a quella della richiesta apre l'area dell'intestazione (header line) che è chiusa da una riga contenente i soli caratteri \r \n.

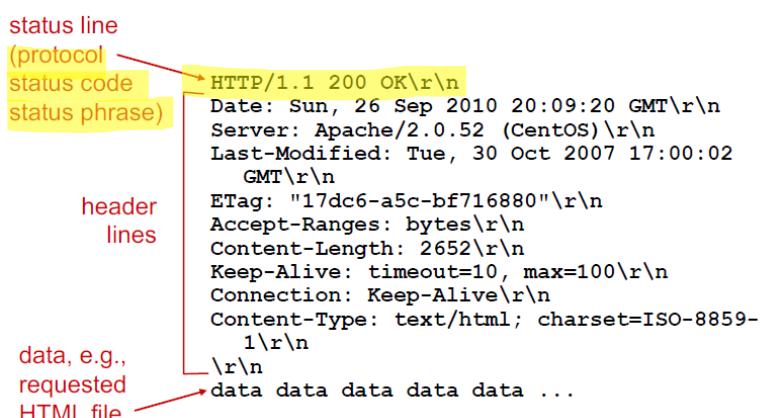
Inizia poi l'entity body, utilizzato in alcuni casi particolari.

- **Metodo POST:** si usa quando il client compila un form (come quello del motore di ricerca), l'utente sta ancora richiedendo una pagina web ma i contenuti dipendono dall'input che l'utente ha messo nel form.
- **Metodo URL:** usa a sua volta il metodo GET: l'input è caricato nel campo URL della prima riga (request line).

Per quanto riguarda le *versioni*, distinguiamo:

- **HTTP/1.0:** include i metodi **GET, POST, HEAD** → richiede al server di lasciare l'oggetto richiesto senza risposta
- **HTTP/1.1:** include i metodi **GET, POST, HEAD, PUT** → carica I file dell'entity body in base al percorso specificato nel campo URL, **DELETE** → cancella i file specificati nel campo URL

2.2.4 HTTP response message



Analizziamo la *status line* (1° linea)

- 200 OK** → successo, l'oggetto richiesto si trova in fondo alla risposta
- 301 Moved permanently** → l'oggetto è stato spostato, in fondo è indicata la nuova posizione
- 400 Bad Request** → il server non ha capito la richiesta
- 404 Not Found** → l'oggetto cercato non è su questo server
- 505 HTTP Version Not Supported**

2.2.5 I Cookies

I server HTTP sono *stateless*, quindi la progettazione è più semplice e possiamo raggiungere prestazioni molto alte. Però i server necessitano di **tener traccia degli utenti** per poter offrire contenuti personalizzati.

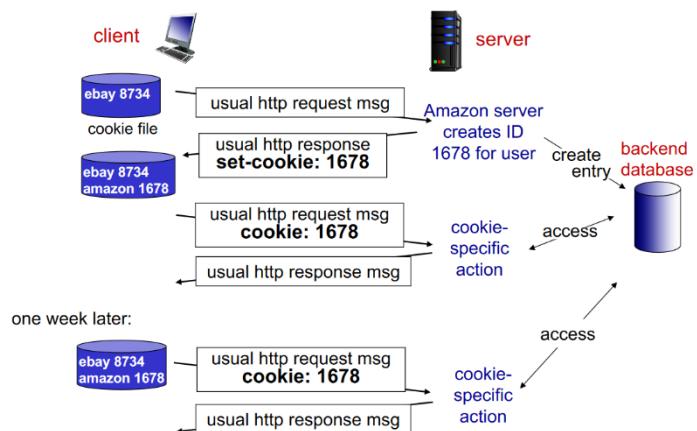
Molti siti web utilizzano i cookies a quattro componenti:

- Una riga di intestazione nel response message di HTTP
- Una riga di intestazione nel request message di HTTP
- Un cookie file mantenuto sul sistema dell'utente e gestito dal browser dell'utente
- Un database sul sito

Vediamo un esempio:

Susan è già registrata sue Ebay (ha un cookie per l'accesso), oggi però accede ad Amazon per la prima volta. Il server di Amazon crea un nuovo ID per Susan (1678) e crea un accesso nel server. Il nuovo ID è comunicato a Susan con l'istruzione **set cookie: 1678**, nell'intestazione del messaggio di risposta.

Il browser di Susan che lo riceve aggiunge una riga al file dei cookie che gestisce, contenente il nome del server di Amazon e il numero comunicato con *set cookie*. Ogni nuova richiesta alla pagina di Amazon conterrà nell'intestazione del messaggio di richiesta **Cookie: 1678** → in questo modo Amazon può tenere traccia dell'attività di Susan sul sito. Se Susan fornisce dati personali, Amazon aggiorna il suo database mettendoli in corrispondenza del numero 1678.



Tramite i cookie Amazon può tenere una lista dei prodotti messi nel carrello affinché un utente possa pagare tutto insieme alla fine, può fornire il “one-click shopping” grazie ai dati registrati nel database, può usare le informazioni raccolte per indagini di mercato e per collezionare *feedback* sui suoi servizi.

2.2.6 Web caches (proxy server)

Obiettivo: → soddisfare le richieste dei client senza consultare il server di origine.

L'utente imposta il browser in modo che invii tutte le richieste alla cache,

- Se l'oggetto è in cache viene rimandato al client
- Se non c'è la cache richiede l'oggetto al server, lo ottiene e lo ritorna al client
La cache, quindi, agisce sia da client (chiede al web server) che da server (fornisce al browser)

Vantaggi? → si reduce l'RTT, si reduce il traffic di banda e provider “deboli” possono lavorare bene (tipo P2P)

GET condizionale

Obiettivo: non mandare oggetti al proxy se questo ne ha una versione più recente (ritardi diminuiti diminuisce il traffico sul collegamento)

- Cache: specifica la data dell'ultima modifica dell'oggetto compilando il *request message*
If-modified-since: <date>

- Server: il *response message* non contiene alcun oggetto se l'oggetto in cache non è da aggiornare

HTTP/1.0 304 Not Modified

2.3 ELECTRONIC MAIL

Tre componenti principali:

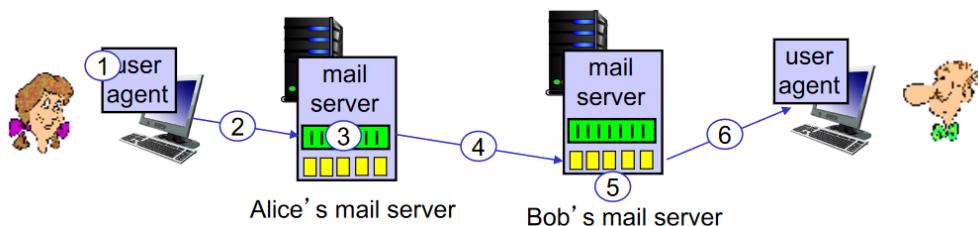
- **User agents** → aka “mail reader”, è l’interfaccia che l’utente utilizza per leggere/scrivere le mail
- **Mail servers** → memorizzano le mail in entrata/uscita, costituito da 3 parti:
 1. **Mailbox** → contiene i messaggi in arrivo
 2. **Coda dei messaggi** → coda dei messaggi in uscita
 3. **SMTP** → mette in comunicazione diversi mail server
- **SMTP (Simple Mail Transfer Protocol)**

2.3.1 SMTP

Per prima cosa effettua un three-way-handshake e usa solamente TCP per trasferire messaggi email da client a server (porta 25).

Esempio:

- 1) Alice usa il suo agent (Outlook) per scrivere un'email a Bob
- 2) la invia al suo mail server. Il messaggio entra nella coda dei messaggi in uscita
- 3) il mail server di Alice apre una connessione TCP con il mail server di Bob
- 4) il client STMP di Alice manda la mail di Alice con connessione TCP
- 5) il server di Bob deposita la mail nella sua mailbox.
- 6) Tramite il suo agent Bob scarica la mail e la legge.



2.3.2 Protocolli di accesso mail

- **POP (Post Office Protocol)** → è un protocollo *stateless* che riguarda la gestione delle autorizzazioni e del download.

Durante l’autorizzazione il client dichiara user e password e il server risponde con +OK oppure -ERR
Nella fase di transazione, il client può scegliere tra le seguenti operazioni:

- **list**: si ottiene una lista ordinata degli identificativi dei messaggi nel server
- **retr x**: ritorna il messaggio nella lista che occupa la posizione *x*
- **dele x**: cancella il messaggio in posizione *x*
- **quit**

Se POP prevede il messaggio **delete**, allora è di tipo *download & delete* ⇒ una volta eliminata la mail non è recuperabile

Se invece non prevede il messaggio **delete**, è di tipo *download & keep*

- **IMAP (Internet Mail Access Protocol)** → Fornisce strumenti di:
 - manipolazione dei messaggi memorizzati nei server
 - organizzazione in cartelle

è un protocollo che mantiene lo stato al cambio di sessione, in particolare mantiene la corrispondenza tra l’ID dei messaggi e le cartelle in cui sono conservati.

2.4 DNS

Domain Name System

Gli host di internet e i router possono essere identificati

- Indirizzo IP (32 bit) usati per indirizzare i datagrammi
- “nome” (ES: www.amazon.com)

Le persone ricordano meglio i nomi, mentre i router lavorano con gli IP essendo strutturati gerarchicamente.

Per conciliare i due approcci il DNS traduce i nomi degli host nei loro indirizzi IP.

È un **database distribuito** e implementato in una gerarchia di server DNS

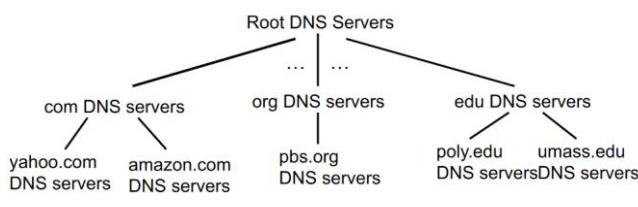
Ed è anche un **protocollo** di livello applicazione basato su UDP (porta 53) che consente agli host di interrogare il database. I suoi servizi principali sono:

- **Host aliasing:** un host dal nome complicato può avere uno o più sinonimi facili da ricordare
- **Mail server aliasing:** ad indirizzo email canonico può essere associato un sinonimo più semplice
- **Distribuzione del carico di rete:** è possibile distribuire il carico tra server replicati

Perché non centralizzare tutto il servizio DNS?

- Perché rappresenterebbe un *single point of failure*
 - Dovrebbe sopportare un traffico troppo elevato
 - Di difficile e costosa manutenzione
 - Gli host lontani sarebbero lenti da raggiungere
- ⇒ NON SAREBBE SCALABILE

Proprio per questioni di scalabilità i server DNS sono organizzati in modo gerarchico.



Distinguiamo 3 classi di server DNS

1. **Root server:** in cima alla gerarchia, ne esistono 13 in tutto l'internet
2. **TLD (Top Level Domain) server:** si occupano dei domini di primo livello (.com, .org, .net, ...) e dei domini dei vari paesi (.it, .fr, .uk, ...)
3. **Server DNS autoritativi:** ogni organizzazione che ha host accessibili su internet ha dei server DNS che fanno accedere gli utenti a questi server.
4. **DNS locali:** (non fa proprio parte della gerarchia) Ogni ISP ne ha uno (Default Name Server), che funziona come un proxy server.

2.4.1 Risoluzione di un nome

Facciamo due esempi:

L'host a cis.poly.edu vuole l'indirizzo IP per gaia.cs.umass.edu

2.4.1.1 QUERY ITERATA

Il server locale contatta un server nella gerarchia,

se il server contattato non ha il mapping del nome ⇒ restituisce al local il nome del server da contattare

“io non lo conosco questo nome, ma prova a chiedere a questo server, lui magari lo conosce”

2.4.1.2 QUERY RICORSIVA

La risoluzione del nome spetta al server contattato, non del local

“non conosco questo nome, aspetta che te lo cerco io”

2.4.2 Caching

Il DNS utilizza il caching per migliorare le prestazioni generali: dopo aver ricevuto una risposta con il mapping “hostname → IP”, la mette in cache. In una richiesta per quell’hostname il server con cache può fornire direttamente l’IP corrispondente pur non essendo il server autoritativo di quell’IP.

Dato che le associazioni hostname-IP non sono permanenti, i DNS invalidano gli indirizzi tenuti in cache per più di 2 giorni (*out-of-date*).

2.4.3 Resource record

I server che implementano il database distribuito dei DNS memorizzano i **RR** (Resource Records)

RR format: (name, value, type, TTL)

→ TTL = Time To Leave

→ Nome & Value = Cambiano in funzione del tipo

→ Type :

○ Type = A

- name = hostname
- Value = il suo IP

Es. (foo.com, 145.37.93.126, A)

○ Type = NS

- Name = dominio
- Value = hostname del DNS autoritativo che sa come ottenere gli IP del dominio

Es. (foo.com, dns.foo.com, NS)

○ Type = CNAME

- Value = rappresenta il nome canonico dell’host per il sinonimo name

Es. (foo.com, relay1.bar.foo.com, CNAME)

○ Type = MX

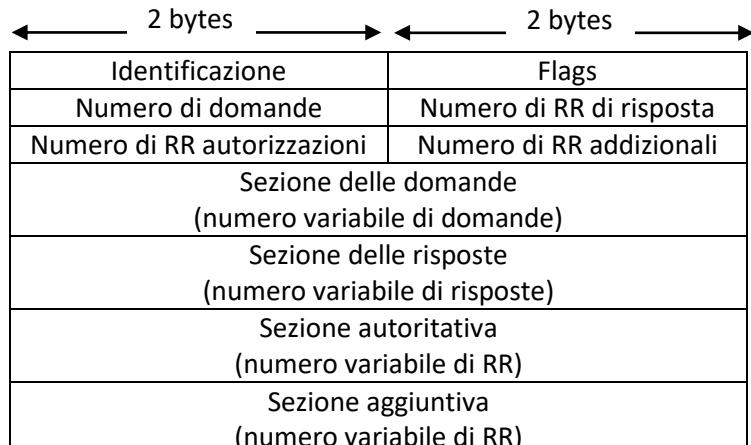
- Value = nome canonico del mailserver associato con name

Es. (foo.com, mail.bar.foo.com, MX)

2.4.4 Formato dei messaggi DNS

I messaggi di query e quelli di reply hanno lo stesso formato

- L’*identificatore* è un numero di 16 bit (2 bytes), una sorta di “ID della transazione”
Ad una query con un certo identificatore corrisponderà univocamente una risposta con lo stesso ID
- *Flags*: si tratta di una query o una risposta? Si desidera ricorsione o iterazione? La ricorsione è disponibile? La risposta è autoritativa?
- 4 campi “Numero di ...”: numero di occorrenze delle 4 sessioni dopo
- *Sezione delle domande*: contiene informazioni sulle richieste che stanno per essere effettuate. Include un campo “nome richiesto” e il type della domanda
- *Sezione delle risposte*: contiene i RR inviati dal DNS server contattato. Una singola risposta può avere più RR, dato che ad un hostname possono corrispondere più IP
- *Sezione autoritativa*: contiene i record dei server autoritativi
- *Sezione aggiuntiva*: contiene una serie di informazioni aggiuntive



2.4.5 Attaccare un DNS

- Attacchi DDoS (Distributed Denial of Service) → bombardare di traffico:
 - Root server: poco dannoso, gerarchia e caching permettendo di distribuire il traffico in modo efficiente
 - TLD server: decisamente più pericoloso
- Man in the middle → intercetta le query.
- DNS poisoning → L'attaccante invia risposte fasulle ai server DNS
- Sfruttare la struttura DNS per provocare DDoS → inviare molte query sotto il nome del bersaglio, per poi fargli arrivare tutte le risposte e sommergerlo.

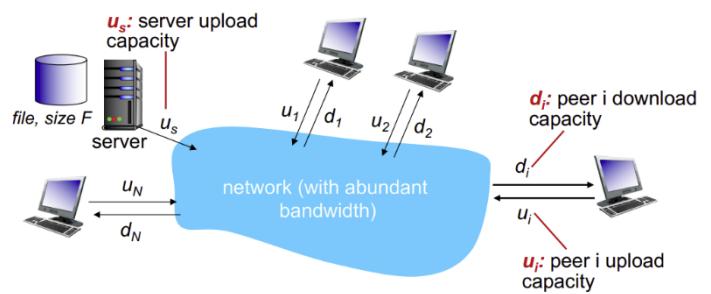
Conclusioni: il DNS è molto **robusto** e nessuno è stato ancora in grado di interromperne il servizio.

2.5 CLIENT-SERVER Vs P2P

Quanto tempo impieghiamo per distribuire un file di dimensione F da un server a N peers sapendo che la capacità di download e upload è limitata?

CLIENT-SERVER:

- Lato server:
 - tempo per l'invio di una copia F/u_s
 - tempo per mandare N copie NF/u_s
- Lato client: ogni client deve fare il download del file
 - d_{min} = minimo tasso di download del client
 - tempo minimo di download del client è F/d_{min}



Il tempo per distribuire il file F a N client è: $D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$

La prima componente cresce linearmente al numero N di host.

P2P

- Lato server: bisogna trasmettere come minimo una copia
 - tempo per mandare una copia: F/u_s
- Lato client: ogni client deve fare il download di una copia:
 - tempo minimo di download del client è: F/d_{min}
- Insieme di client: essendo un insieme devono scaricare NF bits
 - Massimo tasso di upload (upload che tutti i componenti possono effettuare) $u_s + \sum u_i$

Otteniamo un tempo totale di: $D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$

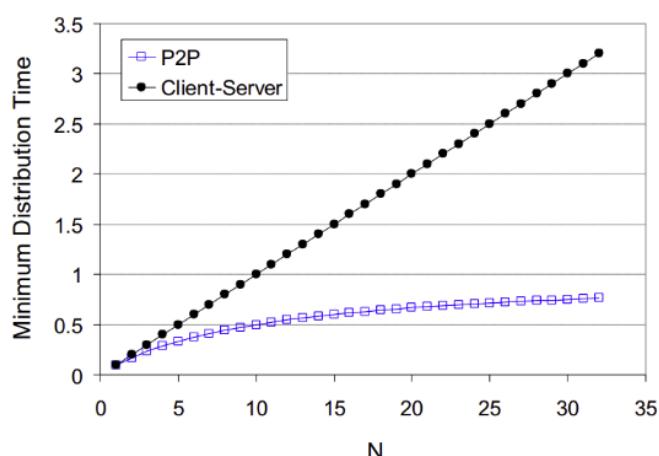
CLIENT-SERVER vs P2P

client upload rate = u

$F/u = 1$ ora

$u_s = 10u$

$d_{min} \geq u_s$



2.6 SERVIZI CON ARCHITETTURA P2P

2.6.1 Distribuzione di file: BitTorrent

Definiamo **torrent** un gruppo di peers che si scambiano *chunks* (frazioni da 256Kb di file)

Un sistema Torrent è dotato di un tracker, che mantiene e aggiorna una lista dei peers che stanno prendendo parte al torrent.

Un client che si aggiunge al torrent (Alice) sarà inizialmente sprovvisto di chunks; ottiene la lista dal tracker, entra in connessione con un sottoinsieme di peers della lista (quelli a lei “vicini”) e inizia a scambiare file.

Mentre effettua il download, mette a disposizione in upload i chunks appena ottenuti.

Dal momento che alcuni peer potrebbero abbandonare il torrent una volta ottenuto il file completo, i peers da cui un client ottiene chunks variano nel tempo.

Richiesta dei chunk: In un dato istante peer diversi hanno chunk diversi dello stesso file: periodicamente Alice ottiene da ogni peer la lista dei chunk che possiede e richiede i pezzi che le mancano partendo dal più raro al più comune.

Invio dei chunk: Alice manda i chunk ai top 4 peer che gli mandano chunk al tasso più alto e “strozza” gli altri. Ogni 10 secondi questa top 4 è aggiornata, e ogni 30 sec è scelto random un altro peer che si unisce alla top 4.

2.6.2 Streaming video

Il traffico video è il maggior consumatore di banda. Come rendiamo scalabile un sistema streaming? E soprattutto come possiamo gestire le differenti prestazioni/capacità degli host?

Soluzione: ci serve un’infrastruttura di livello applicazione distribuita e non centralizzata.

Video: sequenza di immagini mostrate ad un tasso costante (24 fps)

Immagine digitale: array di pixels, ognuno dei quali rappresentato da bit

Codifica con ridondanza: con la ridondanza interna, anziché inviare N bit che hanno lo stesso valore (quindi lo stesso colore) inviamo solamente 2 valori, quello del colore e il numero di volte in cui il colore è ripetuto. Nella ridondanza tra immagini dato il frame i anziché inviare il frame $i + 1$ completo, inviamo solamente le differenze tra i e $i + 1$ (che normalmente fra due frame consecutivi sono poche)

CBR (Constant Bit Rate): il tasso di codifica è fisso;

VBR (Variable Bit Rate): il tasso di codifica cambia al variare della quantità di dati necessari per la codifica

DASH: Dynamic Adaptive Streaming over HTTP

→ **server:**

- divide il file in tanti chunks (ognuno dei quali è memorizzato, codificato a velocità differenti)
- *manifest file*: costruito dal server, assegna un URL per ogni chunk

→ **client:**

- misura periodicamente la banda disponibile tra client e server
 - in base a questo sceglie il massimo tasso di codifica possibile.
 - può scegliere tra codifiche differenti durante lo streaming, in base alla banda
- consulta il manifesto, richiede un chunk alla volta.

- ➔ “intelligenza” del client → è il client a decidere:
 - quando richiedere il chunk per evitare *overflow* o *starvation*
 - a che tasso di codifica chiederlo
 - a che server richiederlo

2.6.3 CDN (Content Distribution Networks)

Come facciamo lo stream di un contenuto a centinaia di migliaia di utenti simultanei?

Abbiamo 2 modalità:

- **Enter deep:** si entra profondamente nelle reti di accesso dei provider installando gruppi di server (cluster) nei provider di accesso sparsi in tutto il mondo.
L'obiettivo è essere vicini agli utenti in modo da diminuire il ritardo percepito dall'utente.
- **Bring home:** si porta a casa l'ISP costruendo grandi cluster in pochi punti chiave e interconnetterli usando una rete privata ad alta velocità. Invece di entrare negli ISP di accesso queste CDN pongono ogni cluster in un luogo vicino ai POP.

CDN memorizza copie del contenuto ai suoi nodi; il client che necessita di una copia la richiede al CDN più vicino. Può scegliere diverse copie se la rete è congestionata

Esempio di **accesso ai contenuti** di un CDN:

- il video richiesto da Bob è `http://netcinem.com/6Y7B23V`, che è memorizzato in `http://KingCDN.com/NetC6Y7B23V`
- Il DNS autoritativo di netcinema decide su quale server della CDN verrà scaricata la copia e lo comunica al local server di Bob.

2.7 PROGRAMMAZIONE SOCKET

Obiettivo: imparare a costruire applicazioni client/server che comunicano con i socket

Socket: è la porta tra il livello applicazione e il livello trasporto

Essendoci due protocolli di trasporto ci sono anche due tipologie di socket:

- ➔ UDP
- ➔ TCP

2.7.1 Programmazione con UDP

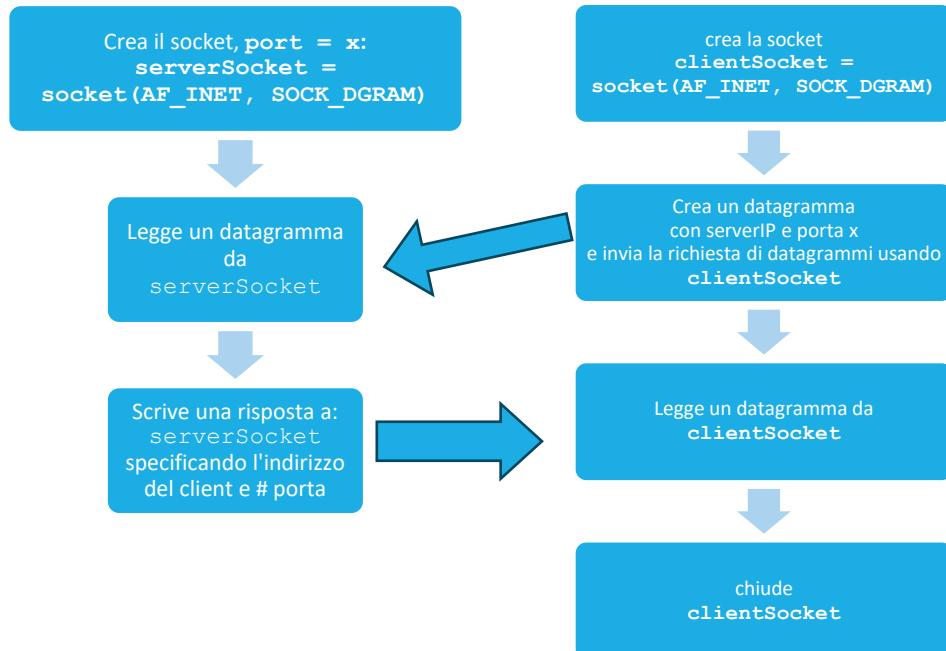
UDP: non c'è “connessione” tra client e server

- ➔ Nessun handshaking prima di scambiarsi i messaggi
- ➔ Il mittente allega solo l'IP e il numero di porta mirata per ogni pacchetto
- ➔ Il destinatario estrae l'IP del mittente e il numero di porta dal pacchetto

I dati trasmessi da UDP possono venire persi o ricevuti quando siamo fuori servizio.

Dal punto di vista delle applicazioni UDP fornisce un inaffidabile trasferimento di datagrammi tra C-S.

2.7.1.1 Interazione client-server con UDP:



Python UDP client:



```

1  from socket import *  #importiamo il modulo socket
2
3  serverName = "130.136.5.36" #assegnamo alla variabile di tipo stringa serverName l'IP dell'host
4  serverPort = 12000 #assegna alla variabile intera serverPort il numero 12000
5
6  clientSocket = socket(AF_INET, SOCK_DGRAM) #crea il socket lato client, memorizzandone il riferimento
7          # il primo parametro indica la famiglia di indirizzi (AF_INET = IPv4),
8          # il secondo dice che il socket è di tipo SOCK_DGRAM (UDP e non TCP).
9  message = raw_input('frase in minuscolo') #L'utente può digitare da tastiera una stringa assegnata a message
10 clientSocket.sendto(message.encode(), (serverName, serverPort)) # sendto() attacca l'indirizzo di destinazione
11          # (serverName, serverPort) al messaggio e invia il pacchetto nella clientSocket
12 modifiedMessage, serverAddress = clientSocket.recvfrom(2048) #quando un pacchetto arriva da internet
13          # al socket client, i dati contenuti vengono assegnati a modifiedMessage
14          # mentre l'indirizzo sorgente del pacchetto è messo in serverAddress
15          # (che contiene sia l'IP che il numero di porta del server)
16          # recvfrom prende in input la grandezza del buffer che è 2048
17 print(modifiedMessage.decode()) # mostra modified message sul display dell'utente;
18 clientSocket.close() # chiude il socket, il processo termina.
  
```

Python UDP server:

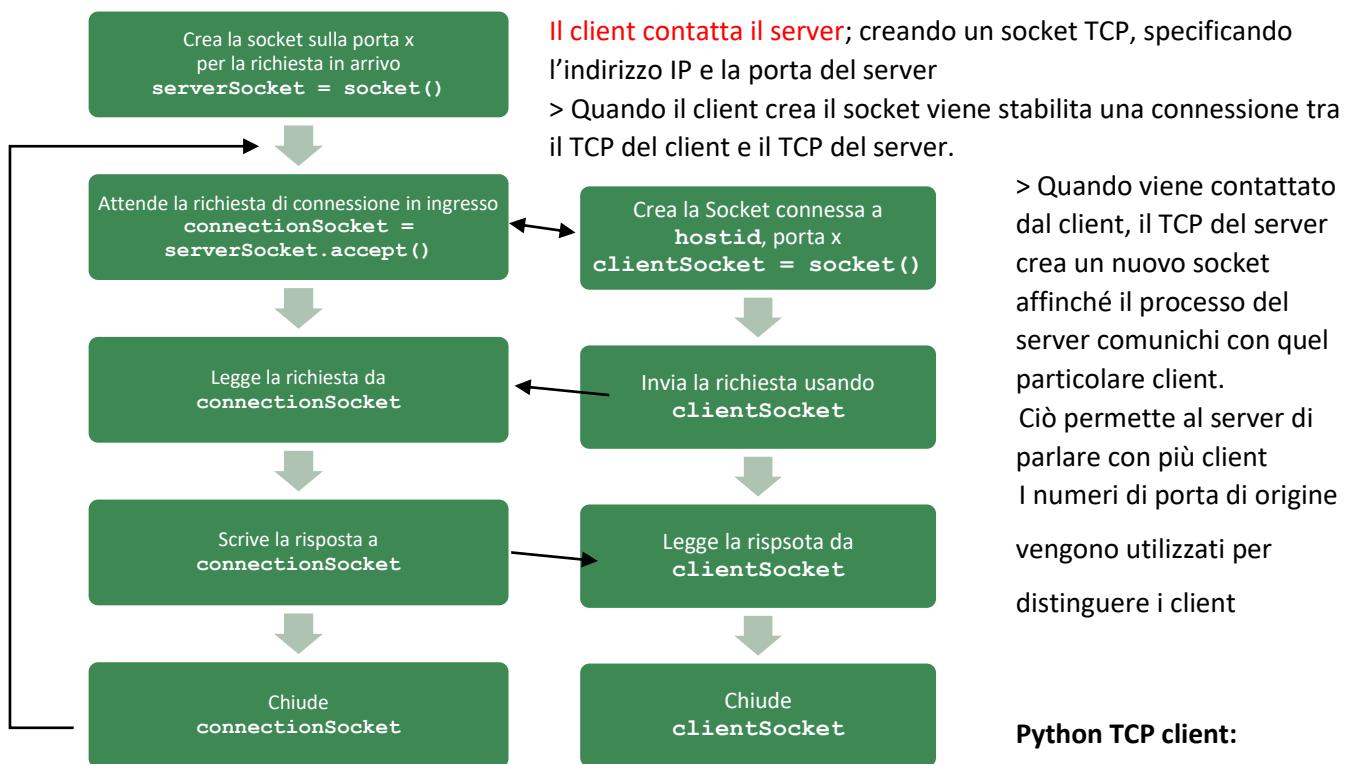


```

1  from socket import * # Import socket module
2  serverPort = 12001
3  serverSocket = socket(AF_INET, SOCK_DGRAM)
4  serverSocket.bind(("0.0.0.0", serverPort)) # assegna al socket lato server il numero di porta
5      # => quando qualcuno manderà un pacchetto a questa porta lo manderà a questo socket
6  print('The server is ready to receive')
7
8  while True:
9      message, clientAddress = serverSocket.recvfrom(2048) # quando un pacchetto arriva al
10                 # socket lato server i dati sono messi in message
11                 # e l'indirizzo sorgente a clientAddress (IP e porta del client)
12  print("client address:", clientAddress)
13  modifiedMessage = message.decode().upper() # uppercase del messaggio inviato al client
14  print(modifiedMessage)
15  serverSocket.sendto(modifiedMessage.encode(), clientAddress) # attacca l'indirizzo del client (IP e porta)
16                 # al messaggio in maiuscolo e invia il pacchetto al lato server
  
```

2.7.1.2 Interazione Client-Server con TCP

Il client deve contattare il server; i processi del server devono essere in esecuzione e il server deve aver creato la porta contattata dal client.



```

1 from socket import *
2 serverName = "130.136.5.36"
3 serverPort = 12001
4 clientSocket = socket(AF_INET, SOCK_STREAM) # creazione socket lato client; SOCK_STREAM indica TCP
5 clientSocket.connect((serverName, serverPort)) # connect inizializza una connessione TCP tra client e Server
6
7 message = raw_input('Inserire frase in minuscolo: ')
8 clientSocket.sendto(message.encode(), (serverName, serverPort)) # non serve allegare il nome e la porta
9 modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
10 print(modifiedMessage.decode())
11 clientSocket.close()

```

Python TCP server:



```

1 from socket import *
2 serverPort = 12001
3 tcpServerSocket = socket(AF_INET, SOCK_STREAM)
4 tcpServerSocket.bind("", serverPort) # in TCP, serverSocket è una "welcome door"
5 tcpServerSocket.listen(1) #il server si mette in ascolto e attende richieste. Il parametro indica il max num
6 # di connessioni da tenere in coda
7 print('The server is ready to receive')
8
9 while True:
10     tcpConnectedClientSocket, addr = tcpServerSocket.accept() #accept() blocca temporaneamente l'esecuzione
11     # del programma finché il client bussa alla welcomedoор. Quando l'esecuzione riprende, viene creato
12     # un nuovo socket nel server (connectionSocket) dedicato allo specifico client che ha bussato. Handshake completo.
13     print('Received a client connection from:', addr)
14
15     message = tcpConnectedClientSocket.recv(2048).decode() # non resta che scambiarsi i byte
16     modifiedMessage = message.upper()
17     print(modifiedMessage)
18     # sends the modified message to the client. Note the send() vs. sendto()
19     tcpConnectedClientSocket.send(modifiedMessage.encode())
20
21     tcpConnectedClientSocket.close()

```

3 LIVELLO DI TRASPORTO

Obiettivi:

- Comprendere i principi alla base dei servizi di livello trasporto
 - o Multiplexing e demultiplexing
 - o Trasferimento affidabile dei dati
 - o Controllo del flusso
 - o Controllo delle congestioni
- Conoscere i protocolli di livello trasporto di Internet
 - o UDP: trasporto connectionless
 - o TCP: trasporto affidabile connection-oriented
 - o TCP congestion control

3.1 SERVIZI DI LIVELLO TRASPORTO

A livello trasporto: ‘Pacchetto’ \Rightarrow ‘segmento’

Il protocollo a livello rete è IP che, che fornisce comunicazione logica tra gli host.

Il suo modello prende il nome di best-effort, in quanto IP fa di tutto per consegnare i segmenti fra host comunicanti, **ma non offre garanzie**, poiché non assicura l’integrità dei pacchetti o la loro consegna in ordine sequenziale. Per questo IP è un servizio non affidabile.

I protocolli TCP e UDP lavorano in modo tale da estendere il servizio di consegna host-to-host a process-to-process.

Questo passaggio prende il nome di **multiplexing e demultiplexing a livello trasporto**.

3.2 MULTIPLEXING E DEMULTIPLEXING

Il compito del livello trasporto è quello di “consegnare i dati dei segmenti al processo applicativo appropriato”.

Dato che un processo può gestire più socket \Rightarrow il livello trasporto non manda i dati direttamente al processo ma ad una socket intermedia.

→ Multiplexing al mittente:

Gestisce i dati da vari socket e gli aggiunge l’intestazione di trasporto

Anna effettua un’operazione di multiplexing quando raccoglie le lettere dai mittenti e le imbuca

→ Demultiplexing al destinatario:

usa le intestazioni per distribuire i segmenti che gli arrivano ai rispettivi socket

Quando Andrea riceve le lettere dal postino, effettua un’operazione di demultiplexing leggendo il nome riportato sopra la busta e consegnando ciascuna missiva al rispettivo destinatario

Affinché multiplexing funzioni ha bisogno di:

- Identificatori unici per le socket \rightarrow **numero di porta d’origine**
- Campi che indichino la socket di destinazione nel segmento \rightarrow **numero di porta di destinazione**

I numeri di porta sono di 16 bit e vanno da 0 a 65535, quelli che vanno da 0 a 1023 sono chiamati numeri di porta noti (*well-known port number*) e sono riservati per essere usati da protocolli applicativi ben noti quali HTTP (porta 80) e FTP (porta 21).

Numero di porta d’origine	Numero di porta di destinazione
Altri campi dell’intestazione	
Dati dell’applicazione (messaggio)	

3.2.1 Mux e demux non orientati alla connessione

La prima cosa da fare per poter descrivere mux/demux del protocollo UDP è assegnare i numeri di porta alle socket UDP.

Lo facciamo con le funzioni viste nella parte di programmazione socket.

Esempio:

un processo nell'Host A, con porta UDP 19157, vuole inviare un blocco di dati applicativi a un processo con porta UDP 46428 nell'Host B.

- Il livello di trasporto di A crea un segmento che include: i dati applicativi, numero della porta di origine, e di destinazione, e altri valori...
- Il livello trasporto passa il segmento risultante al livello rete, che lo incapsula in un datagramma IP, ed effettua un tentativo best-effort (IP fa "del suo meglio" per consegnare i segmenti tra host comunicanti, ma non offre garanzie) di consegna del segmento all'Host di ricezione.
- Se il segmento arriva a B il suo livello trasporto lo esamina la porta di destinazione e lo consegna.

3.2.2 Mux e demux orientati alla connessione

C'è differenza tra una socket UDP e una socket TCP,

La socket TCP è identificata da 4 parametri:

1. IP di origine
2. N° porta di origine
3. IP di destinazione
4. N° porta di destinazione

Questi 4 valori vengono usati per dirigere (fare demultiplexing) il segmento verso la socket appropriata

In particolare, segmenti con stessa destinazione (IP e porta) ma diversa provenienza (anche stesso IP ma diversa porta o viceversa), vengono mandati a socket differenti.

Es.

- L'app server TCP ha un "socket di benvenuto", questo si pone in attesa di **richieste di connessione** da parte dei client TCP sulla porta 1200
- Il client TCP crea una socket e genera un segmento per stabilire la connessione con il "socket di benvenuto"
- Una **richiesta di connessione** è un segmento TCP con porta di destinazione → 1200 e con un bit di richiesta di connessione = 1.
- Il SO dell'Host quando riceve il segmento con richiesta di connessione, localizza il processo server in attesa di connessioni e il server crea una nuova connessione.
- Il livello trasporto sul server prende nota dei seguenti valori nella richiesta di connessione:
 1. N° porta di origine
 2. IP host di origine
 3. N° porta di destinazione
 4. Il proprio IP (del server)

La socket appena creata viene identificata con questi 4 valori

- Tutti i segmenti successivi che hanno gli stessi 4 valori verranno indirizzati a questa porta.

3.3 TRASPORTO NON ORIENTATO ALLA CONNESSIONE: UDP

UDP fa il minimo che un protocollo a livello trasporto deve fare, fornisce multiplexing e demultiplexing per la distribuzione dei pacchetti, fornisce un minimo controllo degli errori e non aggiunge nulla a IP.

Infatti quando si sceglie UDP il livello applicazione dialoga in modo quasi diretto con il livello rete (con IP)

UDP:

- Prende i messaggi dal processo applicativo ↓
- Aggiunge numero di porta d'origine/destinazione (per mux/demux)
- Aggiunge altri due piccoli campi
- Passa il segmento risultante al livello rete ↓↓
- Il livello rete incapsula il segmento in un datagramma IP
- Prova a consegnarlo in modo best-effort
 - Se arriva a destinazione UDP server usa numero di porta di destinazione per consegnare i dati.

In UDP **non esiste handshaking** tra mittente e destinatario a livello trasporto, per questo si dice che UDP *non è orientato alla connessione*.

❓ Perché si utilizza UDP se non è affidabile? Non si potrebbe usare solo TCP? ❓

- Molte applicazioni funzionano meglio con UDP perché:
- *Ha un controllo più preciso a livello applicazione su quando e quali dati sono inviati*
TCP invece ha dei meccanismi di controllo della congestione che ritarda l'invio di pacchetti
Inoltre TCP continua ad inviare pacchetti fino a quando non riceve un acknowledgement, incurante del tempo richiesto per un trasporto affidabile.
- *Non viene stabilita nessuna connessione*
TCP usa un handshake a tre vie prima di iniziare il trasferimento dei dati
UDP "spara" dati senza preliminari
⇒ UDP non introduce ulteriore ritardo nel formare una connessione
(è il motivo per cui DNS utilizza UDP, perché con TCP risulterebbe più lento)
- *Non c'è stato di connessione*
UDP non conserva la connessione e non tiene traccia di congestione o acknowledgement
Per questo motivo un server dedicato ad un'applicazione può supportare molti più client usando UDP che TCP .
- *Meno spazio per l'intestazione.*
TCP mette **20byte** (!!!) di intestazione, UDP solo 8.

3.3.1 Struttura del segmento UDP

- *I numeri di porta* consentono il corretto indirizzamento (demultiplexing)
- *Lunghezza* specifica il numero di byte del segmento UDP
Necessario perché la grandezza del campo Dati può essere diversa tra un segmento e il successivo
- *Checksum* serve per il controllo degli errori

Numero di porta di origine	Numero di porta di destinazione
Lunghezza	Checksum
Dati dell'applicazione (messaggio)	

3.3.2 Checksum UDP

Serve per il rilevamento degli errori

Lato mittente UDP effettua **il complemento a 1 della somma di tutte le parole a 16 bit nel segmento.**

Es.

Le parole a 16 bit sono:
0110011001100000
0101010101010101
100011100001100

La somma delle prime due è:
0110011001100000
0101010101010101

1011101101101011

Sommendo la terza:
1011101110110101
1000111100001100

0100101011000011
0100101011000010

Il complemento a 1 si ottiene invertendo 1 e 0, quindi il checksum sarà:

1011010100111101

in ricezione: si sommano le 3 parole iniziali e il checksum, se non ci sono errori nel pacchetto il risultato sarà

1111111111111111

Se un bit vale 0 \Rightarrow c'è almeno un errore nel pacchetto.

3.4 PRINCIPI DI TRASFERIMENTO AFFIDABILE DEI DATI

Il problema del trasferimento dati affidabile non si presenta solo a livello trasporto, ma anche a livello applicazione e di collegamento.

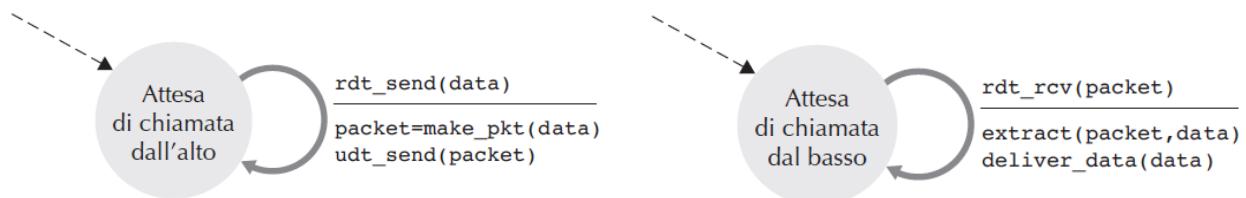
Con un canale affidabile nessun bit dei dati trasferiti è corrotto o va perduto e anche l'ordine d'invio viene rispettato.

Il compito di un **protocollo di trasferimento dati affidabile** è l'implementazione di questa astrazione del servizio. L'antagonista che rende difficile il trasferimento affidabile è il livello inferiore al protocollo.

3.4.1 Canale perfettamente affidabile: rdt1.0

Caso più semplice, il canale sottostante è completamente affidabile

\Rightarrow chiamiamo rdt1.0 introducendo la **macchina a stati finiti** (FSM, *Finite-State-Machine*) per mittente e destinatario rdt1.0



rdt_send(data) si accettano i dati

packet=make_pkt(data) crea un pacchetto di dati

udt_send(packet) invia il pacchetto sul canale

rdt_recv(packet) si raccolgono i pacchetti

extract(packet,data) rimuove i dati dai pacchetti

deliver_data(data) passa i pacchetti al livello superiore

3.4.2 Canale con errori sui bit: rdt2.0

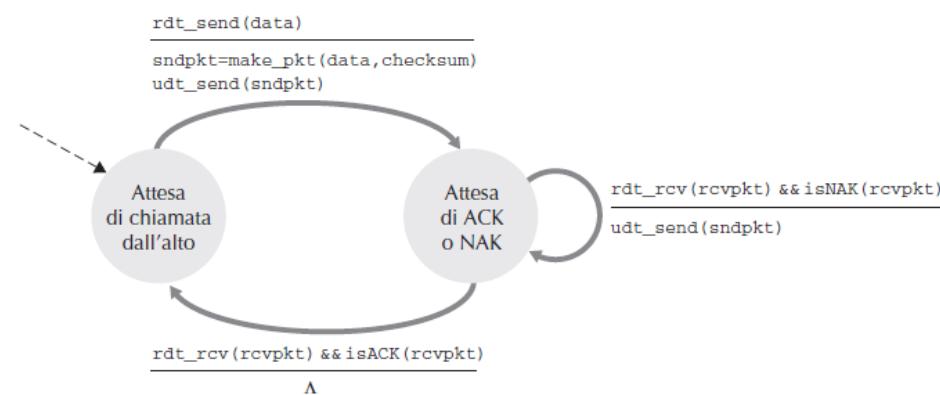
Questi errori si verificano nei componenti fisici delle reti quando il pacchetto viene trasmesso, propagato o inserito nei buffer.

Questo protocollo di dettatura dei messaggi usa **notifiche (acknowledgement) positive** (OK) e **notifiche negative** ("Per favore ripeti")

I protocolli di trasferimento dati affidabili basati su ritrasmissioni sono detti protocolli ARQ (Automatic Repeat Request)

Questi protocolli hanno 3 funzioni aggiunte:

1. Rilevamento degli errori: consente al destinatario di rilevare gli errori sui bit, queste tecniche richiedono l'uso di bit extra tra mittente e destinatario (verranno raccolti nel checksum dati di rtd2.0)
2. Feedback del destinatario: è l'unico modo che il mittente ha per sapere il POV del destinatario
Il nostro protocollo rtd2.0 manderà pacchetto ACK e NAK da destinatario a mittente
3. Ritrasmissione: Un pacchetto ricevuto con errori sarà ritrasmesso al mittente.



a. rdt2.0: lato mittente

Nello stato di destra, il mittente sta aspettando un ACK (`rdt_rcv(rcvpkt) && isACK(rcvpkt)`) allora il pacchetto non presenta errori, altrimenti se riceve un NAK (`rdt_rcv(rcvpkt) && isNAK(rcvpkt)`) allora il pacchetto presenta errori e dovrà rimandarlo.

Se il mittente è in attesa di un ACK o un NAK non potrà mandare dati, perciò, si dice che è un protocollo **stop-and-wait**.

Il lato ricevente ha solo uno stato. All'arrivo del pacchetto il destinatario risponde con un ACK o NAK

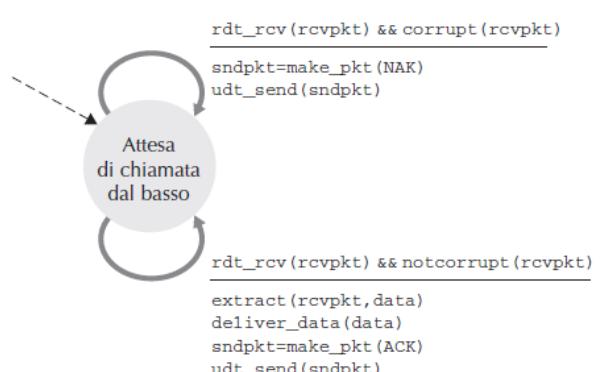
Con `rdt_rcv(rcvpkt)&&corrupt(rcvpkt)` indichiamo un pacchetto alterato.

È possibile che l'ACK o il NAK possano essere corrotti a loro volta. Per risolvere questo problema il mittente rimanda il pacchetto dopo aver ricevuto una notifica alterata, ma facendo così si introducono pacchetti duplicati nel canale.

Facendo questo, però, il destinatario non sa se l'ultimo ACK o NAK mandato sia stato ricevuto correttamente.

A livello mittente sono presenti due stati. Nella prima parte a sinistra si sta aspettando di ricevere i dati.

Quando si verifica `rdt_send(data)` si crea un pacchetto con `sndpkt` contenente i dati e il campo per checksum. Infine si spedisce il pacchetto con `udt_send(sndpkt)`.



b. rdt2.0: lato ricevente

Soluzione: aggiungere un campo al pacchetto dati, numerando i pacchetti con un **numero di sequenza**.

3.4.3 Canale con errori sui bit: rdt2.1

Ora bisogna gestire anche il numero di sequenza. Il protocollo rdt2.1 usa acknowledgement positivi e negativi verso il mittente: positivo quando arriva un pacchetto fuori sequenza e negativo se è alterato.

Se il mittente riceve due ACK per lo stesso pacchetto (ACK duplicati), capisce che il destinatario non ha ricevuto correttamente il pacchetto che viene dopo quello confermato due volte.

Come abbiamo visto prima, una differenza tra rdt2.1 e rdt2.2 è che ora si ha a che fare anche con i numeri di sequenza, per questo motivo il mittente deve controllare il numero di sequenza del pacchetto confermato dall'ACK.

3.4.4 Canale con perdite & errori sui bit: rdt3.0

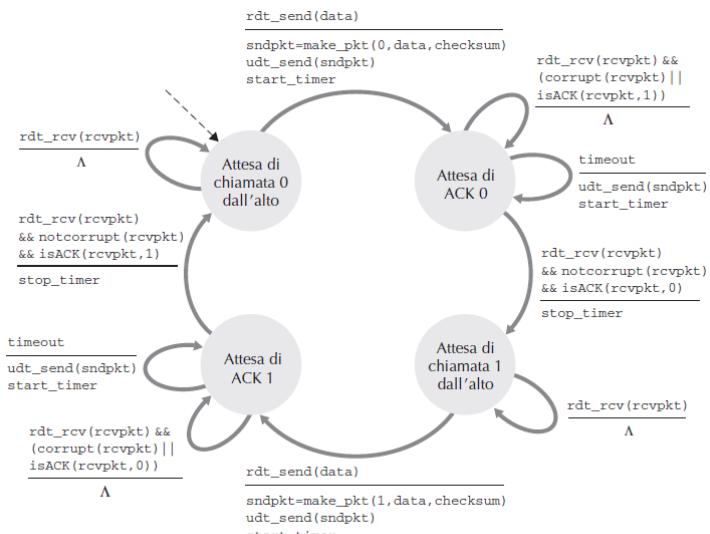


Figura 3.15 Mittente rdt3.0.

vediamo ora un canale che, oltre a danneggiare i bit, possa anche perdere i pacchetti.

Il protocollo deve ora risolvere due problemi:

- come rilevare la perdita dei pacchetti
- cosa fare quando succede

Per risolvere quest'ultimo problema abbiamo già a disposizione il checksum, i numeri di sequenza, gli acknowledgement e la ritrasmissione.

Per quanto riguarda il primo problema spetta al mittente rilevare e risolvere la perdita dei pacchetti.

Consideriamo il caso in cui il mittente manda un pacchetto al destinatario ma l'ACK di questo vada smarrito, così il mittente non saprà mai se il pacchetto sia arrivato o meno. Se il mittente è disposto ad aspettare un tempo sufficiente per essere certo della

perdita, può rimandare il pacchetto. Si sceglie quindi un valore di tempo per il quale lo smarrimento risulti probabile, ma non sicuro. Se non si riceve l'ACK in questo lasso, il mittente rimanda il pacchetto.

Può succedere, però, che il pacchetto presenti ritardo ma non sia andato perso, quindi il destinatario riceverà un duplicato (problema risolto con l'utilizzo dei numeri di sequenza). La ritrasmissione sembra l'unica soluzione. Il mittente deve fare uso di un contatore che segnali la scadenza di un dato periodo di tempo:

1. Si inizializza il contatore quando si manda il pacchetto (anche se si tratta di una ritrasmissione)
2. Si risponde all'interrupt del timer
3. Si ferma il contatore

Il protocollo rdt3.0 viene anche chiamato **protocollo ad alternanza di bit**.

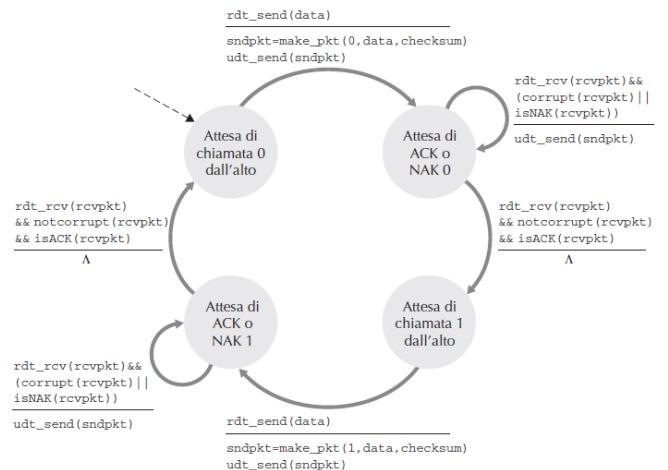


Figura 3.11 Mittente rdt2.1.

3.4.5 Trasferimento affidabile con pipeline

È uno stop-and-wait \Rightarrow poco utilizzo da parte del mittente:

$$U_{mittente} = \frac{L/R}{RTT + L/R} = \frac{,008}{30,008} = 0,00027$$

Il mittente è stato attivo per l'1% del tempo.

Quindi non operiamo più in modalità stop-and-wait, il mittente può inviare più pacchetti senza attendere gli acknowledgement.

Molti pacchetti in transito possono essere visualizzati come il riempimento di una tubatura; quindi, questa tecnica (il non aspettare l'acknowledgement) è nota come **pipelining**. Le conseguenze sul protocollo sono:

- Intervallo dei numeri di sequenza aumentato: ogni pacchetto in transito deve avere un numero di sequenza univoco e bisogna tener conto che ci possono essere più pacchetti in attesa.
- I lati di invio e di ricezione dei protocolli devono memorizzare più pacchetti in un buffer
il mittente deve memorizzare quelli inviati e senza acknowledgement.

La quantità dei numeri di sequenza e la dimensione dei buffer dipendono dal protocollo, da come reagisce alla perdita, alterazione o alto ritardo

Due approcci: **Go-Back-N (GBN)** e **ripetizione selettiva**.

3.4.6 GBN

Il mittente può trasmettere più pacchetti senza aspettare alcun acknowledgement

Però non deve avere più di N

pacchetti in attesa di acknowledgement nella pipeline.

Definiamo come **base** il numero di sequenza del pacchetto più vecchio senza ACK, e **nextseqnum** come il numero di sequenza del prossimo pacchetto da spedire. Abbiamo quindi 4 intervalli:

L'intervallo di numeri di sequenza ammissibili per i pacchetti trasmessi, ma che non hanno ricevuto ancora l'acknowledgement è una finestra di dimensione N .

Quando il protocollo è attivo questa finestra scorre lungo lo spazio dei numeri in sequenza. Per questo motivo N viene chiamata **ampiezza della finestra** e il protocollo viene detto **finestra scorrevole (SW)**.

IL MITTENTE GBN deve rispondere a 3 tipi di evento:

1. Invocazione dall'alto: quando si chiama `rdt_send`, inizialmente si controlla se la finestra è piena.

> Se la finestra non è piena, si crea e si invia il pacchetto e le variabili vengono aggiornate.

> Se la finestra è piena, i dati vengono restituiti a livello superiore

2. Ricezione di un ACK: l'ACK del pacchetto con il numero

di sequenza n verrà visto come un ACK cumulativo, e ci fa capire che tutti i pacchetti con numero di sequenza $\leq n$ sono stati ricevuti correttamente dal destinatario.

3. Evento di timeout: anche qui viene usato un contatore per risolvere il problema dei pacchetti persi. Quando si verifica il timeout, il mittente rispedisce tutti i pacchetti ancora senza ACK

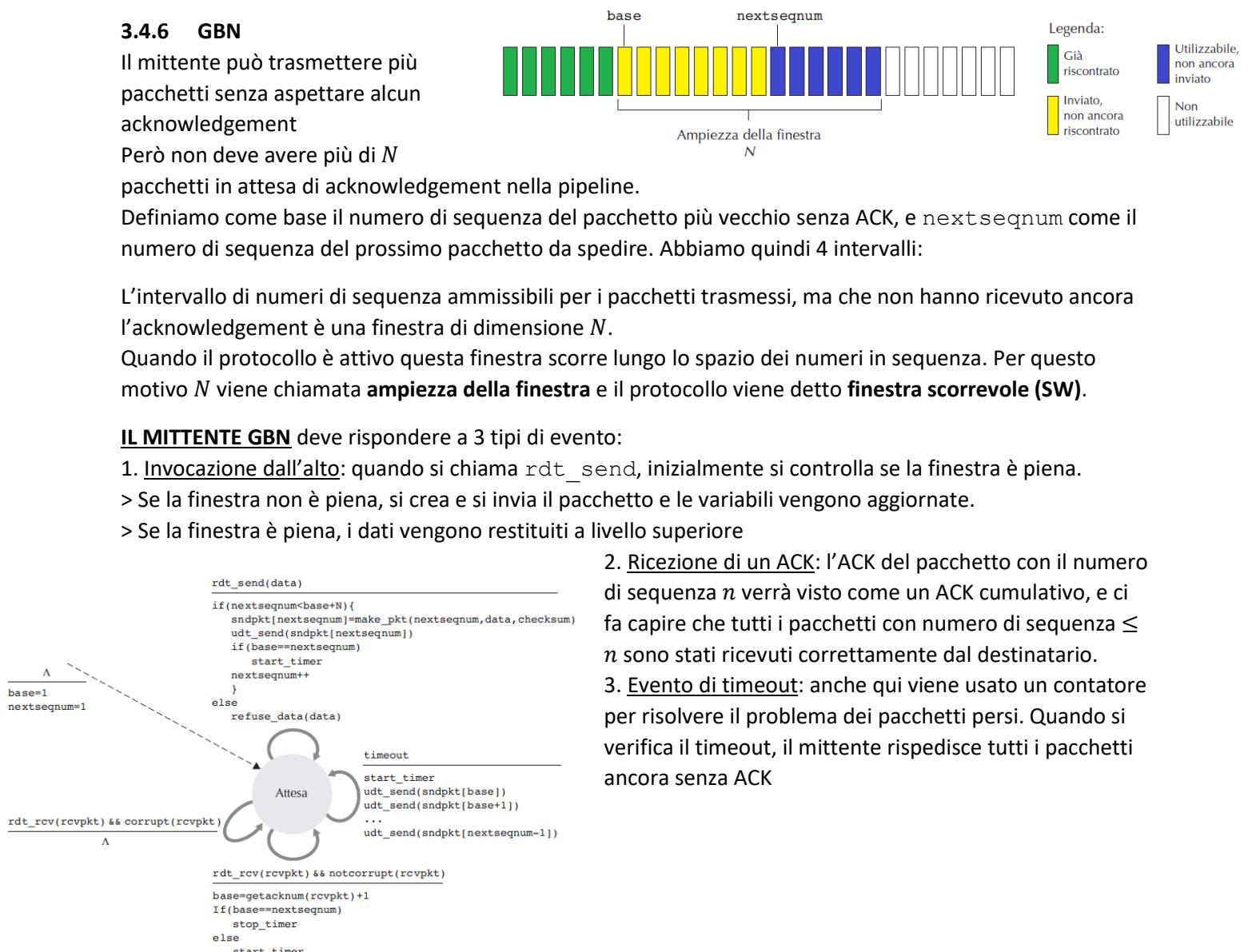


Figura 3.20 Descrizione con automa esteso del mittente GBN.

IL DESTINATARIO GBN

Se un pacchetto con numero di sequenza n arriva in sequenza e correttamente, il destinatario manda ACK per quel pacchetto e consegna i dati al livello superiore. Negli altri casi, il destinatario scarta i pacchetti e manda ACK per il pacchetto in ordine senza errori.

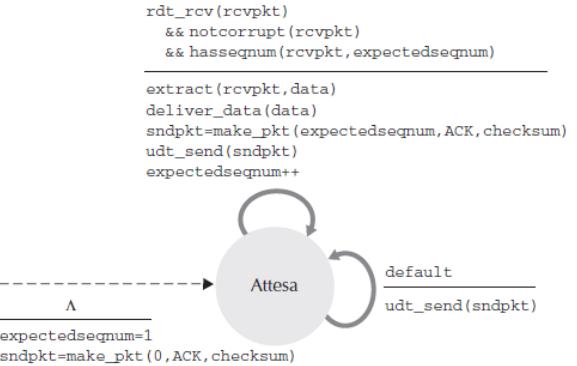


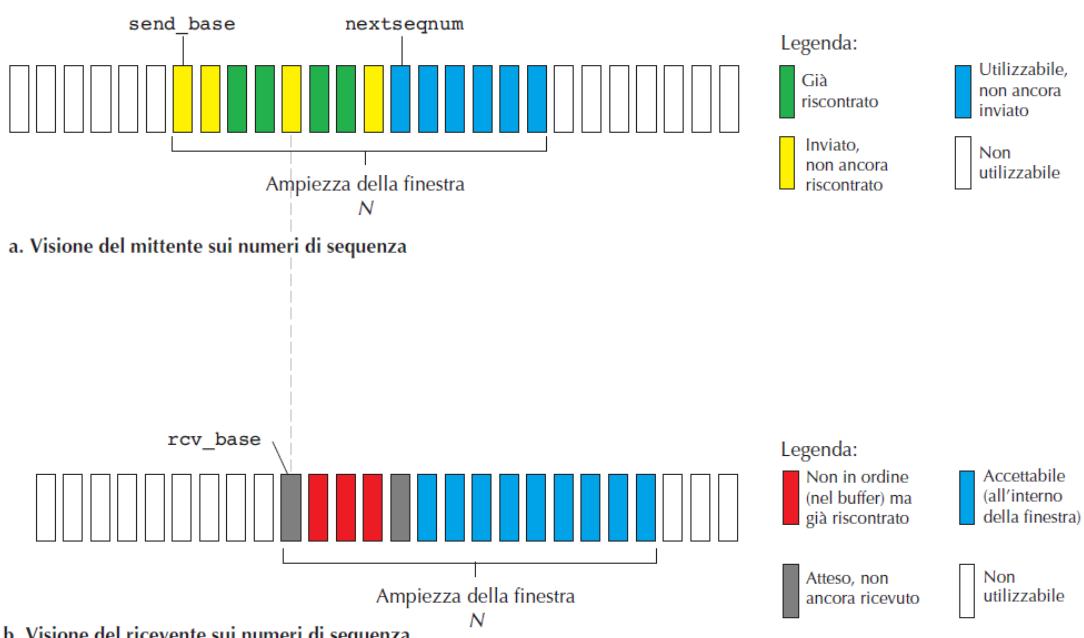
Figura 3.21 Descrizione con automa esteso del ricevente GBN.

3.4.7 Ripetizione selettiva (SR)

È un altro approccio al pipelining, nel quale si ritrasmettono solo i pacchetti dove c'è il sospetto di errore, evitando così delle ritrasmissioni inutili.

Il destinatario manderà ACK specifici per i pacchetti corretti.

Il meccanismo della ripetizione selettiva è simile a GBN, una differenza è che il mittente avrà già ricevuto qualche ACK per qualche pacchetto presente nella finestra.



Inoltre il destinatario manderà gli ACK solo per i pacchetti che non presentano errori, anche se non sono in sequenza. Questi ultimi (i pacchetti non in sequenza), verranno mantenuti in un buffer finché non saranno arrivati i pacchetti per ristabilire l'ordine.

Se il destinatario non invia un ACK per pacchetto, la finestra del mittente non può avanzare.

Questo ci fa capire che non sempre mittente e destinatario hanno le finestre che coincidono. Ciò porta a delle conseguenze quando si ha a che fare con un intervallo finito di numeri di sequenza.

Esempio: intervallo di numeri di sequenza per i pacchetti 0,1,2,3 e ampiezza finestra $N = 3$.

Supponiamo che i pacchetti 0,1,2 vengano inviati e ricevuti correttamente ✓ e che abbiano i loro ACK. Così la finestra del destinatario sposta in avanti la sua finestra verso il 4°, 5° e 6° pacchetto, quindi quelli con numero di sequenza 3,0,1; Due scenari:

- 1) Gli ACK dei primi tre pacchetti **vanno persi** e il mittente li rispedisce ⇒ si verificano dei duplicati
- 2) Gli ACK dei primi tre pacchetti **arrivano correttamente**, il destinatario sposta in avanti la sua finestra verso 4°, 5°, 6° pacchetto, con numero di sequenza 3,0,1.

Il pacchetto 3 va perso, ma arriva il pacchetto 0 con nuovi dati. Quello che vede il destinatario è la sequenza di messaggi ricevuti dal canale e che lui stesso manda sul canale.

Non c'è modo quindi di distinguere la ritrasmissione del primo pacchetto alla trasmissione originaria del quinto.

3.5 TRASPORTO CONNECTION-ORIENTED: TCP

- È **orientato alla connessione** → due host prima di iniziare a scambiarsi dati devono effettuare un handshake.
- Offre un servizio **full-duplex** → flusso bidirezionale dei dati in una stessa connessione
- **Punto a punto** → ha luogo tra un singolo mittente e un singolo destinatario.

3.5.1.1 Come si instaurano le connessioni TCP:

Il processo applicativo del client informa il livello trasporto di voler instaurare una connessione con un server.

- Il client invia per primo un segmento TCP (senza dati di lv app) al server host.
- Il server, ricevuto il segmento, invia al client un altro segmento (anch'esso senza dati) al mittente
- Il client, ricevuto il segmento dal server invia un ulteriore segmento (che può contenere dati) al server.

- Dopo questo "saluto" lo scambio di dati può cominciare

Questo "saluto" viene detto **handshake a tre vie**, perché i segmenti scambiati per instaurare una connessione sono 3.

3.5.1.2 punto di vista dei processi applicativi:

- Il processo client manda i dati attraverso la socket,
- Una volta attraversata sono in mano di TCP che li dirige verso un **buffer di invio**

3.5.1.3 Dimensione massima del segmento

La massima quantità di dati all'interno è limitata.

È limitata da **MSS** (Massima dimensione di segmento)

MSS è ottenuto determinando la lunghezza del frame più grande che può essere mandato sul canale (MTU massima trasmisiva unità)

Si sceglie poi un MSS tale che il segmento TCP stia all'interno di un frame.

3.5.2 Struttura dei segmenti TCP:

La MSS limita la dimensione massima di un segmento TCP, quando si invia un file di grandi dimensioni lo si frammenta in porzioni di dimensione MSS.

- L'**intestazione** include i numeri di porta d'origine e di destinazione (usati per mux e demux) e checksum
- Poi prevede: **numero di sequenza** e **numero di acknowledgement** usati per il trasferimento affidabile
- **Finestra di ricezione** → 16 bit, usato per il controllo di flusso
- **Lunghezza dell'intestazione** → 4 bit, specifica la lunghezza dell'intestazione in multipli di 32 bit
- **Opzioni** → facoltativo con lunghezza variabile, usato quando mittente e destinatario decidono MSS
 - **CWR, ECE**: usati per il controllo della congestione
 - **PSH**: 1 = destinatario deve mandare subito i dati ad applicazione
 - **URG**: indica se i dati sono urgenti
- **Flag** → 6 bit:
 - **ACK**: indica che l'ACK è valido
 - **RST, FYN, FIN**: usati per iniziare e chiudere una trasmissione

Numero di porta di origine										Numero di porta di destinazione	
Numero di sequenza										Numero di acknowledgement	
Lung. intest / CWR ECE URG ACK PSH RST SYN FIN										Finestra di ricezione	
Checksum										Puntatore ai d. urg.	
Opzioni										Dati	

32 bit

3.5.2.1 Numeri di sequenza e numeri di acknowledgement

TCP vede i dati come un flusso di byte non strutturati ma ordinati.

Il **numero di sequenza** in un segmento è il numero nel flusso di byte del primo byte del segmento.

Dato che TCP fa l'ACK solo dei byte fino al primo byte mancante del flusso, si dice che il protocollo offre **acknowledgement cumulativi**.

3.5.3 RTT e timeout

TCP usa un meccanismo di timeout e ritrasmissione per recuperare i segmenti persi.

L'RTT misurato si chiama SampleRTT è la quantità di tempo che passa tra l'istante di invio del segmento e quello di ricezione dell'acknowledgement.

Il SampleRTT non si misura ogni volta ma viene calcolato per uno solo dei segmenti mandati che ancora non hanno l'ACK.

Inoltre il SampleRTT non si calcola per i segmenti ritrasmessi.

Per fare una stima bisogna calcolare una media dei valori di SampleRTT che TCP chiama EstimatedRTT

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT} \quad \text{con } \alpha = 0,125 = \frac{1}{8}$$

Oltre ad avere una stima di RTT è importante anche avere una misura della sua variabilità, che chiameremo DevRTT:

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

Dati questi valori possiamo calcolare l'intervallo di timeout di TCP come

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

3.5.4 Trasferimento affidabile dei dati

IP non è affidabile, non ci garantisce che i pacchetti arrivino privi di errori e in sequenza, per questo TCP crea un servizio di **trasporto di dati affidabile** al di sopra del best-effort di IP.

Ci sono 3 eventi principali che riguardano la trasmissione e ricezione dei dati:

- Dati provenienti dall'applicazione → incapsula i dati che arrivano dal livello applicazione e li passa a IP
- Timeout → TCP ritrasmette il segmento che ha fatto suonare il timeout e riavvia il timer.
- Arrivo dell'ACK con valore valido

Varianti:

1. Raddoppio dell'intervallo di timeout

TCP allo scadere del timer ritrasmette il segmento con il n° sequenza più basso tra quelli che non hanno ricevuto ACK ma ogni volta che ciò accade imposta l'intervallo di timeout successivo al doppio.

Ciò porta ad una limitazione nel controllo della congestione

2. Ritrasmissione rapida

Uno dei problemi della ritrasmissione è che il periodo di timeout può essere molto lungo ed obbliga il mittente ad aspettare prima di rimandare il pacchetto aumentando il ritardo.

Il mittente può rilevare la perdita dei pacchetti prima che si verifichi il timeout con gli **ACK duplicati**.

Questo succede quando il destinatario riceve un pacchetto con n° seq > di quello che aspettava.

Con questo capisce che c'è un buco nel flusso.

Dato che il mittente spedisce svariati pacchetti se uno di questi viene smarrito allora ci saranno più ACK duplicati. Quando questo succede il mittente esegue una **ritrasmissione rapida**, *rispedendo il segmento mancante prima della fine del timer*.

3.5.5 Controllo di flusso

TCP offre un servizio di controllo di flusso per evitare che il mittente saturi il buffer del destinatario.

Per fare ciò TCP fa mantenere al mittente una variabile, chiamata **finestra di ricezione (RW)**, che fornisce al mittente un'indicazione dello spazio libero nel buffer del destinatario.

3.5.6 Gestione della connessione

Come si stabilisce una connessione TCP:

1. TCP lato client manda un segmento TCP a lato server.
Questo segmento speciale (detto **SYN**) non contiene dati a livello applicativo ma ha il bit **SYN = 1**
2. Quando il datagramma SYN arriva al TCP server-side,
Il server: estraе il segmento dal datagramma, alloca le variabili e il buffer, ed invia un segmento (senza dati) chiamato **SYNACK** di connessione approvata al client
3. Quando al client arriva SYNACK anche lui alloca il buffer e le variabili, e infine invia un segmento (che può contenere dati) al server, in questo modo gli comunica che la connessione è stata approvata.

Questo procedimento è detto **handshake a tre vie**.

Durante una connessione TCP, i protocolli TCP in esecuzione negli host attraversano vari **stati TCP**:

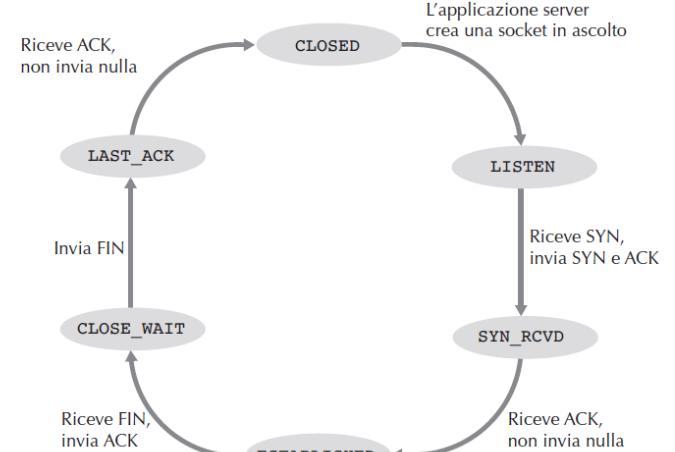
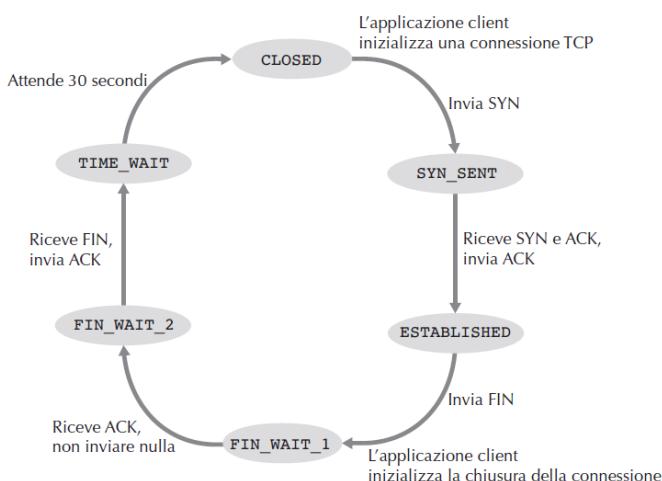


Figura 3.41 Tipica sequenza di stati visitati da un **client TCP**.

Figura 3.42 Tipica sequenza di stati visitati da un **server TCP**.

Se si vuole chiudere la connessione:

- Client e server chiudono la loro parte. Viene mandato un segmento dal client al server con il bit **FIN = 1**.
- Il server risponde al client con l'ACK
- Il server invia un segmento con FIN = 1
- Il client risponde al server con l'ACK
- Fine delle trasmissioni.

3.6 PRINCIPI DI CONTROLLO DELLE CONGESTIONI

In poche parole, si ha **congestione** quando troppe fonti inviano pacchetti troppo velocemente per essere gestiti dalla rete. La congestione si manifesta con perdita dei pacchetti e lunghi ritardi.

Consideriamo 3 scenari

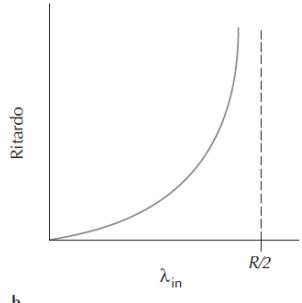
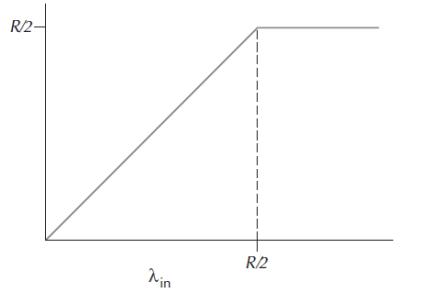
3.6.1 Due mittenti e un router con buffer illimitati

Due host A e B con una connessione che condivide un router. Immaginiamo che un'applicazione in A stia inviando dati sulla connessione con una frequenza λ_{in} .

I dati vengono incapsulati e mandati senza tener conto di errori, congestione o controllo di flusso.

Il tasso con cui A presenta traffico al router è λ_{in} ; B si comporta analogamente.

I pacchetti da A e da B passano da un router e da un collegamento uscente di capacità R .



A sinistra (a.) c'è il throughput per connessione in funzione del tasso d'invio; finché non supera $R/2$ tutto quello che viene mandato dal mittente viene ricevuto con ritardo finito.

(b.) Il collegamento non riesce a mandare con un tasso superiore a $R/2$

3.6.2 Due mittenti e un router con buffer limitato

Ora il buffer ha memoria limitata \Rightarrow se i pacchetti quando arrivano trovano il buffer pieno vengono scartati.

Supponiamo ora che le connessioni siano affidabili, quindi se il pacchetto viene scartato, il mittente lo rimanda. Indichiamo con λ'_{in} il tasso con cui il livello trasporto manda i segmenti.

Le prestazioni dipendono dal comportamento della rete con le ritrasmissioni.

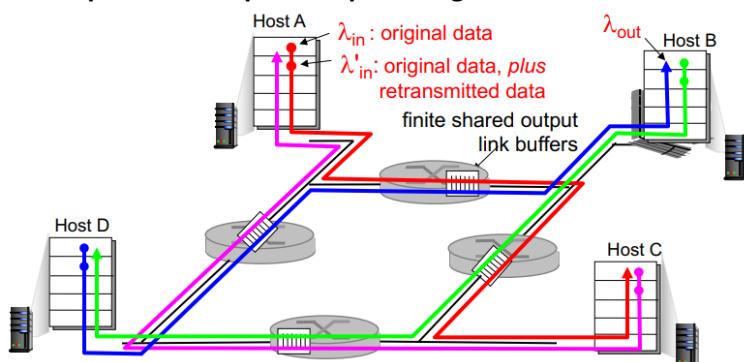
Consideriamo il caso in cui A sia capace di capire quanto spazio libero ha il buffer di B e quindi manda il pacchetto solo quando è libero,
in questo caso avremo che $\lambda'_{in} = \lambda_{in}$ e il throughput sarà λ_{in} .

Consideriamo il caso in cui il mittente trasmette solo quando il pacchetto è perso:
il carico offerto λ'_{in} vale $R/2$, con questo carico di rete il tasso vale $R/3$.

Ultimo caso: mittente che va in timeout prematuramente e ritrasmette un pacchetto che ha subito ritardi ma che non è perduto.
il destinatario quindi si tiene una copia e scarta le altre
Questo però rappresenta un altro costo legato alla congestione di rete: ritrasmissioni inutili.
il throughput così avrà valore $R/4$

3.6.3 Quattro mittenti, router con buffer finiti e percorsi composti da più collegamenti

All'aumentare del λ'_{in} rosso, tutti i pacchetti in blu in arrivo nella coda superiore vanno persi, il throughput blu andrà a zero.



3.7 CONTROLLO DI CONGESTIONE TCP

Spetta a TCP il controllo della congestione perché IP non lo fa (non da feedback sullo status del traffico)

TCP impone a ciascun mittente un limite sulla velocità di invio sulla propria connessione relativamente alla congestione della rete.

- Se il mittente si accorge che c'è poco traffico \Rightarrow incrementa il tasso trasmissivo

- Se c'è tanto traffico \Rightarrow lo diminuisce

Tre domande:

1. Come può TCP limitare la velocità di invio sulla propria connessione?
2. Come percepisce la congestione sul percorso?
3. Quale algoritmo dovrebbe usare il mittente per variare la velocità di invio?

Per controllare la congestione, TCP mette a disposizione una variabile, la **finestra di congestione** ($cwnd$), che impone un vincolo alla velocità di immissione del traffico sulla rete da parte del mittente.

La quantità di dati che non ha ancora ricevuto l'ACK non può superare la soglia:

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

La velocità con cui il mittente spedisce i dati è $cwnd/\text{RTT}$.

Se dovesse esserci una congestione eccessiva, uno o più buffer dei router vanno in overflow, causando la perdita di un datagramma ("evento di perdita") facendo capire a TCP (che non riceve ACK) che c'è congestione sulla rete.

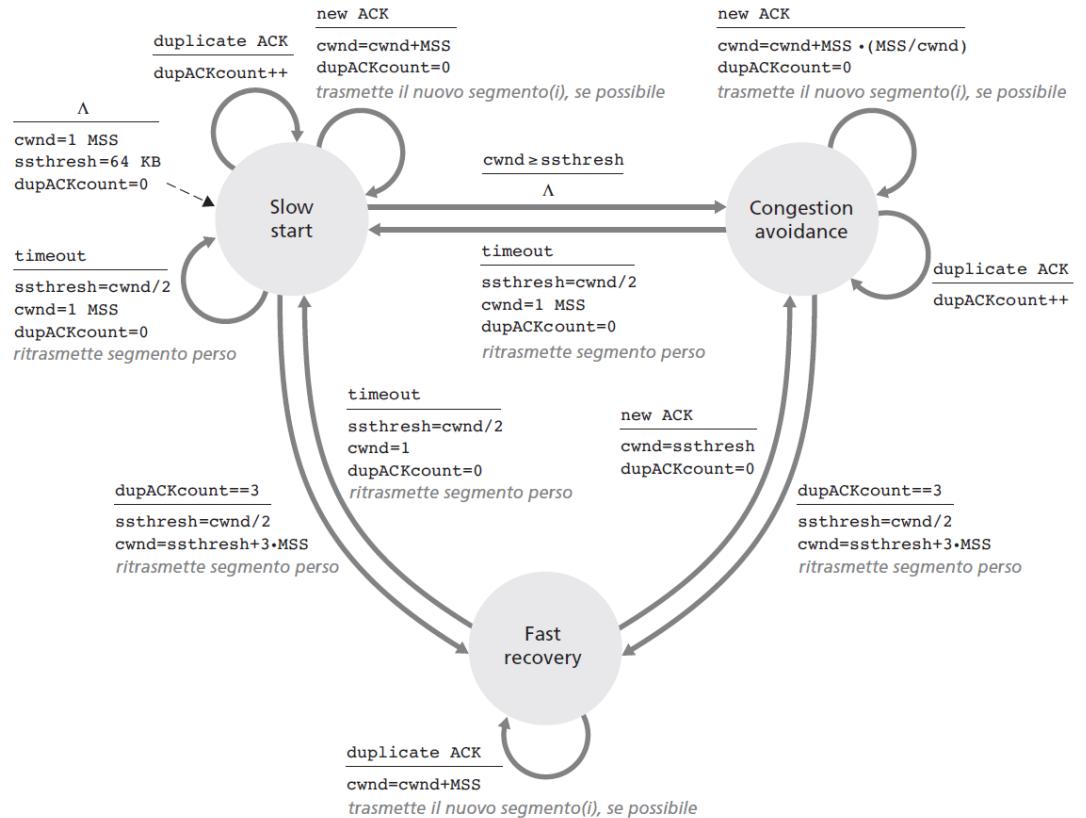
TCP usa gli ACK per scandire gli incrementi della finestra, per questo si dice che TCP è **auto-temporizzato**.

Algoritmo di congestione TCP:

- 1) **Slow start (obbligatoria):** $cwnd$ inizializzato a 1MSS e si incrementa di 1MSS ogni volta che un segmento trasmesso riceve un ACK. Ciò comporta un raddoppio della velocità di trasmissione a ogni RTT, cresce in modo esponenziale.
- 2) **Congestion avoidance (obbligatoria):** Quando entra in questa fase il valore di $cwnd$ è circa la metà di quello che aveva l'ultima volta in cui è stata rilevata congestione. Si incrementa $cwnd$ di 1 MSS ogni RTT

- 3) **Fast recovery (facolt)**

il valore di $cwnd$ è aumentato di 1MSS ogni ACK duplicato ricevuto relativamente al segmento perso, TCP entra in congestion avoidance e riduce il valore di $cwnd$. Se si verifica timeout TCP entra in slow start.



3.7.1 Fairness

Obiettivo: se ci sono K connessioni TCP che passano per lo stesso collo di bottiglia di capacità R , ogni connessione deve impostare il tasso di trasmissione a R/K .

Bisogna fare in modo che $\lambda'_{in} = \lambda_{in}$

Se conoscessimo il più piccolo R del router più lento potremmo limitare il sender ma non sappiamo nulla dei router intermedi

3.7.1.1 Fairness e UDP

Le applicazioni multimediali non usano TCP perché non vogliono che la velocità sia limitata dal congestion control.

Usano UDP invece, perché invia pacchetti a velocità costante e può tollerare la perdita di qualche pacchetto

3.7.1.2 Fairness e connessioni TCP parallele

Applicazioni possono aprire più connessioni parallele tra due host, è quello che fanno i browser web.

3.7.2 Notifica esplicita di congestione (ECN)

È una forma di controllo di congestione assistita dalla rete effettuata in internet.

Vengono coinvolti sia TCP che IP, a livello rete vengono usati 2bit di intestazione nel campo "tipo di servizio" che vengono segnati dal router congestionato.

L'indicazione di congestione arriva all'host destinatario

Il destinatario che vede l'informazione del router congestionato imposta il bit ECE sul datagramma di ACK receiver-to-sender per notificare il mittente della connessione.

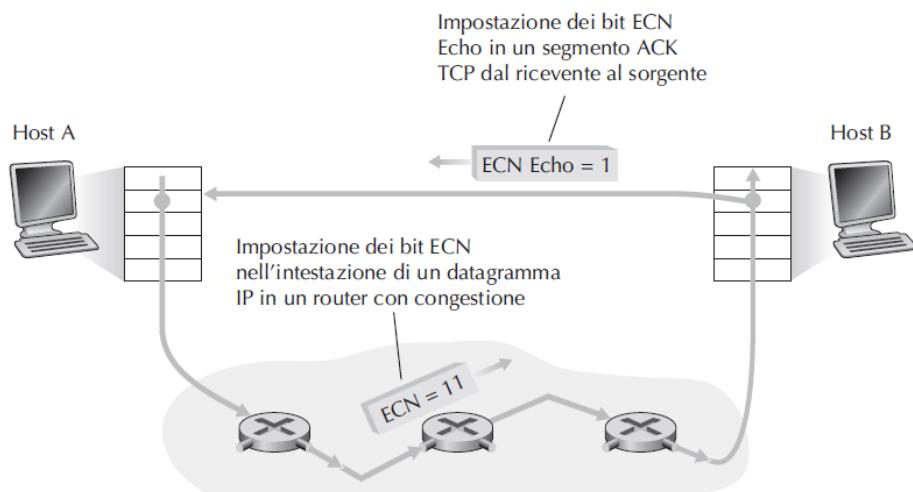


Figura 3.56 Notifica esplicita di congestione: controllo di congestione assistito dalla rete.

4 LIVELLO DI RETE: IL PIANO DEI DATI

Il livello di rete verrà descritto sotto due punti di vista: il **piano dei dati** e il **piano di controllo**, uno incentrato sulla gestione dei dati e l'altro sul controllo (chi l'avrebbe mai detto).

- Lato mittente a livello rete il segmento che viene passato dall'alto viene incapsulato in una busta arancione detta **pacchetto**
- Lato destinatario quel pacchetto viene aperto e si passa al livello trasporto solo la busta rossa "segmento"

Il livello rete ha protocolli implementati su tutti gli host, mittente, destinatario e anche quelli intermedi che devono gestire i pacchetti in viaggio.

Quindi il livello rete unisce concretamente tutta la rete internet dando a tutti gli host un protocollo in comune, il protocollo IP.

I **router** sono degli smistatori di pacchetti verso le destinazioni finali.

Sono gli intermediari che fanno andare avanti i pacchetti e hanno il compito di esaminare l'header della busta arancione e decidere in che direzione inoltrare i pacchetti.

- **Forwarding:** instradare i pacchetti nella direzione giusta grazie alla *forwarding table*
- **Routing:** aggiornamento delle tabelle di instradamento

4.1.1 Piano dei dati Vs Piano di controllo

Il **data plane** è quella serie di azioni realizzate e implementate localmente in ogni router.

Si determinano attraverso il data plane e tutte le sue funzioni la destinazione di inoltro di ogni pacchetto. In pratica tutto ciò che arriva dalle porte di input deve essere analizzato e inoltrato alla porta di output corretta il più velocemente possibile (trasferimento dal buffer di ingresso al buffer di uscita) (**Forwarding**)

Il **control plane** è una logica di livello rete. Non è relativo al comportamento di un solo router, bensì di tutta la rete. Grazie al control plane si può determinare come i pacchetti andranno inoltrati tra i router in modo da farli giungere a destinazione nel minor tempo possibile (**routing**)

4.2 MODELLI DI SERVIZIO A LIVELLO RETE

Ci sono vari modelli di servizio, ognuno definisce le caratteristiche del trasporto end-to-end di pacchetti tra host di origine e di destinazione.

Dobbiamo decidere "quale *modello di servizio* per il canale di trasporto dei datagrammi dal mittente al destinatario"

Esempi di servizio per datagrammi individuali:

- Consegnata garantita
- Consegnata garantita con ritardo limitato

Esempi di servizi per un flusso di datagrammi:

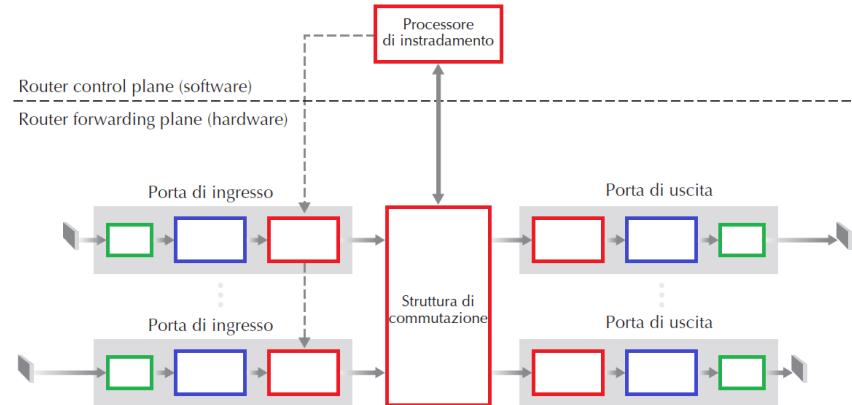
- Consegnata ordinata
- Banda minima garantita

Il livello rete di internet mette a disposizione un solo servizio, noto come "**best-effort**" ovvero "col massimo impegno possibile". Con questo servizio non c'è garanzia che i pacchetti vengano ricevuti nell'ordine in cui sono stati inviati, così come non c'è garanzia la loro eventuale consegna. Non c'è garanzia sul ritardo end-to-end come non c'è garanzia su una larghezza di banda minima. Nonostante lo sviluppo di alternative il modello di servizio best-effort combinato con un'adeguata larghezza di banda si è dimostrato abbastanza buono da supportare una stupefacente gamma di applicazioni. Quindi si usa quello.

4.3 COSA SI TROVA ALL'INTERNO DI UN ROUTER?

Guardiamo le funzioni di inoltro, ovvero alle architetture dei router per trasferire i pacchetti dai collegamenti in ingresso a quelli in uscita.

- Porte di ingresso.** Svolgono le funzioni di:
 - Terminazione di un collegamento in ingresso al router
 - Protocollo a livello collegamento (es. Ethernet)
 - Ricerca, inoltro e accodamento
- Struttura di commutazione.** Connette fisicamente le porte di ingresso con quelle di uscita, è una vera e propria rete in un router di rete.
- Porte di uscita.** Memorizzano i pacchetti provenienti dalla struttura di commutazione e li trasmettono al collegamento in uscita
- Processore di instradamento.** Esegue le funzioni del piano di controllo.

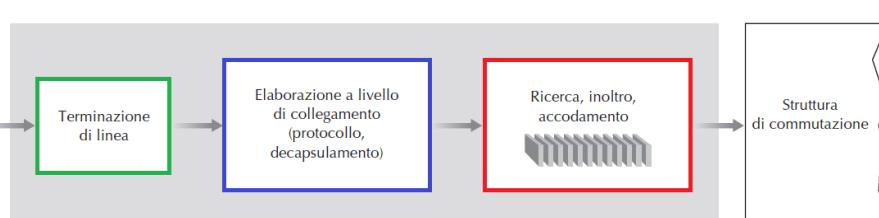


Nei router tradizionali esegue i protocolli di instradamento, gestisce le tabelle di inoltro e le informazioni sui collegamenti attivi ed elabora la tabella di inoltro del router.
Nei router SDN, il processore di instradamento è responsabile della comunicazione con il controller remoto, in modo da ricevere le occorrenze della tabella di inoltro e installarle alle porte di ingresso.

La linea tratteggiata distingue control plane (sopra) e data plane (sotto)

4.3.1 Struttura delle porte di input

Come possiamo notare nelle porte a sinistra sono contenuti tre riquadri di colore diverso, che rappresentano i vari livelli che un pacchetto attraversa quando arriva a un router.



- Il **Verde** è il livello fisico dove i segnali ricevuti vengono convertiti in bit.
- Il **Blu** è il livello MAC/LLC relativo alla tecnologia implementata in quel router a cui il livello fisico passa i bit appena ricevuti.

- Il **Rosso** rappresenta il livello rete che riceve i dati dal livello sottostante e possiede un buffer di ricezione che serve per mantenere i pacchetti che aspettano di essere analizzati e inoltrati.

In particolare, **nell'ultima fase**, viene analizzato l'header del pacchetto. Si fa il lookup (controllo del valore della rete destinazione del pacchetto) e si controlla nella tabella di forwarding per decidere in che direzione l'high-speed switching fabric deve inoltrare il pacchetto. L'high-speed switching fabric andrà quindi configurato ogni volta in base alle informazioni contenute nel routing processor.

Il router confronta un **prefisso** dell'indirizzo di destinazione del pacchetto con una riga della tabella; se c'è corrispondenza il router inoltra il pacchetto al collegamento associato.

Corrispondenza di prefisso	Interfaccia
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
altrimenti	3

Supponiamo che l'indirizzo di destinazione sia 11001000 00010111 00010110 10100001

Dato che i primi 21 bit corrispondono con il prefisso sulla 1 riga, il router inoltra il pacchetto all'interfaccia 0
L'indirizzo di destinazione può corrispondere a più righe; per esempio, prendiamo i primi 24 bit di 11001000 00010111 00011000 10101010 corrispondono alla 2° riga, ma i primi 21 alla 3°...

Quando si verificano corrispondenze multiple il router adotta la **regola di corrispondenza del prefisso più lungo**: → viene determinata la corrispondenza più lunga all'interno della tabella e i pacchetti vengono inoltrati all'interfaccia di collegamento associata

Una volta determinata la porta di output di un pacchetto, esso può essere inviato alla struttura di commutazione.

L'azione di cercare la corrispondenza tra l'indirizzo IP di destinazione ("match") e poi inviare il pacchetto alla porta di uscita specificata attraverso la struttura di commutazione ("action") è un caso specifico dell'astrazione **match-action** che viene eseguita in molti dispositivi di rete.

4.3.2 Struttura di commutazione

Attraverso cui i pacchetti vengono commutati (inoltrati) dalla porta di ingresso alla porta d'uscita.

La commutazione può avvenire in vari modi:

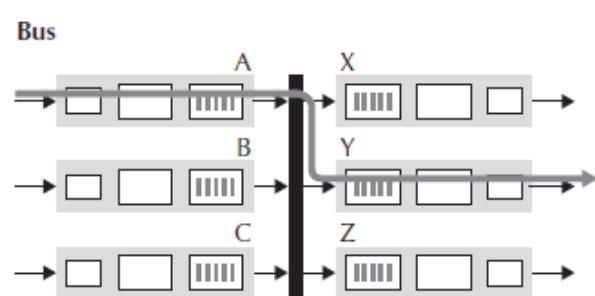
- **Commutazione in memoria.** È stato il metodo utilizzato dalla prima generazione di router che erano dei computer veri e propri che commutavano usando la CPU copiando i pacchetti sulla memoria del sistema.

Il problema è che la velocità è limitata dalla larghezza di banda della memoria.

- **Commutazione tramite bus.** Le porte di ingresso trasferiscono un pacchetto direttamente alle porte di uscita tramite un bus condiviso e senza l'intervento del processore di instradamento.

La commutazione avviene "aggiungendo un'etichetta al pacchetto che indica la porta di output dove deve andare il pacchetto. Il pacchetto viene ricevuto da tutte le porte di output, ma solo la porta corrispondente all'etichetta lo raccoglierà.

Se più pacchetti arrivano contemporaneamente al router tutti tranne uno devono aspettare il loro turno (come una rotonda che può avere una sola macchina per volta al suo interno). ⇒ La lunghezza della banda di commutazione è limitata dalla lunghezza del bus.



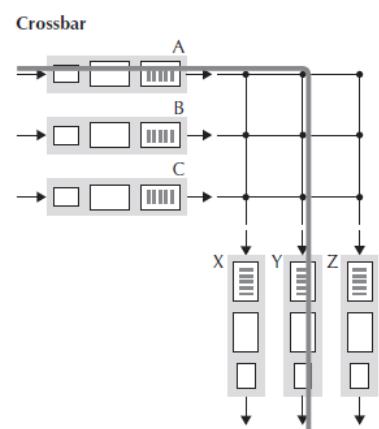
- **Commutazione attraverso rete di interconnessione.** Si utilizza una rete di interconnessione più sofisticata, che implementa una *matrice di commutazione*, ovvero una rete di interconnessione formata da $2n$ bus che collegano n porte di ingresso a n porte di uscita.

Ogni bus verticale interseca tutti i bus orizzontali a un punto di incrocio gestito dal controller.

Questa commutazione è in grado di trasmettere più pacchetti in parallelo.

Una matrice di commutazione è **non-blocking**: un pacchetto che sta essendo inoltrato verso una porta d'uscita non viene bloccato a meno che esista già un altro pacchetto in via di inoltro sulla stessa porta d'uscita.

Se due pacchetti da input diversi devono raggiungere lo stesso output uno dei due dovrà accodarsi alla porta di input perché un solo pacchetto alla volta può essere inoltrato su di uno stesso bus.



4.3.3 Accodamento in ingresso

La struttura di commutazione è più lenta dell'insieme di tutte le porte; quindi, ci può essere accodamento sulle porte di input. L'accodamento ritarda e provoca perdite perché può generare un buffer overflow

Blocco in testa alla coda HOL (Head-Of-the-Line): un pacchetto in testa alla coda dell'input deve attendere il trasferimento per colpa di un pacchetto che lo precede in uscita fermo.

4.3.4 Accodamento in uscita

Abbiamo accodamento in uscita quando la struttura di commutazione lavora più velocemente di quanto i buffer in uscita possono gestire. A causa dell'overflow dei pacchetti possiamo avere ritardo ed eventuale perdita dei pacchetti.

4.3.5 Scheduling

Lo scheduling è la scelta del prossimo pacchetto da inviare sul canale trasmisivo.

La tecnica più intuitiva è FIFO, quindi inviare i pacchetti in ordine di arrivo (come alle poste).

Un'altra scheduling policy è la priority: quando arriva un pacchetto se ne controlla l'urgenza e si smista nella coda d'urgenza corrispondente.

Consideriamo il caso di due code, una ad alta priorità e l'altra a bassa priorità: si controlla se ci sono pacchetti nella prima coda e nel caso vengono spediti per primi, se non ci sono si passa alla seconda cosa. Un pacchetto urgente ha comunque la precedenza su un pacchetto non urgente anche se è arrivato dopo.

Così facendo si rischia che i pacchetti non urgenti non vengano mai inviati ma statisticamente non succede. Comunque nel caso succedesse si parla di *starvation*, scatta il timer del pacchetto in questione che viene rispedito inutilmente creando traffico inutile. Inoltre, aumenta l'RTT dei pacchetti che porta ad un rallentamento nell'invio.

Altra scheduling policy è il **round robin (RR)**: in questo caso prendiamo un pacchetto da una coda e un pacchetto dall'altra in modo alternato. Evitiamo così il problema della starvation dei pacchetti ma perdiamo il concetto di priorità.

C'è anche un'altra generalizzazione del round robin, l'**accodamento equo e ponderato (WFQ)**: ad ogni ciclo viene mandato un certo numero di pacchetti (ordinati per urgenza) per ogni coda.

4.3.5.1 Discard policy

Nel caso di riempimento della coda quale pacchetto andrebbe scartato per liberare la coda?

Ci sono tre politiche diverse per scartare i pacchetti:

- **Tail drop**: si scarta quello in coda
- **Priorità**: si scarta in base alla priorità dei pacchetti
- **Random**: si scarta un pacchetto random.

4.4 INTERNET PROTOCOL (IP)

- Versione: del pacchetto IP (4)
- Header Length: in byte senza campo data
- Type of service: usato per valutare l'urgenza
- Length: lunghezza in byte totale del pacchetto (max 2^{16})
- Identifier: numero di identificazione del pacchetto
- Flags: per la frammentazione
- Fragment offset: per la frammentazione

In particolare, un pacchetto IPv4 da 6400byte non potrà mai essere

<----- 32 bit ----->			
Versione	Lunghezza dell'intestazione	Tipo di servizio	Lunghezza del datagramma (byte)
Identificatore a 16 bit		Flag	Offset di frammentazione a 13 bit
Tempo di vita		Protocollo di livello superiore	Checksum dell'intestazione
Indirizzo IP sorgente (32 bit)			
Indirizzo IP destinazione (32 bit)			
Opzioni (se presenti)			
Dati			

trasmesso su una ethernet come un solo pacchetto da 64000 byte, andrà necessariamente frammentato in frammenti da 1500 byte.

L'identifier serve per capire di quale pacchetto originale il datagramma fa parte; l'identifier avrà il bit che indica la frammentazione settato a 1 e nell'ultimo campo, "fragment offset" ci sarà indicato quale pezzo del pacchetto originale rappresenta, così che possano essere riordinati.

Frammentando un pacchetto IPv4 otteniamo tanti pacchetti IPv4 che avranno vita propria su internet e potrebbero arrivare disordinati o non arrivare mai. Se perdo un frammento tutto il pacchetto va ritrasmesso, dato che quello che ritrasmette è TCP e non esegue lui la frammentazione.

L'unica persona che si occupa della ricostruzione è il destinatario, essendo il *reassembly* molto oneroso, ciò richiede un buffer adatto.

flag = 0 e offset != 0 → ultimo pacchetto

flag = 0 e offset == 0 → pacchetto unico

- TTL: numero di volte che un pacchetto può essere gestito da un router prima di essere ucciso
- Upper Layer protocol: specifica che protocollo di trasporto va passato il pacchetto TCP/UDP
- Header checksum: checksum solamente dell'header, vogliamo assicurarci che l'header sia corretto per poterlo mandare avanti correttamente.
- Indirizzo IP sorgente
- Indirizzo IP destinazione
- Opzioni: tempo di partenza, tracciatura dei router attraversati, lista di router da cui passare
- Dati: segmento TCP/UDP

In totale tra TCP (20 byte) e IP (20 byte) abbiamo 40 byte di overhead. A cui vanno sommati eventuali byte di overhead del livello applicazione.

4.4.1 Indirizzamento IPv4

Un host ha un solo collegamento con la rete

Interfaccia: confine tra host e collegamento fisico

Un router presenta più interfacce, una su ciascuno dei suoi collegamenti

IP richiede che ogni interfaccia abbia un proprio indirizzo IP; quindi, l'indirizzo IP associa un'interfaccia, non un host o un router.

Gli indirizzi IP sono lunghi 32 bit (4byte) ci sono quindi 2^{32} indirizzi disponibili ($\approx 4\text{mld}$), vengono scritti in **notazione decimale puntata**, ogni byte indicato in decimale e separato dagli altri byte da ''

193.32.216.9 → 11000001 00100000 11011000 00001001

Ogni interfaccia ha un IP globalmente univoco ma non possono essere scelti in modo arbitrario, una parte dell'indirizzo è determinata dalla sottorete a cui l'interfaccia è collegata.

Una sottorete (per IP) può essere, per esempio, l'interfaccia di un router che connette 3 host.

IP assegnerà a questa sottorete l'indirizzo 223.1.1.0/24 → 11011111 00000001 00000001 00000000

La notazione /24 è detta **maschera di sottorete** (*subnetmask*) indica che i 24 bit più a sinistra dell'indirizzo (quelli sottolineati) definiscono l'indirizzo della sottorete

4.4.2 CIDR (Classless InterDomain Routing)

È la strategia di assegnazione degli indirizzi.

L'indirizzo IP viene diviso in due parti e mantiene la forma decimale puntata $a.b.c.d/x$

x sono i bit che costituiscono il **prefisso di rete** dell'organizzazione.

I rimanenti $32 - x$ bit possono essere usati per distinguere i vari dispositivi all'interno dell'organizzazione (tutti con lo stesso prefisso di rete)

4.4.3 DHCP (Dynamic Host Configuration Protocol)

Serve per **assegnare** un blocco indirizzi alle interfacce host e router nella propria struttura. Consente ad un host di ottenere un indirizzo IP in modo automatico, conoscere la maschera di rete, l'indirizzo del router e l'indirizzo del DNS server locale. DHCP è **plug-and-play** perché automatizza la connessione degli host alla rete.

Questa connessione avviene attraverso quattro messaggi:

- **Identificazione DHCP:** identificazione del server DHCP con cui interagire, il client manda un messaggio **DHCP discover** di destinazione broadcast 255.255.255.255 sulla rete con sorgente 0.0.0.0
- **Offerta del DHCP:** Un server DHCP che riceve un messaggio di identificazione risponde al client con un messaggio **DHCP offer**, inviato a tutti i nodi della sottorete.

L'offerta server contiene:

- l'ID di transazione del messaggio di identificazione
 - L'IP proposto al client
 - La maschera di sottorete
 - La durata di concessione dell'indirizzo IP
- **Richiesta DHCP:** Il client sceglie tra le offerte e risponde con un **DHCP request**
 - **Conferma DHCP:** Il server risponde al messaggio di richiesta con un **DHCP ACK**, confermando i parametri richiesti

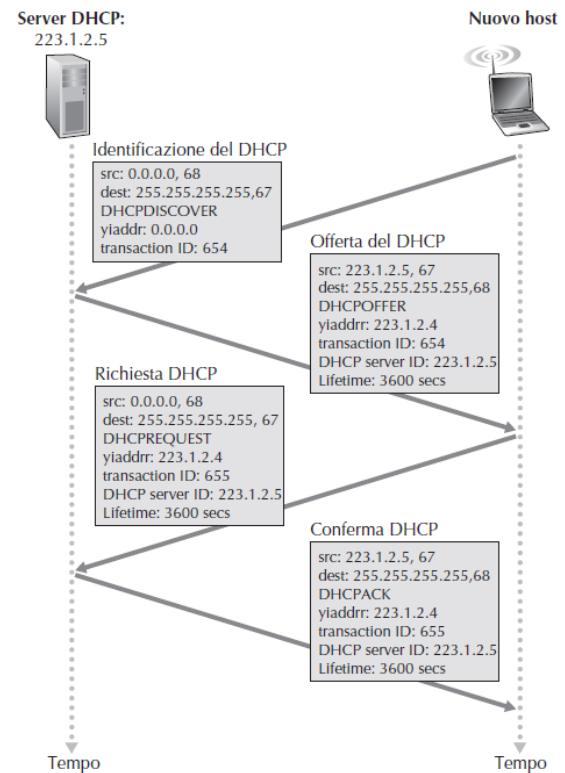
La richiesta DHCP è incapsulata in un pacchetto UDP perché non richiede l'apertura di una connessione

4.4.4 NAT (Network Address Translation)

Sottoscrivendo un contratto con il nostro ISP otteniamo un indirizzo IP che ci permette di connettere un solo host a Internet. In casa però abbiamo decine di dispositivi da connettere ad internet, come facciamo avendo solo 1 indirizzo IP? Usiamo un processo chiamato NAT.

L'indirizzo IP che otteniamo dal nostro ISP è quello associato al nostro router e si trova all'ultimo livello, quello più esterno della struttura di internet. Da una parte, quindi, è connesso al mondo di internet, dall'altra è connesso a una rete locale, quella di casa nostra, dove avviene una magia: possiamo convincere il router che dalla parte della rete locale si trova una rete fittizia, anche di classe A.

Come faccio però a sfruttare questi indirizzi fittizi e permettere agli host della mia rete locale di comunicare su Internet? Fino a quando resto nella rete locale non ho alcun problema; il router avrà come interfaccia interna verso la rete locale un indirizzo IP coerente con la rete fittizia e tutti i dati scambiati all'interno verranno riconosciuti dal router come interni e non li farà uscire su Internet.



Il problema di comunicare su internet con questo sistema non è tanto l'invio della richiesta, perché basta inserire l'indirizzo destinatario e a prescindere da dove viene può essere recapitato al ricevente giusto. Il problema è nella risposta dato che **l'indirizzo associato al mittente è un indirizzo fittizio che non esiste su Internet**. Quindi è necessario utilizzare l'unico indirizzo IP esistente su Internet in nostro possesso e poi occuparsi dello smistamento di tutto quello che arriva a quell'indirizzo ai vari host nella rete locale.

Di questo meccanismo se ne occupa il NAT. In particolare, vengono inventati dei socket utilizzando delle "porte" finte che servono per tener traccia di chi ha fatto richiesta e vuole risposta. Facciamo un esempio:

L'host con indirizzo fittizio nella rete locale 10.0.0.1 apre un browser e si vuole mettere in contatto con il web server che ha indirizzo reale 128.119.40.186 sulla porta 80. Il NAT router che vede arrivare un pacchetto sulla sua rete locale controlla la destinazione e se è interno alla rete locale non viene fatto uscire, altrimenti va inoltrato all'esterno, verso Internet.

Fare questa operazione significa che il NAT router deve convertire l'indirizzo di rete locale in indirizzo pubblico e compilare la tabella di conversione NAT, presente su ogni router.

Il router, quindi, rimpiazza l'indirizzo del mittente della LAN con il suo indirizzo, unico esistente su Internet e per ogni pacchetto proveniente dalla LAN e diretto su Internet c'è una riga in cui troviamo la corrispondenza tra [indirizzo IP mittente + porta vera dell'host] e [indirizzo IP router + porta finta assegnata dal router a quell'host].

Tabella di traduzione NAT	
Lato WAN	Lato LAN
138.76.29.7, 5001	10.0.0.1, 3345
...	...

Nella parte "Lato WAN" si indica ciò che viene visto dall'esterno quindi ci sarà sempre l'indirizzo del router ma con porte differenti a seconda dell'host, nella parte "Lato LAN" è la corrispondenza interna alla rete locale.

3345 è la porta da cui è partita la richiesta del mittente.

5001 è un "numero di porta" inventato dal router.

Moltiplico gli indirizzi usabili su internet, utilizzandone uno solo e tanti numeri di porta.

Il destinatario, ricevuto il pacchetto, genererà una risposta indirizzata a [IP router + porta finta assegnata dal router a quell'host] che arriverà al router il quale avrà premura di leggere il numero di porta e inoltrare il pacchetto al mittente originale nella LAN che aveva fatto richiesta.

Il pacchetto arrivato al router risalirà lo stack di rete fino al livello trasporto e dietro alla porta artificiale 5001 troverà il processo NAT che controllerà nella tabella la riga corrispondente alla porta 5001 e inoltrerà al destinatario legittimo nella LAN il pacchetto (la provenienza è sempre il web server).

Socket diversi dallo stesso host nella LAN avranno righe differenti nella tabella e allo stesso modo due richieste indirizzate allo stesso host su Internet avranno due righe di richiesta diverse, il web server si vedrà arrivare due richieste dallo stesso IP ma da due porte differenti e genererà due risposte distinte.

L'unico limite che ho è il numero di porte artificiali che il NAT router può creare (64.000) ma allora creo due NAT router nella mia rete locale e ottengo il doppio delle porte possibile.

Le connessioni TCP vengono instaurate normalmente tra i due host con l'ausilio trasparente del NAT router che traduce o sostituisce al volo destinatario/mittente quando è necessario.

4.4.5 IPv6

Le motivazioni che portano ad un passaggio a IPv6 sono:

- Gli indirizzi a 32 bit sono già esauriti
- Overhead elevato soprattutto a causa della frammentazione
- Mancanza di quality of service

IPv6 utilizza un header da 40byte (il doppio di IPv4) ma non consente la frammentazione.

Inoltre, vengono introdotti dei nuovi campi:

Versione	Classe di traffico	Etichetta di flusso		
		Lunghezza del payload	Intestazione successiva	Limite di hop
Indirizzo sorgente (128 bit)				
Indirizzo destinazione (128 bit)				
Dati				

- Versione: 4bit che identificano la versione di IP ver = 6 \Rightarrow IPv6 (non vale lo stesso per 4)
- Priority (classe di traffico): 8 bit utilizzato per attribuire priorità ad alcuni datagrammi
- Etichetta di flusso: 20bit per indicare un flusso di datagrammi
- Lunghezza del payload: puntatore all'inizio dell'header relativo al livello trasporto nel campo data
Oppure punta ad una zona del campo data che è estensione dell'header
- Limite di Hop: equivalente di TTL

Altre differenze con IPv4:

- **Checksum:** Non abbiamo più il campo checksum. Considerando la dimensione dell'header in rapporto alla dimensione degli indirizzi, l'header di IPv6 è molto più compatto.
- **Options:** consentite ma fuori dall'intestazione, vengono indicate da "intestazione successiva"
- **ICMPv6:** nuova versione di ICMP
Con ICMPv6 è possibile evitare la frammentazione grazie alla notifica di errore "pacchetto troppo grande" → quando un pacchetto risulta troppo grande viene inviata questa notifica alla sorgente che si mette all'opera per creare dei pacchetti più piccoli.
Viene quindi notificata la massima grandezza che possono assumere i pacchetti in relazione al percorso che devono fare. È un meccanismo che agisce alla sorgente e con un po' più di lavoro all'inizio possiamo evitare la frammentazione.

4.4.5.1 Da IPv4 a IPv6

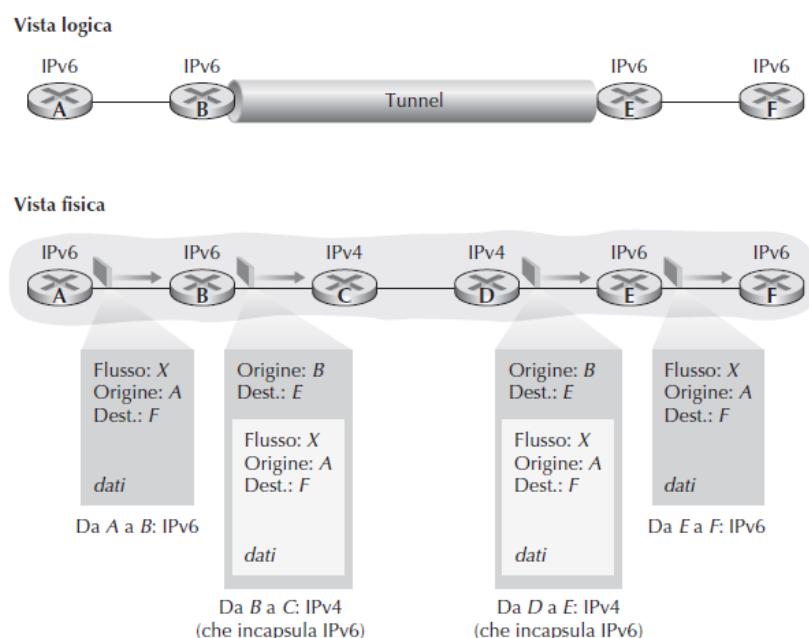
IPv6 è retrocompatibile mentre IPv4 non è compatibile con IPv6, non riesce a gestirne i datagrammi.

Si usa un approccio detto "*tunneling*"; supponiamo di avere due router IPv6 (B ed E) che vogliono utilizzare datagrammi IPv6, ma sono connessi da un insieme di router intermedi IPv4 detti "tunnel".

Il Nodo B, al lato di invio del tunnel prende l'*intero* datagramma IPv6 prevenutogli da A e lo pone nel campo dati di un datagramma IPv4.

Il datagramma IPv4 viene indirizzato al Nodo E al lato ricezione del tunnel e inviato al primo nodo nel tunnel (C). I router IPv4 intermedi instradano il datagramma IPv4, come farebbero normalmente, ignari che questo contenga un datagramma IPv6 completo.

Il Nodo E riceverà il datagramma IPv4, capisce che dentro c'è un IPv6 (lo capisce perché nel numero di protocollo IPv4 c'è scritto 41) per poi estrarre e instradarlo.



4.4.6 Inoltro generalizzato e SDN

SDN (Software Defined Network)

In un approccio SDN il networking è controllato da software e noi possiamo scrivere questo software programmando le funzioni di rete concernenti al forwarding di pacchetti o, meglio, di flussi di pacchetti.

Un algoritmo o un sistemista possono definire a livello centrale delle regole generali di funzionamento che vengono poi trasmesse a livello locale ad ogni singolo router e inserite nelle tabelle locali di controllo dei flussi, diventando l'equivalente delle tabelle di instradamento.

Vediamo un esempio per capire il funzionamento di SDN:

immaginiamo di che nella nostra rete ci sia un router che riceve un pacchetto; confrontando i dati nell'header con i dati nelle tabelle di inoltro locali può capire se quel pacchetto può proseguire o meno e se può in che direzione.

Questa è l'operazione primaria di instradamento ma non è l'unica operazione che può essere fatta; infatti, nelle tabelle ci sono tre colonne:

- Headers: serve per l'instradamento
- Actions: indica che azioni fare per ogni pacchetto
- Counters: serve per tener traccia di svariati valori (numero di pacchetti transitati per un certo flusso, con determinate caratteristiche, provenienti o destinatari a certi indirizzi, ...)

Open flow è una metodologia di definizione delle tabelle di inoltro locali basato su SDN

Partiamo con la definizione del flusso che avviene nell'header del pacchetto.

Le regole di forwarding generalizzate e standard:

- Pattern: prima cosa da fare è riconoscere un pattern, quindi riconoscere il flusso e ricondurre i valori contenuti nell'header ad una delle righe della tabella di flusso.
- Actions: a seconda del match si hanno delle determinate azioni da fare su quel pacchetto (forward, drop, modify)
- Priority: distingue quei flussi che hanno priorità diverse e necessitano gestioni diverse.

La flow table in un router definisce quindi la coppia match-actions per un router.

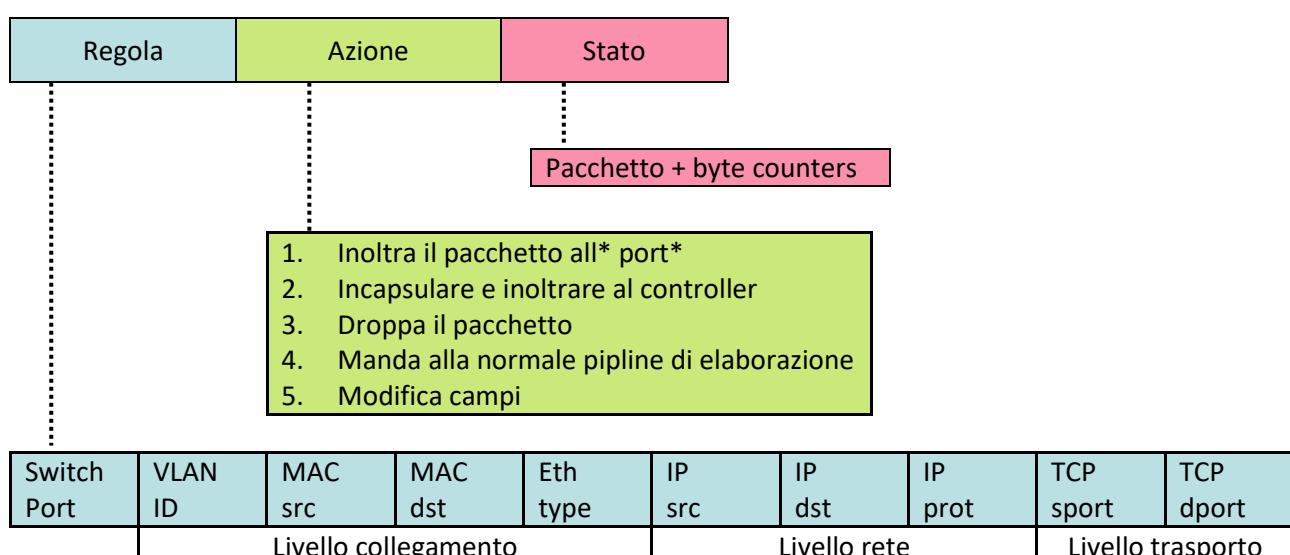
Come possiamo vedere dall'immagine

I campi dell'header che vengono analizzati sono sia relativi al livello 2 che a livello 3, che a livello 4 e a seconda del matching vengono prese delle determinate decisioni (actions).

Possiamo agire e discriminare i pacchetti con un'unica formula rispetto a moltissimi campi.

Es.

- | | | | |
|----|----------------|----------------|----------------------|
| 1. | src = 1.2.*.* | dest = 3.4.5.* | → drop |
| 2. | src = *.*.*.* | dest = 3.4.*.* | → forward (2) |
| 3. | src = 10.1.2.3 | dest = *.*.*.* | → send to controller |



4.4.6.1 Esempi

Switch Port	VLAN ID	MAC src	MAC dst	Eth type	IP src	IP dst	IP prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	51.6.0.8	*	*	*	port6

I datagrammi IP destinati all'indirizzo 51.6.0.8 devono essere indirizzati alla porta di uscita 6 del router

Switch Port	VLAN ID	MAC src	MAC dst	Eth type	IP src	IP dst	IP prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

Non inoltrare (blocca) tutti i datagrammi destinati alla porta TCP 22

Switch Port	VLAN ID	MAC src	MAC dst	Eth type	IP src	IP dst	IP prot	TCP sport	TCP dport	Action
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

Non inoltrare (blocca) tutti i datagrammi inviati dall'host 128.119.1.1

Switch Port	VLAN ID	MAC src	MAC dst	Eth type	IP src	IP dst	IP prot	TCP sport	TCP dport	Action
*	22:A7:23	*	*	*	*	*	*	*	*	port3

11:E1:02

I frame di livello 2 dall'indirizzo MAC 22:A7:23:11:E1:02 devono essere inoltrati alla porta di uscita 3

Possiamo così implementare, grazie a questa astrazione, quattro diversi servizi match-actions che abbiamo già visto o che vedremo:

- Router:
 - Match: longest destination IP prefix
 - Actions: forward su una delle porte
- Switch:
 - Match: Indirizzo MAC di destinazione
 - Actions: forward del pacchetto
- Firewall
 - Match: controllo dell'indirizzo IP e della porta UDP e TCP
 - Actions: permesso o blocco del pacchetto
- NAT
 - Match: indirizzo IP e porta
 - Action: riscrittura di questi due valori

Viene così generalizzato il forwarding che non si limita più ad essere solo destination-based, ma diventa generalizzato. Chi si occupa quindi della computazione delle tabelle di forwarding e delle local flow tables? Il control plane trattato nel prossimo capitolo.

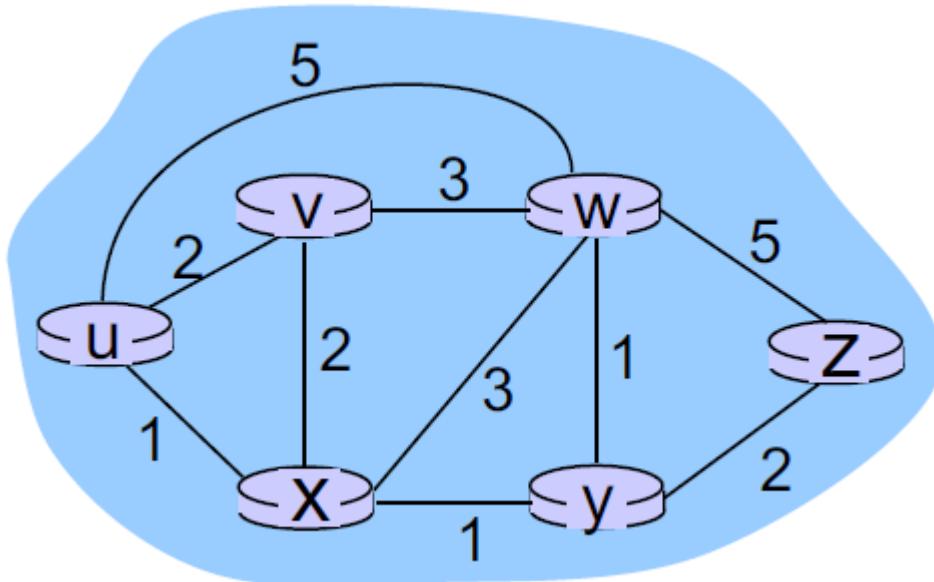
5 LIVELLO DI RETE: IL PIANO DI CONTROLLO

Mentre il data plane implementa le funzioni di forwarding, control plane scrive le tabelle (routing) che poi vengono usate dal data plane per il forwarding. Ci sono due approcci per scrivere le tabelle di routing:

- **Per-router** (metodo tradizionale): ogni router scrive la sua tabella di routing;
- **Controllo centralizzato logicamente (SDN)**: le tabelle di routing sono gestite da un sistema centralizzato da cui tutti i router attingono.

5.1 PROTOCOLLI DI ROUTING

I protocolli di routing cercano i cammini migliori da un router mittente a un router destinatario passando dai router della rete. Il concetto di “**cammino migliore**” non è fisso: si può parlare di costo o di velocità o di congestione della rete. Il problema del routing è tra i più complessi nelle reti, specialmente se per quanto riguarda le reti senza fili, all’interno delle quali gli host possono spostarsi cambiando i possibili cammini.



Per analizzare i problemi di routing si può astrarre una rete ad un **grafo** non orientato (o orientato se necessario) con dei costi che sono appunto relativi al problema da risolvere: possono rappresentare la larghezza di banda di un collegamento o un indicatore di congestione. Si implementa quindi un algoritmo che risolva un problema di cammini di costo minimo e salvi nelle **tabelle di routing** i cammini trovati.

Implementare un simile algoritmo è complicato seguendo l’approccio distribuito (per router); un router non vede necessariamente tutto il grafo (tutta la rete) quindi potrebbe scegliere un percorso che non è ottimale, risulterebbe necessario confrontare le tabelle di routing di ogni router, rendendo questo approccio poco funzionale.

Gli **algoritmi di routing** sono classificati come:

- link state → se tutti i router vedono interamente la rete
- distance vector → se i router conoscono solo le connessioni (e i relativi costi) con i router vicini (a distanza 1), rendendo quindi necessario un approccio iterativo e basato sullo scambio di informazioni con i vicini.

Inoltre, in base al “refresh” delle tabelle di routing gli algoritmi possono essere:

- Statici: se cambiano molto raramente (connessioni cablate)
- Dinamici: se cambiano molto velocemente (connessioni wireless)

5.1.1 Algoritmi link-state

Il più famoso algoritmo link-state è l'algoritmo di Dijkstra, che computa i percorsi di costo minore da un host (sorgente) a tutti gli altri host (destinazioni), riuscendo quindi a costruire una forwarding table completa per l'host sorgente.

Notazione:

$c(x, y)$ = costo collegamento tra host x e host y (settato a $+\infty$ se non sono direttamente collegati)

$D(v)$ = valore corrente del costo del cammino dalla sorgente alla destinazione v

$P(v)$ = nodo predecessore di v nel cammino dalla sorgente

N' = nodi raggiunti e il cui costo è stato calcolato definitivamente

```

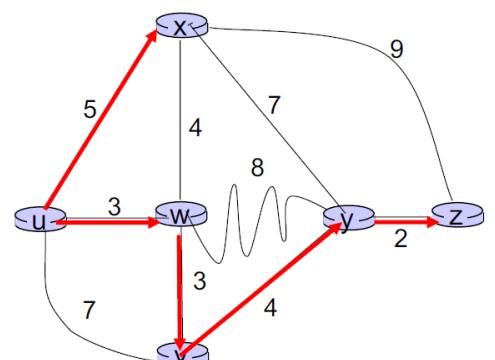
1  Inizializzazione:                                La parte di inizializzazione mette tra i nodi già raggiunti solo 'u'
2    N' = {u}                                         (sorgente) e poi setta i costi dei nodi adiacenti 'v' come c(u, v)
3    per tutti i nodi v                               altrimenti setta il costo a +∞
4      se v è adiacente a u
5        allora D(v) = c(u, v)
6      altrimenti D(v) = ∞
7
8  Ciclo
9    determina un w non in N' tale che D(w) sia minimo
10   aggiungi w a N'
11   aggiorna D(v) per ciascun nodo v adiacente a w e non in N':
12     D(v) = min ((D(v), D(w) + c(w, v))
13   /* il nuovo costo verso v è il vecchio costo verso v oppure il
      costo del percorso minimo noto verso w più il costo da w a v */
14   ripeti il ciclo finché non si verifica che N' = N

```

Nel loop invece sceglie il nodo ' w ' in N' e analizza tutti i ' v ' adiacenti a ' w ' che non siano salvati in N' (ancora non definitivi, quindi): controllo se sia più conveniente lasciare il loro costo di raggiungimento come prima, oppure se aggiornarlo con il costo di ' w ' + $c(w, v)$ (costo dell'arco $w-v$), ovvero sceglie se passare per ' w ' per raggiungere ' v '. Ripete questo finché tutti i nodi non sono in N' .

È omessa la parte relativa ai predecessori, che però è fondamentale per salvare su quali host si sono attraversati per raggiungerne uno.

Step	N'	$D(v)$ $p(v)$	$D(w)$ $p(w)$	$D(x)$ $p(x)$	$D(y)$ $p(y)$	$D(z)$ $p(z)$
0	u	7, u	3, u	5, u	∞	∞
1	uw	6, w	5, u	11, w	∞	
2	uwx	6, w		11, w	14, x	
3	uwxv			10, v	14, x	
4	uwxvy				12, y	
5	uwxvyz					



I costi e predecessori evidenziati sono i vari nodi scelti con costo minimo, quindi si lavora sui relativi adiacenti, ad ogni iterazione. È così costruito un albero grazie ai vari $p(v)$ che costruiscono un percorso.

Problema: mandare pacchetti lungo la strada migliore non rischia di congestionarla? Certo! Ma è comunque meglio che mandarli lungo una strada peggiore. La soluzione è data dal fatto che le tabelle di routing vengono costruite con refresh rate di pochi secondi/millisecondi; quindi, se si pone come parametro la congestione dei collegamenti, il problema verrà risolto subito, cambiando rotta repentinamente.

Complessità Dijkstra: $O(n^2)$ (esistono implementazioni migliori con costi $O(n \cdot \log n)$)

Oscillazioni possibili: quelle citate sopra, ovvero spedendo pacchetti sul cammino migliore, la congestione delle reti migliora, quindi dopo si potrebbe scegliere un altro cammino meno congestionato, che potrebbe congestionarsi a sua volta.

5.1.2 Algoritmi Distance vector

L'algoritmo più famoso è quello Bellman-Ford, che si basa sulla programmazione dinamica (ovvero salvare i risultati già ottenuti e usarli per restituire il valore richiesto).

Notazione:

$d_x(y) :=$ costo del percorso di costo minimo da x a y , ovvero:

$$d_x(y) = \min_v \{c(x, v) + d_v(y)\}$$

(si somma il costo di un collegamento a un nodo adiacente con il costo del cammino da quel nodo destinazione desiderato; prendo quindi il nodo adiacente che fornisce la somma minima).

L'algoritmo è quindi un'iterazione asincrona (detto algoritmo gossip, ovvero un algoritmo che si basa sul cambiamento di eventuali parametri che porta quindi a un ricalcolo) che avviene per ogni cambiamento di costi nei collegamenti.

Si tratta inoltre di un sistema distribuito poiché un host avverte i vicini solo se il loro vettore 'd' cambia, i vicini a loro volta iterano Bellman-Ford e avvisano i vicini **solo se** il loro 'd' cambia.

Bellman-Ford quindi non raggiunge mai l'ottimale vero e controlla continuamente; tuttavia, in caso di reti basate su distance vector per forza (reti wireless) B-F è l'unica scelta.

DIFFERENZE TRA LINK STATE E DISTANCE VECTOR

Velocità:

- LS: $O(n^2)$
- DV: il tempo è variabile, potrebbero esserci dei routing loops, inoltre l'algoritmo è sempre pronto a reiterare

Robustezza, che succede se "un router segnala un costo di collegamento errato a un altro router"?

- LS → quel router avrà una routing table errata.
- DV → tutti i router della rete avranno una routing table sbagliata, dato che i vari 'd' si basano sulla comunicazione dagli altri router (quindi l'errore si propaga nella rete intera)

5.1.3 Instradamento interno ai sistemi autonomi:

Le reti non sono "piatte" ma gerarchiche, è quindi molto difficile creare un algoritmo che non lavori su router "allo stesso livello", si creano quindi distinzioni gerarchiche dette **sistemi autonomi** (AS, *autonomous system*), generalmente composti da gruppi di router posti sotto lo stesso controllo amministrativo.

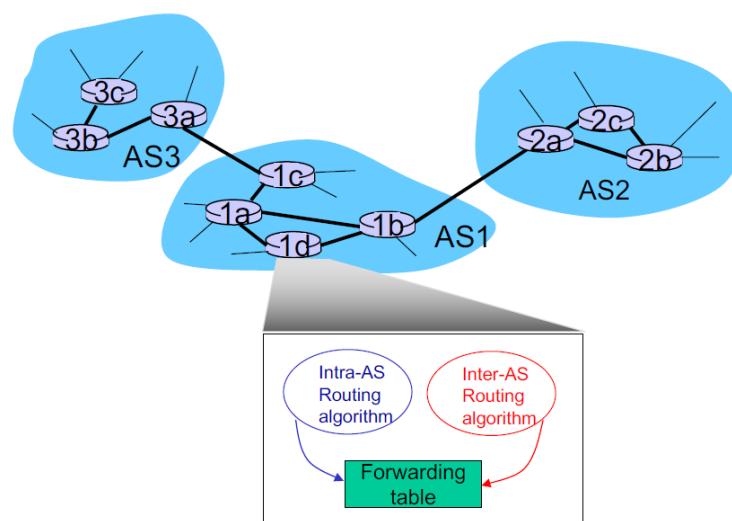
L'algoritmo di instradamento in esecuzione in un AS è detto **protocollo di instradamento interno al sistema autonomo** (*intra-AS routing protocol*)

Spezziamo quindi il problema in due parti:

- Muoversi nel proprio AS (intra)
- Muoversi tra diversi AS (inter)

Es. voglio andare da 3b a 1°

→ per muovermi da 3b a 3° uso un algoritmo intra-AS,
→ per poi andare da 3a a 1c uso un algoritmo inter-AS
→ per poi andare da 1c a 1a uso un algoritmo intra-AS



È quindi facile per un algoritmo come Dijkstra capire come spostarsi da un AS a un altro: usa gli unici collegamenti esistenti, riuscendo quindi a stabilire direttamente i cammini inter-AS.

Nel caso inter-AS quindi non si tratterà per forza di congestione o di larghezza di banda parlando di "costo di collegamento", ma anche di, ad esempio, prezzi di collegamento tra ISP diversi o logiche di sicurezza tra nazioni diverse, ecc.

5.2 INTRA-AS: OSPF

I protocolli Intra-AS sono anche detti **IGP (Interior Gateway Protocol)** e i più comuni sono:

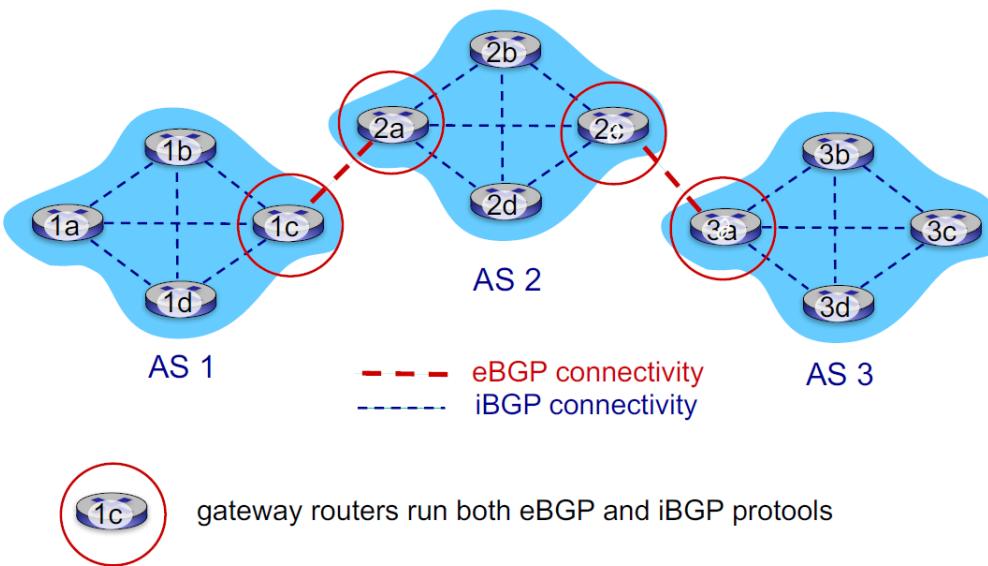
- RIP: Routing Information Protocol
- OSPF: Open Shortest Path First
- IGRP: Interior Gateway Routing Protocol

OSPF fornisce anche una certa sicurezza, garantendo che tutti i messaggi siano autenticati, evitando messaggi da routing “intromessi” nella rete. Inoltre, questo protocollo gerarchizza ulteriormente i router intra-AS, definendo, ad esempio, boundary routers i router che si connettono ad altri AS, che necessitano quindi sia un algoritmo intra-AS che uno inter-AS.

5.3 INTER-AS: BGP

Il protocollo principale è il BGP (Border Gateway Protocol), esso è definito come “*la colla che tiene insieme tutto internet*”, poiché permetta appunto che ogni AS sia connesso agli altri AS, garantendo la globalità di internet. Esso fornisce a ogni AS due componenti informative:

- eBGP (external BGP): ottiene le informazioni sulla raggiungibilità dei router di altri AS
- iBGP (internal BGP): propaga queste informazioni a tutti i router interni all'AS.



gateways routers run both eBGP and iBGP protocols

Esso determina quindi i router migliori degli altri AS da raggiungere a partire dall'AS in questione.

Questo funzionamento permette quindi a tutte le sottoreti di essere riconosciute dalla rete globale come raggiungibili e, soprattutto, permette di sapere come raggiungerle.

Per scambiarsi informazioni si crea una sessione BGP, ovvero uno scambio di messaggi attraverso una connessione TCP semi-permanente, tra i router di “confine”.

Se, per esempio,
si unisse una nuova rete x all' AS-3,
- 3a avviserebbe 2c (via eBGP) che lui è il router migliore da cui passare per mandare i messaggi a x .
- 2c quindi (via iBGP) avviserebbe tutti router di AS-2 (2b, 2a, 2d)
- 2a avviserebbe 1c (via eBGP) che lui è il router migliore da cui passare per arrivare a x .
- 1c (via iBGP) avviserebbe tutti i router di AS-1 (1b, 1a, 1d) di queste informazioni.

MESSAGGI BGP

- **OPEN**: apre una connessione TCP con un BGP peer remoto, autenticando il BGP peer mittente
- **UPDATE**: avvisa di un nuovo cammino
- **KEEPALIVE**: tiene la connessione aperta anche se non vengono inviati update (e ACK per OPEN)
- **NOTIFICATION**: avvisa di errori nel messaggio precedente (e per chiudere la connessione)

5.4 SDN CONTROL PLANE

SDN si basa sull'esistenza di una struttura che calcola tutte le tabelle di instradamento necessarie ai router della rete, favorendo un sistema scaricabile che non si preoccupa della specificità dei singoli router, ma **elabora i percorsi in maniera generalizzata**, permettendo quindi una manutenzione più efficiente e un aggiornamento dei protocolli più semplice.

Due problemi:

- **Single Point Of Failure**: un solo sistema che deve gestire l'intera rete, se fallisce tutta la rete crolla;
- **Enorme congestione possibile**: tutti i dati dalle reti devono arrivare al sistema centrale e da quest'ultimo devono partire per tutti i router delle reti.

❓ Ci si chiede ? : se un sistemista volesse far passare i dati da una sorgente a una destinazione **non** per il cammino minimo, come può fare?

→ cambiare i costi dei collegamenti per portare l'algoritmo a restituire il risultato sperato?

Errato; i costi non sono dei semplici pulsanti da poter premere o meno, ma spesso sono frutto di valutazioni specifiche relative (per esempio) al traffico di collegamento

→ **Risposta giusta**; si utilizzano dei sistemi come quelli OpenFlow permessi da SDN per dare comandi specifici ai singoli router (es. "se arriva pacchetto da 'u' mandalo in 'v' senza curarti di un possibile nodo 'w' collegato con costo minore")

Il problema di questa *potenza* di modulazione di cammini è l'eventuale errore umano: se vengono impostati manualmente percorsi impossibili o fortemente tendenti a congestione, c'è forte possibilità di rendere la rete completamente inutilizzabile. Sarebbe inoltre difficile scoprire l'errore data la complessità e la grande mole di codice relativo ai sistemi OpenFlow.

5.4.1 Traffic engineering

Letteralmente "*ingegneria del traffico (di rete)*", è lo scopo finale del sistema SDN, ovvero poter scegliere arbitrariamente quali percorsi utilizzare per un determinato traffico, in base a QoS, priorità dei pacchetti...

5.5 ICMP (INTERNET CONTROL MESSAGE PROTOCOL)

È un protocollo che fornisce una serie di messaggi (principalmente di errore) necessario per funzioni come Ping o Traceroute.

Traceroute, ad esempio, dato un host di destinazione restituisce l'RTT relativo a ogni router attraversato per raggiungere la destinazione. Vengono inviati una serie di pacchetti UDP con TTL crescente (il primo ha 1, il secondo 2, ecc.) e quando la serie n -esima arriva all' n -esimo router viene restituito un messaggio ICMP (tipo 11 code 0) al mittente così sa quale sia l' n -esimo router e ne ha calcolato l'RTT grazie al messaggio ICMP. Traceroute si ferma una volta raggiunta la destinazione, oppure se la porta indicata è

Tipo ICMP	Codice	Descrizione
0	0	risposta echo (a ping)
3	0	rete di destinazione irraggiungibile
3	1	host destinazione irraggiungibile
3	2	protocollo destinazione irraggiungibile
3	3	porta destinazione irraggiungibile
3	6	rete destinazione sconosciuta
3	7	host destinazione sconosciuto
4	0	riduzione (controllo di congestione)
8	0	richiesta echo
9	0	annuncio di un router
10	0	scoperta di un router
11	0	TTL scaduto
12	0	intestazione IP errata

irraggiungibile, oltre che, ovviamente, se il mittente cessa di inviare pacchetti.

5.6 SNMP (SIMPLE NETWORK MANAGEMENT PROTOCOL)

È un protocollo che sfrutta degli "agenti" definiti per ogni rete che comunicano tra loro fornendosi dettagli riguardo alle funzionalità e i problemi della rete.

6 CRITTOGRAFIA

Obiettivi:

- **Riservatezza:** i testi non devono essere noti ad esterni
- **Integrità:** il testo non deve essere modificato durante la trasmissione
- **Accessibilità:** il testo deve essere accessibile a mittente e destinatario
- **Autenticazione:** mittente e destinatario devono autenticare di essere loro

Attacchi principali:

- **Eavesdrop:** Origliare
- **Insert:** inserire e modificare il messaggio
- **DOS (Denial Of Service):** impedire di utilizzare il servizio
- **IP Spoofing:** fingere di essere un'altra persona
- **Hijacking (911):** dirottamento dei pacchetti

Nel linguaggio della crittografia abbiamo:

- Testo in chiaro PLAINTEXT (m)
 $m = \text{CANE}$
- Testo cifrato CIPHERTEXT
 $K_A(m) = \text{DBOF}$

Quando si vuole trasmettere un messaggio tra due persone è necessaria una **Chiave** per cifrare il messaggio

- Chiave Simmetrica
La possiedono sia Alice che Bob
- Chiave Pubblica
È a disposizione di chiunque

Per forzare uno schema si possono effettuare diversi attacchi:

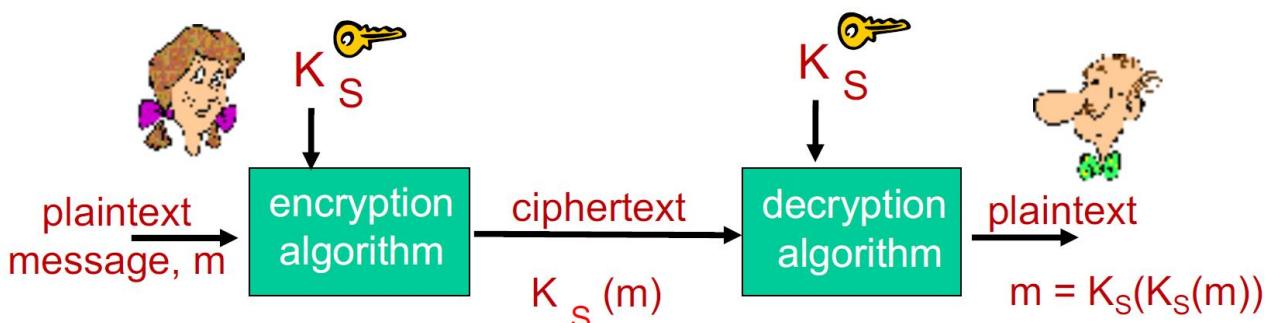
- **Attacco al testo cifrato:** Si analizzano i messaggi e in base ad un'analisi statistica si capisce la chiave
- **Attacco con testo in chiaro noto:** Come con le parole crociate crittografate, si hanno un paio di corrispondenze e si deriva da quelle la crittografia
- **Attacco con testo in chiaro scelto:** Conoscono un messaggio cifrato (es. Hail Hitler con enigma)

6.1 CRITTOGRAFIA CON CHIAVE SIMMETRICA

Nella crittografia con chiave simmetrica, mittente e destinatario condividono la stessa chiave K_S 

Il mittente cifra il proprio messaggio con la chiave K_S

Il destinatario poi la decifra sempre con la stessa chiave K_S



Esempi base di cifrari Simmetrici:

- **Cifrario di cesare:** si shiftano le lettere di k posizioni, k è la chiave (eg. $k = 3$; $a \rightarrow d, g \rightarrow j, \dots$)
- **Sostituzione di caratteri:** come nel gioco delle parole crociate crittografate, non c'è uno schema preciso ma diversi pattern di sostituzione delle lettere
- **DES (Data Encryption Standard):** Si utilizza una chiave simmetrica di 56 bit, da questa chiave vengono create altre 16 sottochiavi che vanno a trasporre per 16 volte il testo.
Per decifrarlo bisogna applicare le sottochiavi al contrario, dalla 16° alla 1°
- **3DES:** DES applicato 3 volte, con 3 chiavi diverse
- **AES:** è come DES ma più grosso, si effettuano più passaggi, chiavi più grandi, messaggi più grandi...

Il principale problema nella crittografia con chiave simmetrica è lo scambio della chiave che ***non può avvenire in chiaro!***

Si sviluppa così la crittografia in chiave pubblica.

6.2 CRITTOGRAFIA CON CHIAVE PUBBLICA

Si basa su una coppia di chiavi per ogni utente; una Pubblica libera e accessibile a tutti (K_B^+) e una Privata e personale. (K_B^-)

Quella pubblica viene utilizzata per criptare il messaggio

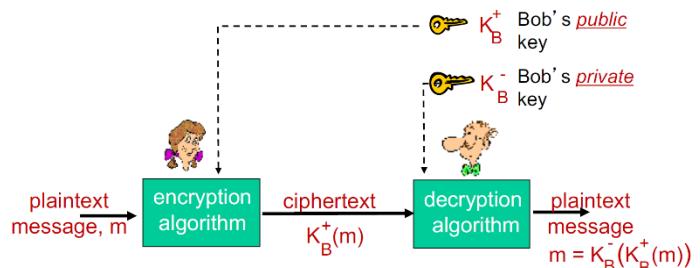
Quella privata invece viene usata per decriptarlo.

Esempio di funzionamento:

Alice cifra il testo con la chiave pubblica di Bob K_B^+

Solo Bob, ora può decifrarlo con la sua chiave privata K_B^-

$$K_B^-(K_B^+(m)) = m$$



Il principale algoritmo a chiave pubblica è RSA

6.2.1 RSA (Rivest Shamir Adleman)

CRIPTAZIONE:

Funzionamento dell'algoritmo

1. Generazione di due numeri primi p, q (da 1024 bit)
2. $n = p \times q$
3. $\phi(n) = (p - 1) \cdot (q - 1)$
4. Scelgo e tra quelli che soddisfano queste condizioni $e = \begin{cases} 1 < e < \phi(n) \\ \text{coprimo con } n, \phi(n) \end{cases}$
5. Scelgo d , tra quelli che soddisfano questa condizione: $d \cdot e(\text{mod}(\phi(n))) = 1$

Esempio

1. $p = 2, q = 7$
2. $n = 2 \times 7 = 14$
3. $\phi(14) = (2 - 1) \cdot (7 - 1) = 1 \cdot 6 = 6$
4. $e = [2, 3, 4, 5]$ ma: 2, 4 sono pari, quindi non vanno bene; e 3 è divisore di 6, Quindi scelgo 5
5. $d = [5, 11, 17, 23, 29, 35, 41, \dots]$
Tra tutti questi, io a caso scelgo 11

N.B. La dimensione del messaggio dev'essere minore di quella di n , altrimenti dovremmo spezzettare il messaggio in m parti più piccole.

6.2.2 Autenticazione nelle reti.

Si utilizza l'**Authentication Protocol (AP)** che ha più versioni:

1. Messaggio con cui mi autentico da solo

Sono io!

Facile da fregare

2. Messaggio con cui mi autentico e faccio veder l'IP

Sono io! Guarda qua (procede a mostrare IP)!

Facendo IP spoofing si aggira facilmente.

(Come se facessi una foto alla tua carta di identità)

3. Mi autentico, faccio vedere l'IP e uso Password

Sono io - IP - Abracadabra

Anche qua se si "origlia" la password, si aggira facilmente

- 3.1 Come prima ma crittografo la password.

La pw viene crittografata con chiave simmetrica, quindi è facile intercettare la chiave.

4. Cambia il metodo.

Alice

trasmette l'autenticazione

Bob

chiede di confermare crittografando
un numero casuale da lui generato

Lo crittografa con simmetrica

Anche qui intercettare la chiave fa cadere tutto.

5. Come prima, ma la chiave è asimmetrica

Problema: man in the middle → se Trudy si mette in mezzo, Bob pensa di parlare con lei ed è fregato.

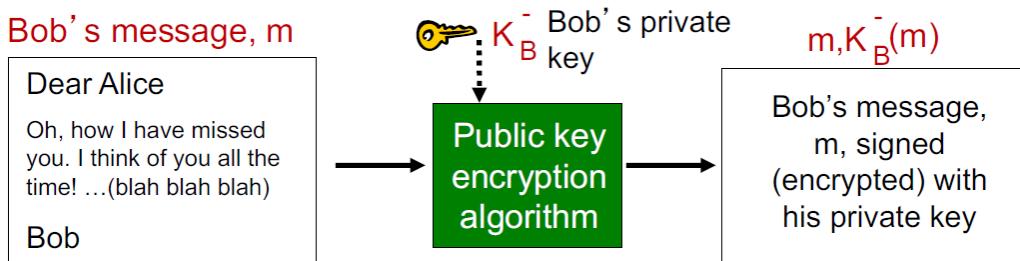
Come si risolve? Esiste un ente, la **CA Certification Authority** che associa una chiave pubblica ad una persona e garantisce per lei.

Quando un utente vuole comunicare con Alice, chiede alla CA e così è sicuro di evitare Man in the middle.

6.2.3 Firma digitale

Praticamente per essere sicuro che il mittente sia lui, si deve autenticare, quindi cosa fa:

- manda due messaggi, uno PLAINTEXT e l'altro CIPHERTEXT (cifrato asimmetricamente con chiave privata)
- Chi riceve, per essere sicuro che sia stata Alice a mandare, decifra con chiave pubblica il messaggio cifrato e se corrispondono vuol dire che il mittente è autenticato.



6.2.4 Message digest

Serve per svolgere il lavoro della firma digitale ma più in grande.

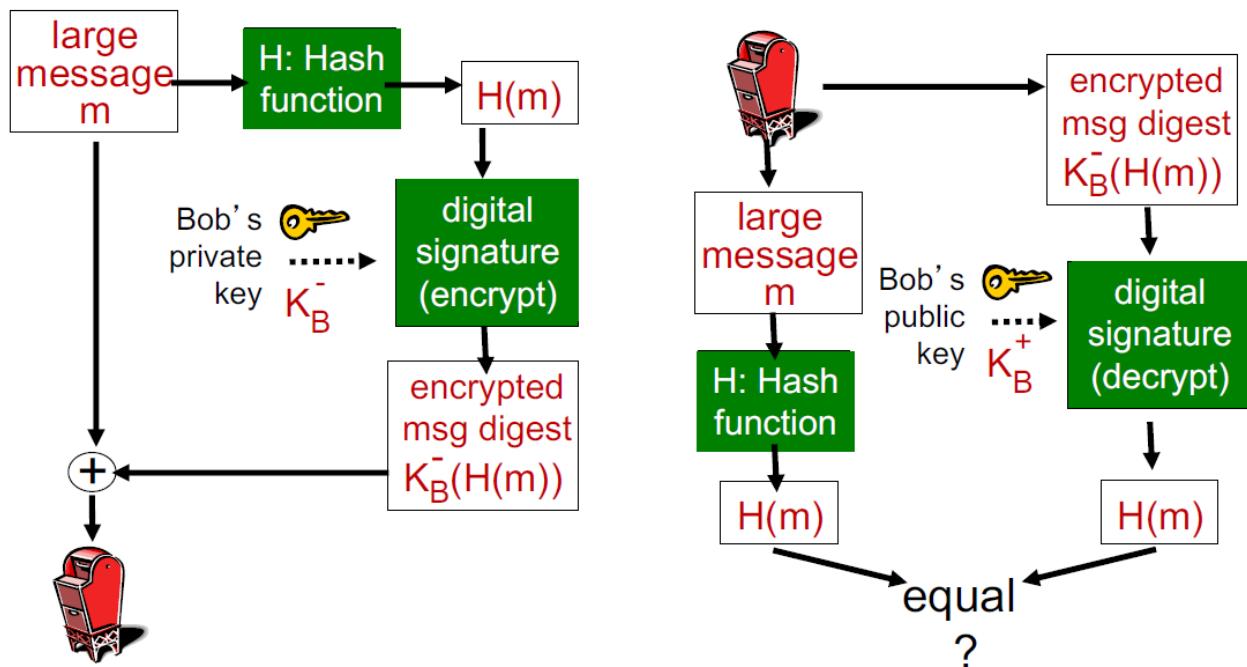
Quando ho messaggi m troppo grandi, cifrarli diventa **pesante**

Quello che faccio è passare il messaggio attraverso una funzione hash (vedi dopo cosa può essere) per poi cifrarlo $m \rightarrow H(m)$ (essendo diventato più maneggevole).

Invio poi entrambi: $m, H(m)$

Chi riceve, riceverà m e $H(m)$

Quello che deve fare è passare m attraverso la funzione per ottenere $H(m)'$
in questo modo se gli $H(m) \equiv H(m)'$ ⇒ il mittente è verificato.



6.2.5 Cifratura ibrida

Essendo gli algoritmi a chiave asimmetrica pesanti per messaggi molto grandi (come viene fatto sopra) si può ovviare a questo attraverso la cifratura ibrida.

Praticamente si prende una chiave simmetrica, la si cifra con chiave asimmetrica e poi si condivide con Bob.

È come mandare un piccolo pacchetto chiuso a chiave (la cui chiave ha solo Bob) dentro al quale c'è il doppione di una chiave che ha Alice così che si possano scambiare i messaggi con chiave simmetrica (che è più veloce)

6.2.6 E-mail sicura

Dipende da cosa voglio garantire, in base a quello decido cosa fare:

- Confidenzialità

Per questo mi basta cifrare tutto con la chiave pubblica di Bob. Solo lui potrà leggerlo.

- Autenticazione e integrità

Calcolo $H(m)$ e lo cifro ed invio: m e $K_A^+ H(m)$, in questo modo Bob, dovrà decriptare $H(m)$ con chiave la mia pubblica $K_A^+ (K_A^-(H(m)))$ [Autenticazione], e poi confrontarlo con m [integrità].

- Autenticazione, integrità e segretezza

Ovviamente il metodo di prima manda il messaggio in chiaro pubblico per tutti.

Se voglio nasconderlo devo implementare la cifratura ibrida di prima, quindi quello che farò è:

- Calcolo $H(m)$ e lo cifro con la mia chiave privata K_A^-
- A $K_A^-(H(m))$ concateno m
- Cifro il tutto con la chiave di sessione.
- Cifro la chiave di sessione con la chiave pubblica di Bob
- Concateno il tutto e spedisco a Bob

Per leggere Bob dovrà:

- Decifrare con la sua chiave privata e trovare la chiave di sessione [segretezza]
- Decifrare con la chiave di sessione il messaggio
- Calcolarsi $H(m)$ da m che avevo concatenato
- Decifrare $K_A^+(K_A^-(H(m)))$ [autenticazione]
- Confrontare i due $H(m)$ [integrità]

6.3 SSL-TLS

SSL → Secure Sockets Layer, è un protocollo che si interpone tra il livello trasporto e il livello applicazione

Serve a garantire: Autenticità, integrità e segretezza delle comunicazioni tra socket

TSL → Trasport Layer Security, è una versione avanzata di SSL

Viene utilizzato lo schema di sopra delle e-mail sicure.

1) Fase di handshake

Cliente e server si conoscono, ovvero si scambiano i certificati quelli della **CA**, e concordano gli algoritmi di decifratura supportati dal client in modo da generare la chiave.

2) Key derivation

Essendo una sola chiave poco sicuro, si generano più chiavi per una trasmissione, utilizzando dei nonce (numeri casuali)

3) SSL record

È la struttura del messaggio SSL, vengono impiegati dei byte per specifiche, poi ci stanno i dati della trasmissione e infine il MAC (**Message Authentication Code**)

4) Sequence number

Utilizzato per evitare **replay attack** si mette un nonce nel record, il nonce è unico e proprio del pacchetto, se viene trasmesso un messaggio con un nonce già usato vuol dire che quel messaggio è stato mandato con replay attack.

5) Control Information

Utilizzato per evitare **truncation attack** si inserisce un record settato a 1 per indicare la chiusura, e settato a 0 per indicare la trasmissione dei dati

6) SSL cipher suit

Si scelgono **Algoritmo a chiave simmetrica**, a **chiave pubblica**, e un **algoritmo MAC** per la comunicazione

Di solito vengono scelti **AES** per la simmetria, **RSA** per la pubblica e **RC2/RC4** per il MAC

6.4 IP SEC

È un protocollo a livello rete (livello 3) implementato per le VPN (Virtual Private Network)

Una VPN è una rete virtuale accessibile da remoto con la quale possiamo comportarci come se fossimo connessi fisicamente (ma non lo siamo).

IP Sec ci permette di spedire pacchetti a livello IP garantendo autenticazione, segretezza e integrità.

6.5 FIREWALL

I firewall sono filtri di rete, filtrano i pacchetti in entrata e in uscita dalla rete/host

6.5.1 Stateless firewall

Vengono determinate delle regole di filtraggio e vengono poi tradotte nella ACL (Access Control List)

6.5.2 Statefull firewall

Servono per verificare le connessioni TCP,

Blocca se vede una richiesta per una connessione TCP che non è mai stata aperta.

6.5.3 Application gateway firewall

Fa da intermediario per la richiesta di connessione.

Chi vuole aprire una connessione, fa richiesta all'application gateway che farà i suoi controlli e poi (eventualmente) la aprirà.

6.5.4 DMZ (Zona Demilitarizzata)

È una zona della LAN delimitata da un firewall per tenere separati gli host che danno servizi esterni alla rete.

7 WIRELESS

7.1 PROPRIETÀ DELLE RF (RADIO FREQUENZE)

Come vengono generate le radiofrequenze ?

L'**energia elettromagnetica** è generata da corrente alternata ad alta frequenza nelle antenne, antenne che convertono l'energia elettrica in Segnali a Radiofrequenza e viceversa.

Il voltaggio (V) varia sinusoidalmente ai due estremi di un'antenna creando uno sbilanciamento di carica; maggiore è lo sbilanciamento implica più elettroni che corrono da sx a dx e da dx a sx.

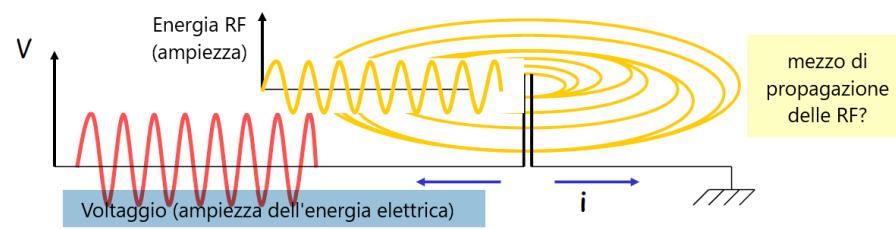
Le **onde elettromagnetiche** si generano quando gli elettroni di un corpo si muovono e tipicamente ogni oggetto materiale genera **onde** costantemente in condizioni ordinarie.

Una carica che si sposta nello spazio

induce un'emissione di onde elettromagnetiche

(il modo in cui varia il segnale rosso fa variare il segnale giallo)

Il mezzo di propagazione delle onde radio è il vuoto.

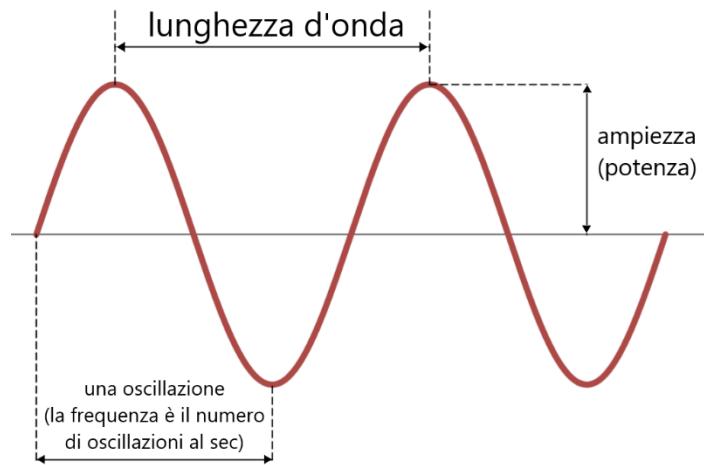


Aampiezza (M): Graficamente è la misura della distanza tra la base e la cima.

L'ampiezza rappresenta l'energia del segnale, e quindi la quantità di potenza energetica utilizzata nella trasmissione.

Per generare un segnale con ampiezza maggiore, quindi, abbiamo bisogno di più energia (Potenza ed energia sono direttamente proporzionali)

$$\text{Potenza di trasmissione (Watt)} = \frac{\text{Energia}}{\text{Tempo}} = \frac{\text{Joule}}{\text{Sec}}$$



Frequenza (f): indica quante volte in unità di tempo la funzione si ripete ("Numero di cicli completi della sinusoide del segnale per unità di tempo")

Lunghezza d'onda (λ): è pari alla velocità della luce (c) diviso la frequenza

$$\lambda = \frac{c}{f}$$

Il WiFi ha lunghezza d'onda 12,5cm.

$$\lambda_{WiFi} = \frac{300.000.000 \text{ m/s}}{2.400.000.000 \text{ Hz}} = 12,5 \text{ cm}$$

Nella pratica, esiste una proprietà fisica che dice che le antenne per essere più efficienti devono avere una lunghezza pari a $1, \frac{1}{2}, \frac{1}{4}$ (o comunque un sottomultiplo binario) della lunghezza d'onda del segnale che stiamo usando. Infatti, tipo le antenne del router WiFi, hanno le antenne che comunicano a $\lambda = 12,5\text{cm}$ e anche le antenne degli smartphone comunicano ad un sottomultiplo binario della lunghezza d'onda.

7.2 PROPAGAZIONE DELLE RF

Una problematica nel mondo delle radiofrequenze ha a che fare con la loro propagazione; non è possibile ottenere una copertura generale, ovvero non è possibile avere un segnale che una volta trasmesso arrivi ovunque.

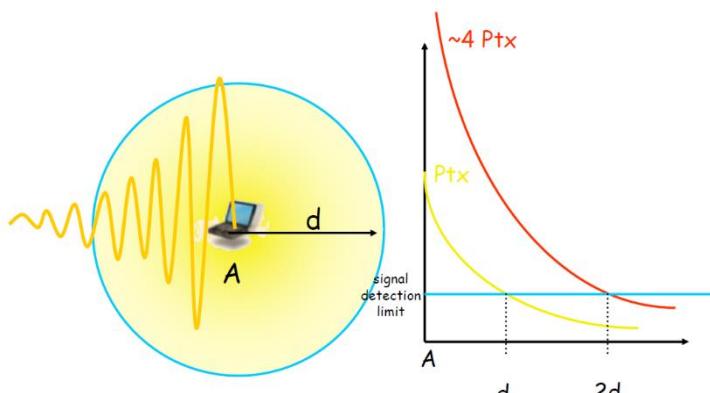
Il segnale parte con ampiezza molto alta ma all'aumentare della distanza l'ampiezza tende a diventare quasi nulla, cioè l'energia del segnale radio è molto bassa ed è come non riceverla.

L'energia radio però è capace di trasmettersi a grandi distanze. L'energia radio della sonda Voyager 1 ad esempio, che è ormai fuori dal sistema solare, è potenzialmente ancora ricevibile e la potenza utilizzata dal trasmettitore è pari a 50W (come quella di una lampada da comodino).

Il segnale decade però in modo esponenziale. La **Signal Detection Limit** (linea Azzurra nel grafico) è la minima energia necessaria al quale ricevere il segnale per capire informazioni dall'onda radio.

Esiste una distanza ' d ' oltre la quale il segnale arriva con un'energia quasi pari al Signal Detection Limit. La circonferenza formata da questa distanza d è pari quindi alla Radio Transmission Coverage, ovvero alla zona in cui abbiamo una copertura necessaria per poter trasmettere.

Fuori da questa circonferenza non è possibile instaurare un link di comunicazione (e quindi non avrei un segmento di rete locale).

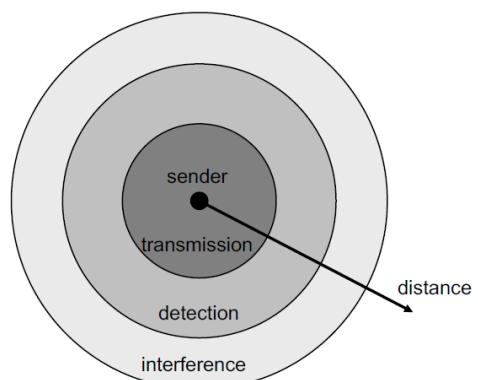


Ci sono due modi per risolvere questo problema:

- Avvicinare il client
- Aumentare la potenza trasmittiva del trasmettitore

← Un'altra cosa molto importante da notare è che, per poter raddoppiare la distanza al quale trasmettere, dobbiamo quasi quadruplicare la potenza trasmittiva del segnale in partenza (e quindi la potenza con cui il segnale viene generato).

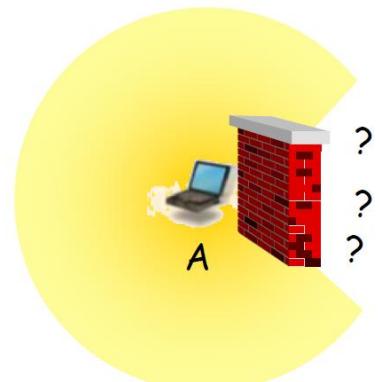
- **Transmission range**
 - Sono possibili le comunicazioni
 - Basso tasso di errore
- **Detection Range**
 - Possibile rilevamento del segnale
 - Comunicazioni impossibili
- **Interference Range**
 - Il segnale può non essere rilevato
 - Il segnale si aggiunge al rumore di fondo.

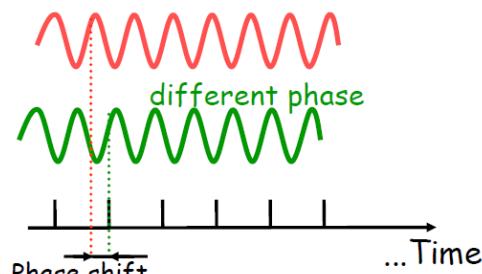


Queste 3 differenti situazioni creano ambiguità per il mac protocol (che deve agire e compiere delle operazioni differenti in base alla zona in cui si trova)

Gli ostacoli possono assorbire o riflettere le onde (dipende dal materiale e dalla frequenza delle onde).

Gli ostacoli bloccano segnali con frequenze più alte (come la luce del sole che ha frequenza altissima è bloccabile anche con un foglio).





Fase (φ): spostamento sull'asse del tempo della sinusoide dell'onda rispetto ad un segnale di riferimento.

"La fase varia in base al punto in cui mi colloco, il segnale impiega un certo tempo t per raggiungermi. Più mi allontano e più il segnale risulta essere in ritardo."

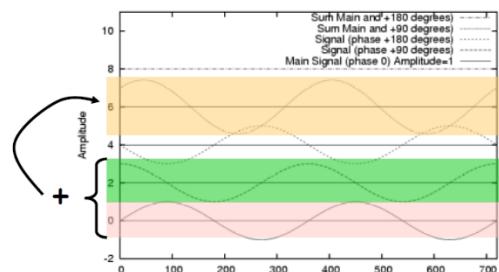
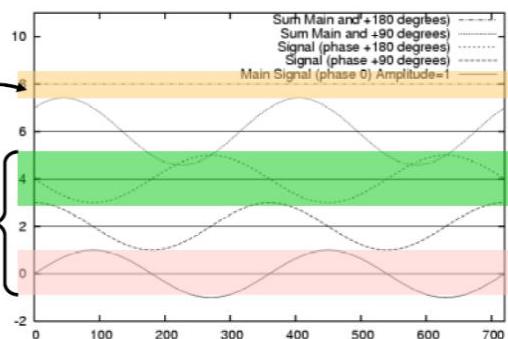
La fase è positiva se rispetto al segnale di riferimento ha uno spostamento a sinistra (segna in anticipo) e negativa altrimenti.

Problema: il segnale che ricevo ad una certa distanza arriva sia per via diretta che facendo una strada più lunga (ad esempio rimbalzando su degli ostacoli), ed i segnali che percorrono una strada più lunga hanno uno sfasamento maggiore.

Il segnale che parte dal trasmettitore è lo stesso; il segnale che il ricevente riceve è pari alla somma vettoriale tra segnale diretto e il segnale che ha fatto una strada più lunga (maggior sfasamento).

In ogni istante di tempo, l'energia che l'antenna del ricevente cattura è sia quella del segnale diretto che del segnale che ha fatto la strada più lunga, e se i segnali hanno dei valori differenti il segnale risultante è la somma vettoriale dei segnali catturati dall'antenna.

Sommando punto a punto (vettorialmente) il valore della sinusoide verde e della sinusoide rosa otteniamo la sinusoide arancione. In questo caso avrei un guadagno di ampiezza.



Lato ricevente, ovviamente è preferibile ricevere segnali risultanti con ampiezze maggiori. In alcuni casi però la somma vettoriale (punto per punto) di più segnali ha come risultante 0 (anche a pochi cm dal trasmittente è possibile avere una somma di segnali che si autoannulla) Questo fenomeno viene chiamato **opposizione di fase**.

Possiamo immaginare questo anche attraverso i suoni, immaginando due segnali acustici opposti emessi da una sorgente che sovrapposti si annullano.

↑ Opposizione di fase, fasi sfasate di circa metà periodo (180°) La somma del segnale diretto e del segnale sfasato produce 0

Polarizzazione: La polarizzazione è fisicamente l'orientamento dell'antenna.

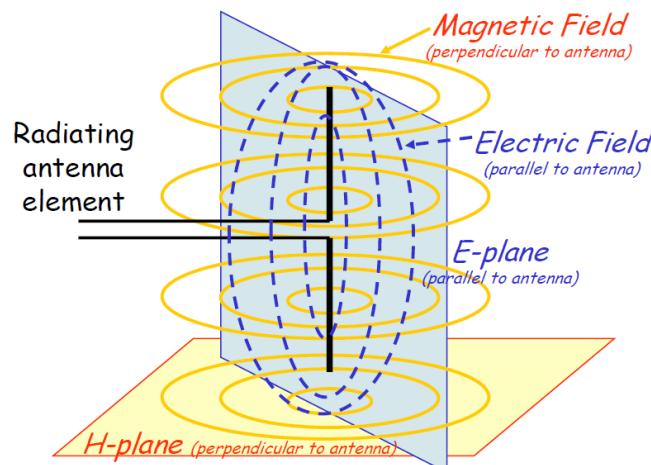
I piani su cui si trasmettono le onde radio sono due:

- Magnetico (perpendicolare all'antenna)
- Elettrico: (parallelo all'antenna).

Questi due piani si autosostengono e viaggiano ruotati di 90 gradi.

Se l'antenna viene posizionata in modo orizzontale è Orizzontale ed il campo elettrico è orizzontale come il pavimento.

La polarizzazione verticale (antenne poste in verticale) è tipicamente utilizzata nelle WLAN. Il massimo grado di trasferimento si ha quando l'antenna del ricevente è polarizzata come l'antenna del trasmittente, altrimenti avremmo una perdita di energia del segnale trasferito, e potremmo rischiare di scendere (proprio a causa di questa perdita) sotto la Signal Detection Limit.



In ambiente chiuso (indoor)

La polarizzazione delle antenne è poco importante, poiché l'onda radio rimbalza su tutti i muri e ci sarà sempre un'onda che rimbalzando arriva alla stessa polarizzazione del ricevente.

In ambiente aperto (outdoor) invece,

La polarizzazione è molto importante; soprattutto in un ambiente come lo spazio cosmico, ad esempio, in cui il segnale deve percorrere grandi distanze in assenza di ostacoli, la polarizzazione è importantissima.

Comportamento Radiofrequenze

Lato ricevente, quello che è possibile captare dinamicamente è la composizione di fasi e frequenze di tanti segnali che si sommano vettorialmente generando interferenze.

Sono i protocolli che riescono a portare ordine in questo sistema (e quindi, utilizzando l'immagine come esempio, riesco a capire quale sia la frequenza gialla, quale quella rossa e quale quella azzurra)

Effetto radiofrequenze in auto:

Simile a gabbia di Faraday, annulla il valore del campo elettromagnetico al di fuori della gabbia stessa.

Un parallelo intuitivo è immaginare l'auto come una barriera fonoassorbente, lo smartphone che cerca di collegarsi "urla" energia che viene abbattuta dall'auto.

Il cellulare si scarica prima perché utilizza più energia per cercare di connettersi al trasmettitore. Effetto che tempesta di microonde chi sta in auto.

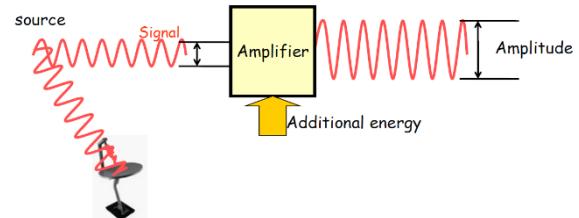
Le onde elettromagnetiche con ampiezza maggiore impiegano più energia per essere prodotte, ma trasportano più energia (quindi dati) al ricevente.

7.3 TRASMISSIONE WIRELESS

7.3.1 Guadagno di segnale (misurato in decibel, dB)

- Aumento dell'ampiezza **M** proporzionale all'energia di trasmissione
Guadagno attivo di energia: energia guadagnata ad esempio

attraverso un amplificatore, che pende in input la sinusoide e restituisce in output la stessa sinusoide variando solo l'ampiezza (attraverso una sorgente di energia es. 



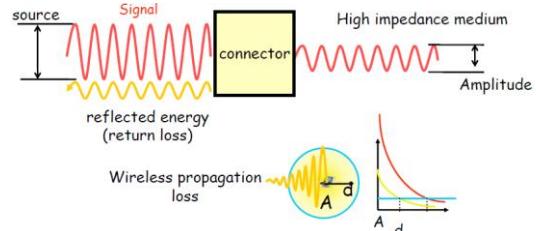
Guadagno passivo di energia: è un altro modo di guadagnare energia, ad esempio attraverso una parabola.

La parabola tende a catturare energia in quantità direttamente proporzionale alla grandezza del disco. L'energia viene catturata e focalizzata in un punto.

Questo è un esempio di guadagno passivo, poiché nessuno aggiunge una quantità di energia additiva come nel caso precedente, ma viene solo sfruttata al meglio l'energia trasmessa dal trasmettitore.

7.3.2 Perdita del segnale: (dB)

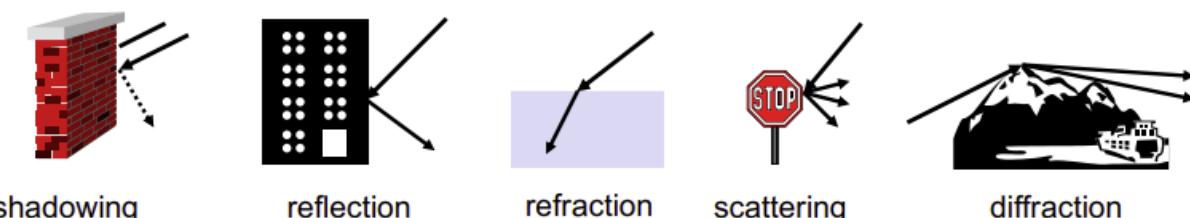
- Diminuzione dell'ampiezza **M** proporzionale allo spreco di energia
Intenzionale: ad esempio attraverso delle resistenze, l'ampiezza del segnale viene ridotta trasformando l'energia in calore.



Ostacoli: è una perdita di segnale 'non voluta', dovuta a ostacoli, ad esempio in caso di nebbia, che rappresenta un ostacolo per le microonde che scaldano le particelle d'acqua di cui è composta la nebbia, facendo ridurre l'energia del segnale (e quindi l'ampiezza), riducendo la distanza che esso può percorrere.

7.3.3 Effetti propagazione del segnale

- Propagazione in spazio libero sempre come la luce (linea retta)
- La potenza ricettiva proporzionale a $\frac{1}{d^2}$ (con d = alla distanza mittente-ricevitore)
- La potenza ricettiva, in più, è influenzata da:
 - Dissolvenza (dipende dalla frequenza)
 - Shadowing
 - Riflessione su grandi ostacoli
 - Rifrazione in base alla densità del mezzo
 - Dispersione su piccoli ostacoli
 - Diffrazione ai bordi



Tutti questi effetti di propagazione del segnale si traducono ad avere i problemi raffigurati nelle seguenti immagini...

A seconda di dove ci poniamo, il segnale ha energia diversa.

Prima immagine: raffigura la propagazione in una zona montuosa. È probabile che si formino dei coni d'ombra come, ad esempio, ai piedi della montagna, che corrispondono a dei punti in cui abbiamo assenza di segnale.

Le striature in foto rappresentano i rimbalzi dell'onda verso l'alto.

Seconda immagine: rappresenta la propagazione del segnale in un edificio, in alcune stanze ad esempio (quelle viola) il segnale non arriva.

Terza immagine: rappresenta la propagazione del segnale in una città come (eg, il segnale di un ripetitore in cima ad un edificio). Abbiamo una buona propagazione nelle strade intorno, tranne i due coni d'ombra dovuti probabilmente alla presenza di alcuni grattacieli che ombreggiano la propagazione del segnale radio.

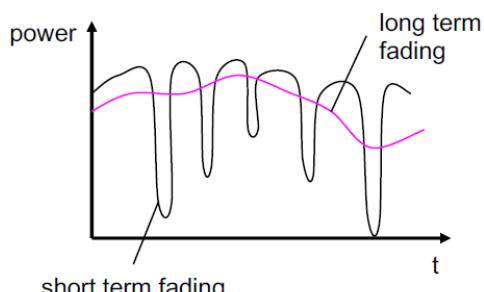
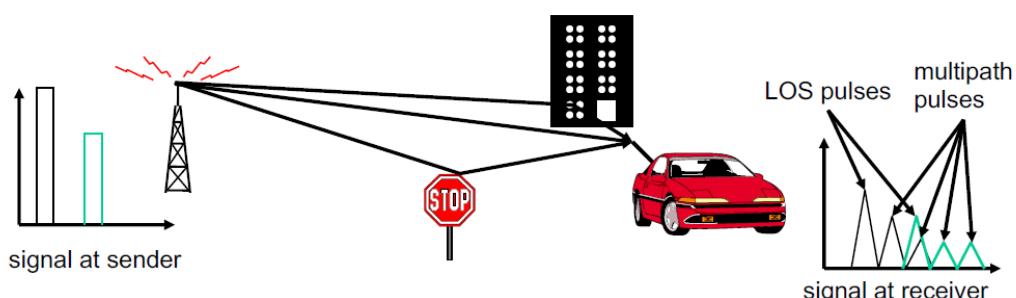
Queste problematiche sono da considerare in fase di progettazione del sistema; altrimenti generano problemi nella fase di funzionamento.

7.3.4 Propagazione multipercorso

Il segnale può percorrere differenti percorsi tra trasmettitore e ricevente, e può subire vari effetti tipo riflessione, diffrazione.

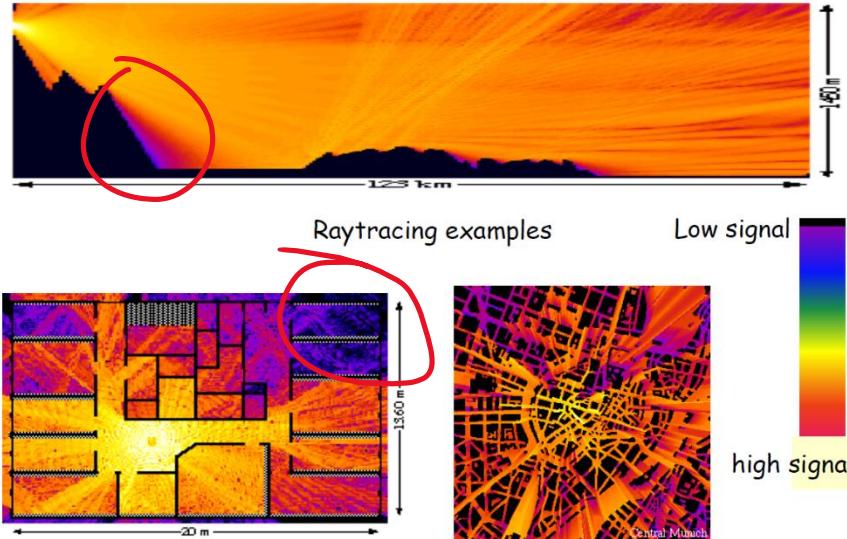
Due segnali che viaggiano verso un'auto (uno che arriva in modo diretto e l'altro per effetto di una riflessione su un grattacielo); arrivano entrambi, ma uno (quello che ha rimbalzato sul grattacielo) arriva con un evidente ritardo di fase.

Se questo ritardo diventa importante c'è il rischio che due segnali consecutivi arrivino sovrapposti grazie all'effetto di sfasamento (l'eco del segnale che fa la strada più lunga arriva a destinazione insieme al segnale diretto, emesso successivamente). Questo effetto di sovrapposizione genera ulteriore confusione. (come avere due persone che parlano, ed ascolto entrambe sentendo solo della confusione dovuta alla sovrapposizione delle due voci)



← Variazione di segnale che è possibile avere spostandosi in una città. Il segnale può variare a seconda della distribuzione del segnale nell'ambiente rappresentato da questa curva.

Questo effetto lo chiamiamo 'fading' (è il nascondersi del segnale). Il segnale scompare e poi riappare. Dobbiamo cercare di superare questo effetto quando utilizziamo la comunicazione attraverso onde radio.



Voltage Standing Wave Radio (VSWR)

Progettando un sistema di comunicazione, ogni volta che mettiamo in collegamento un dispositivo con un altro successivo, dobbiamo fare attenzione al valore di **impedenza nominale** (ovvero alla resistenza della corrente) perché se due componenti hanno delle impedenze diverse si verifica l'effetto chiamato **Voltage Standing Wave Radio (VSWR)**; per evitare ciò dobbiamo fare attenzione che il rapporto di impedenza sia sempre 1 a 1, se non è 1 a 1 abbiamo un effetto di dissipazione di energia (e ciò non è mai positivo poiché abbiamo consumato dell'energia per generare quell'energia persa, senza poi avere degli effetti positivi dal punto di vista della comunicazione).

In un sistema di comunicazione, tutto il blocco (esclusa l'antenna) viene chiamato **Intentional Radiator** (Radiatore intenzionale, IR). Misurando l'energia di un sistema di trasmissione radio, l'energia a cui facciamo riferimento è l'energia che arriva all'antenna. Quest'energia viene calcolata nel sistema come **Intentional Radiator Power Output** (ovvero la potenza di uscita dell'Intentional Radiator), che non è l'energia del trasmettitore, ma l'energia dello stesso meno tutte le perdite causate da collegamenti, connettori, etc..., fino all'ultimo elemento prima dell'antenna.

Un'antenna le cui caratteristiche sono quelle di concentrare le energie in un punto dello spazio (eg parabola) equivale a dare dell'energia a quest'antenna, ma essa non viene diffusa in tutte le direzioni omogeneamente (come una lampadina).

La lampadina è l'equivalente di un **radiatore isotropico**, ovvero un'antenna che emette energia omogeneamente nello spazio. **Se un'antenna concentra l'energia almeno in una direzione preferenziale avremmo una concentrazione di energia in quella direzione maggiore rispetto ad un Radiatore Isotropico.** Il valore **EIRP** (Equivalent Isotropically Radiated Power) è il valore dell'energia di **Intentional Radiator Power Output** che viene consegnata all'antenna equivalente a quell'energia che servirebbe a un radiatore isotropico per generare lo stesso effetto di onda elettromagnetica nella direzione preferenziale. (Energia che dovremmo dare ad un'antenna isotropica affinché generi la stessa energia della parabola)

Quindi:

IR = sistema di comunicazione (antenna esclusa)

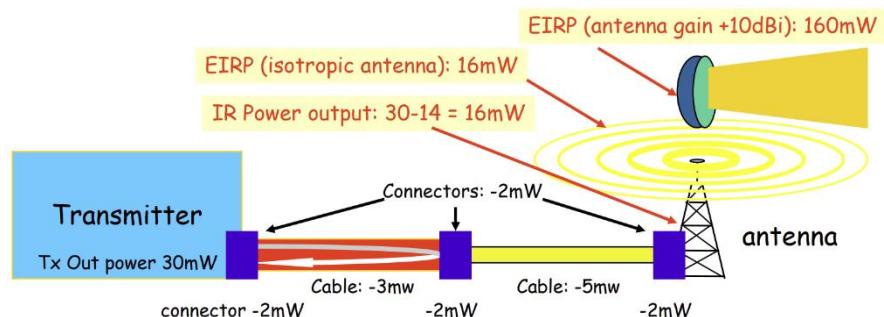
IR Power Output = energia del trasmettitore – tutte le perdite fino all'ultimo elemento prima dell'antenna (che sarebbe proprio l'energia che arriva all'antenna).

Equivalent Isotropically Radiated Power = valore dell'energia IR Power Output che viene consegnata all'antenna, equivalente a quell'energia che servirebbe a un Radiatore Isotropico per generare lo stesso effetto di onda elettromagnetica nella direzione preferenziale.

Esempio EIPO:

Se diamo 16mW direttamente all'antenna isotropica, l'energia irradiata in tutte le direzioni è 16mW.

Se invece diamo 16mW ad un'antenna che focalizza 10 volte l'energia in una direzione preferenziale (sottraendola alle altre direzioni ovviamente) il valore EIRP di questa nuova antenna è il valore che dovremmo dare ad un radiatore isotropico poiché esso possa irradiare **160mW**, e quindi in questo caso il valore EIRP è pari a 160mW.



Quando l'antenna non è isotropica, quindi, EIRP mi dice quale sarebbe l'energia isotropica equivalente.

7.4 PROGETTAZIONE DEL SISTEMA (DAL PUNTO DI VISTA ENERGETICO)

Un sistema di trasmissione (IR) è quindi composto da:

- Un trasmettitore
 - Da cavi
 - Vari dispositivi di connessione che connettono il trasmettitore all'antenna
 - IR Power Output, posto ai bordi dell'IR e destinato all'antenna che genera energia che viene irradiata in base al tipo di antenna utilizzato.
- Una volta generata quest'energia decade velocemente con la distanza, con la legge di propagazione e quest'energia sarà sufficiente a ricevere il segnale radio fino a quando è superiore alla Receiver Sensitivity Limit.

WATT: Unità di misura potenza elettrica

Nella progettazione di sistemi a radiofrequenza l'unità di misura utilizzata è il dB.

Il Decibel (dB) è un'unità di misura per esprimere in modo semplice le perdite di potenza (ma anche i guadagni)

Il dB è nato soprattutto per esprimere le perdite che, possono arrivare a 11, 12, 13 cifre dopo la virgola, diventando difficili da gestire per eseguire dei calcoli (Il dB è una scala logaritmica e non lineare

Il dB mappa i mW ma su una scala logaritmica.

Il dB misura, in scala logaritmica, la forza relativa tra due segnali

Importante: Un valore in dB non è mai una quantità di energia pura, ma rappresenta sempre una differenza di potenza (energia) tra due riferimenti. Serve a marcire le differenze tra due livelli di potenza in scala logaritmica.

“A qyabti cirrusoibde yb segbake firte 10dB” ← Non vuol dire nulla, non ha senso

“Quanto è forte un segnale a 10dB rispetto ad un valore iniziale x” ← OK

$$\log_{10}(X) = Y \Leftrightarrow 10^Y = X$$

Crescita Esponenziale ↓

- $1 = 10^0, \log_{10}(1) = 0$
- $10 = 10^1, \log_{10}(10) = 1$
- $100 = 10^2, \log_{10}(100) = 2$
- $1000 = 10^3, \log_{10}(1000) = 3$

Crescita lineare unità “Bel” (B)

Un valore in dB positivo rappresenta un guadagno di potenza, un valore in dB negativo rappresenta una perdita di potenza.

$$\text{Power Difference (Db)} = 10 \times \log\left(\frac{\text{Power Rx (W)}}{\text{Power Tx (W)}}\right)$$

Di base il rapporto è minore di 1 poiché di solito quello che trasmettiamo si attenua con la distanza.

Il logaritmo produce un valore negativo tra 0 e $-\infty$

Questa funzione trasforma in Bell il rapporto tra segnale ricevuto e il segnale trasmesso.

Moltiplicando poi questo valore per 10 ottengo un valore in dB.

Se power difference (dB) negativo rappresenta una perdita, altrimenti se positivo vuol dire che è stato collocato un amplificatore nel sistema

La scala a destra è utile a rappresentare una differenza di potenze tra due livelli

$$1 \text{ dB} = 1/10 \text{ Bel}$$

Passare da 0 Bel a 1 Bel corrisponde a fare un salto di 10dB

Ogni volta che sommiamo 10dB stiamo passando da un valore Bel ad un valore successivo

Sommare 10dB su scala logaritmica equivale a moltiplicare per 10 i valori su scala lineare.

Se ho 1mW ed aggiungo 10dB vuol dire che amplifico quel 1mW trasformandolo in 10mW.

Se ho invece 10mW ed applico 10dB di guadagno, amplifico i 10mW trasformandoli in 100mW.

Vantaggi di dB:

- Es. Un segnale trasmesso a [TX] 100mW viene ricevuto a [RX] 0.000005 mW
 - Differenza di potenza (dB) = $10 \times \log\left(\frac{RX}{TX}\right) = 10 \times \log\frac{0.000005 \text{ mW}}{100 \text{ mW}} = -73$
 - Un segnale trasmesso a 100mW viene ricevuto con un guadagno di -73 dB
- Es. Un segnale trasmesso a 100mW viene ricevuto a 0.000005mW, poi viene amplificato ($\times 100$) a 0.0005mW
Il segnale viene trasmesso a 100mW e viene ricevuto con un guadagno $-73 + 20 = -53 \text{ dB}$

-3 dB	1/2 potenza in mW (/ 2)
+3 dB	2x potenza in mW (* 2)
-10 dB	1/10 potenza in mW (/ 10)
+10 dB	10x potenza in mW (* 10)

Esempio pratico:

Un segnale Tx a 100mW, perdita di **-3dB** per cavo, amplificato con un guadagno di **+10dB**

$$100\text{mW} / 2 (-3\text{dB}) = 50\text{mW} \cdot 10 (+10\text{dB}) = \text{500mW IR Power output}$$

Un segnale Tx a **30mW** viene ricevuto da un'antenna come **6mW (1/5 della potenza Tx)**

$$\text{IR guadagna} = 30\text{mW} / 10 = 3\text{mW} \cdot 2 = 6\text{mW}$$

$$\text{Guadagno IR} = -10\text{dB} + 3\text{dB} = \text{-7dB}$$

dBm = dB-milliWatt, unità di misura della potenza del segnale

Assumiamo che 1mW = 0dBm

Attenzione:

“A quanto corrisponde un segnale forte 10dBm” ← ha senso solo se ci sta la m

7.4.1 mW – dBm: tabella di conversione

dB_i = dB-isotropic viene utilizzato per rappresentare guadagni o perdite da parte delle antenne (ha come riferimento normativo l'antenna isotropica, che spalma il segnale in tutte le direzioni, a forma sferica)

-40 dBm	-30 dBm	-20 dBm	-10 dBm	0 dBm	+10 dBm	+20 dBm	+30 dBm	+40 dBm
100 nW	1 μW	10 μW	100 μW	1 mW	10 mW	100 mW	1 W	10 W

7.5 ANTENNE

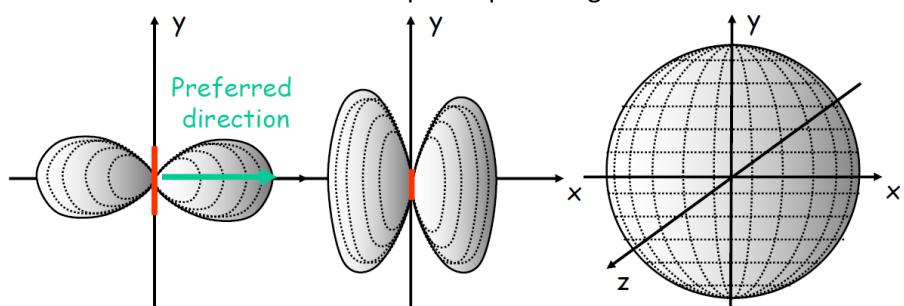
La forma circolare e più sferica possibile è ottenuta con la più semplice delle antenne, il **Dipolo**, che è un segmento conduttore con due poli + e - che

oscillano in modo sinusoidale a seconda della variazione di carica applicata. Produce una corrente sinusoidale che genera l'onda eletromagnetica che si irradia nell'ambiente con una forma che assomiglia a quella di una ciambella (irradia poche onde in verticale).

Alcuni Dipoli tendono ad irradiare un po' verso l'alto ma un po' meno a destra e sinistra (Dipolo a basso guadagno, figura centrale) e altri molto schiacciati che irradiano poco verso la coordinata Y ma irradiano molto sull'asse X (Dipolo ad alto guadagno, figura a sinistra).

Si dicono ad alto guadagno poiché se ci poniamo sull'asse delle X veniamo colpiti da più energia. La

direzione nel quale un'antenna spara la massima componente energetica si chiama **direzione preferenziale**. I dB_i possono essere utilizzati per calcolare le perdite o il guadagno di energia tramite antenne.



7.5.1 Antenne omnidirezionali

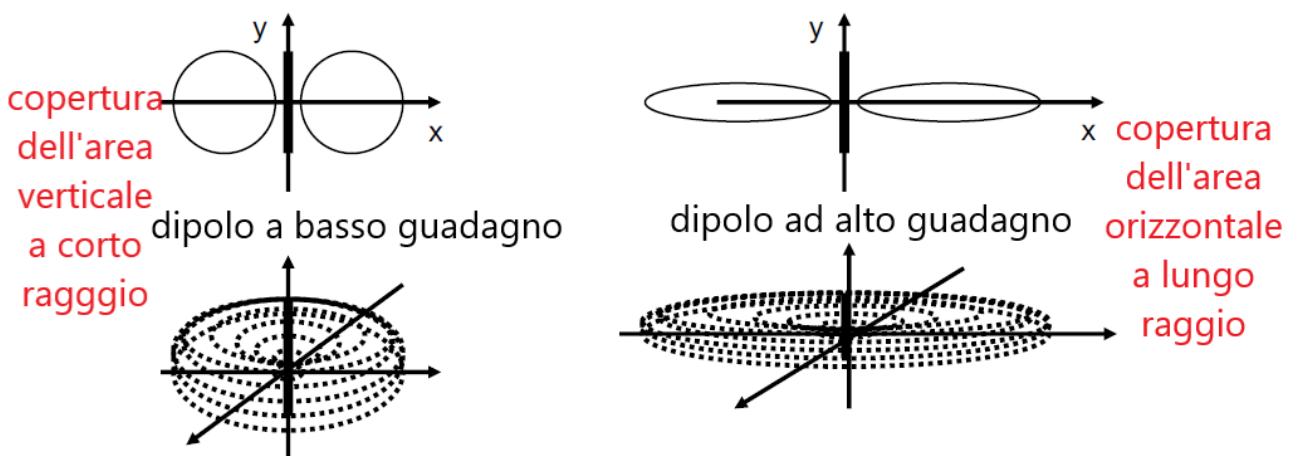
Le antenne Omnidirezionali sono le antenne che assomigliano di più all'isotropico (e quindi che trasmettono il segnale in tutte le direzioni possibili).

L'antenna che utilizziamo di più (che è anche un esempio di antenna Omnidirezionale) nella vita di tutti i giorni è il dipolo.

La direzione preferenziale è il piano X e il piano Z (Emissione a ciambella)

È il tipo di antenna più utilizzata: più semplice, omnidirezionale e più facile da realizzare (per alcuni aspetti).

Rappresentazione grafica dipolo ad alto e basso guadagno



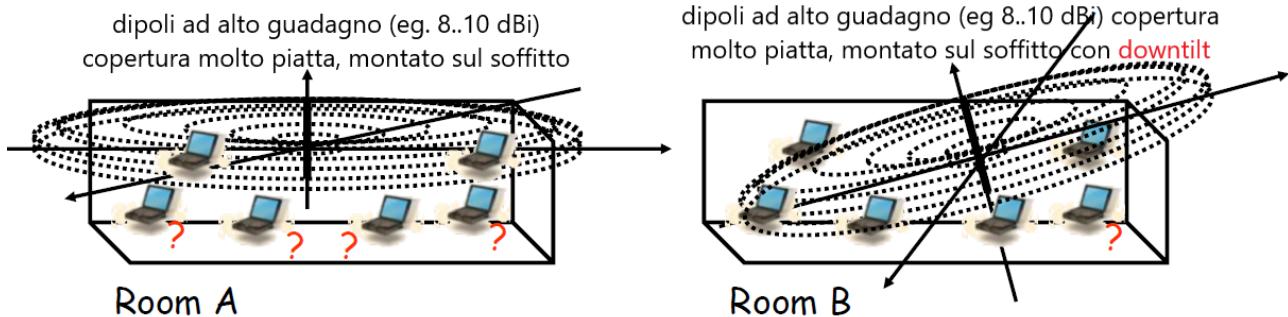
I due tipi di dipolo hanno delle applicazioni differenti in base a ciò che abbiamo bisogno di realizzare

Inclinando l'antenna, otteniamo un'antenna tilt, ovvero un'inclinazione da dare all'antenna in modo da avere una migliore copertura.

Inclinando l'antenna verso il basso otteniamo un Down Tilt.

Quando dobbiamo coprire un'area con il WiFi, dobbiamo porre attenzione al posizionamento delle antenne

Esempio Tilt



Anche se comunque ormai i router attuali hanno più di un'antenna; nei router di ultima generazione è utilizzata una tecnologia chiamata **MIMO**, questa fa in modo che ogni antenna generi un segnale che ottimizzi l'emissione verso il client di destinazione. Questa tecnologia 'segue' il client se si sposta da una stanza all'altra come se le antenne si muovessero per offrire il segnale a ciambella migliore, **spostando le fasi** e ottenendo la massima componente risultante nella direzione giusta.

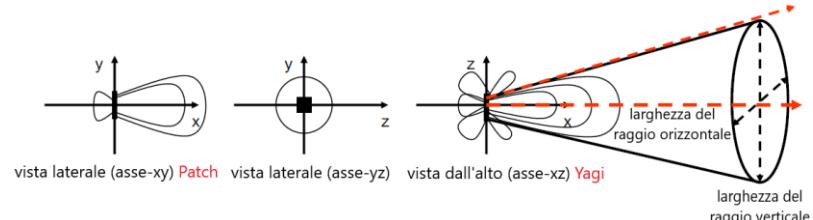
7.5.2 Antenne semi-direzionali

Le antenne semi-direzionali trasmettono di più in una direzione. (utilizzate se non ci importa di trasmettere il segnale da tutte le parti in modo omogeneo, e da una parte non vogliamo trasmettere)

Assomigliano a delle torce elettriche  (direzione della luce = direzione delle onde radio, a seconda di dove la punto)

Alcune antenne semi-direzionali sono:

- Patch (antenne piatte montate sui muri)
- Panel (antenne piatte montate sui muri)
- Yagi (asta con denti sporgenti)



Le antenne Yagi hanno un cono di ampiezza maggiore.

7.5.3 Antenne Molto-Direzionali (Highly-directional)

Sono antenne che concentrano tutta l'energia radio in un cono di ampiezza (in gradi) molto piccola.

- Parabola



- Grid

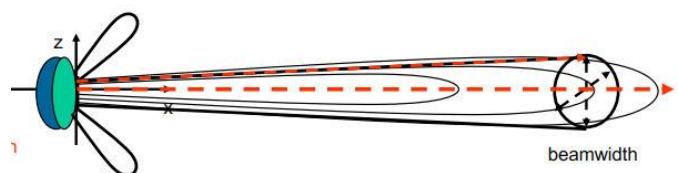
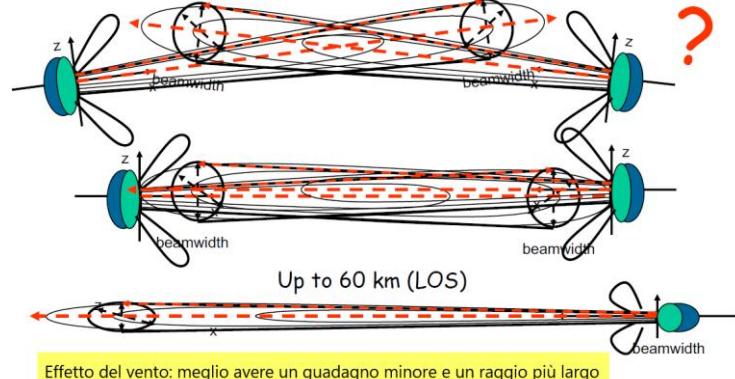


Tabella confronti direzione e coni di emissione:

Tipo di antenna	Larghezza del raggio	
	orizzontale	Verticale
Omnidirezionale	360°	7° ... 80°
Patch/panel	30° ... 180°	6° ... 90°
Yagi	30° ... 78°	14° ... 64°
Parabola	4° ... 25°	4° ... 21°

Uso comune: canale Point-to-Point

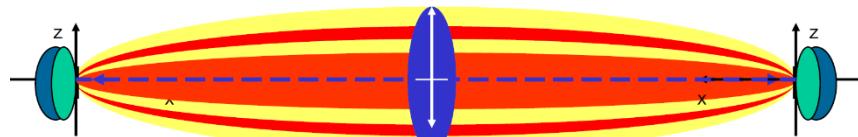
È molto importante centrare correttamente le antenne (vincolarle bene in modo che non si muovano e che il vento non faccia oscillare la parabola), altrimenti i raggi possono andare fuori bersaglio e il segnale non arriva.



Line Of Sight = linea di vista
(← linea blu nel disegno, tra del due antenne)

Immaginiamo di avere due antenne. Queste due funzionano meglio nel trasferire energia quando la Line Of Sight è libera da ostacoli (mettendo in mezzo un ostacolo si attenua l'energia che passa, disturbando la comunicazione).

La **Fresnel Zone (FZ)** è la zona centrale intorno alla Line Of Sight nella quale passano la maggior parte dei segnali di tipo additivo sul ricevente. (il disco blu nell'immagine)



La Fresnel Zone deve quindi essere lasciata libera da ostacoli, altrimenti viene persa energia inutilmente. La regola che devono seguire i progettisti di sistemi radio è calcolare il raggio del disco blu con le appropriate formule e fare in modo che rimanga libero.

Il raggio della Fresnel Zone dipende da alcune costanti, ma anche la distanza tra le due antenne e dalla frequenza utilizzata dalla comunicazione radio.

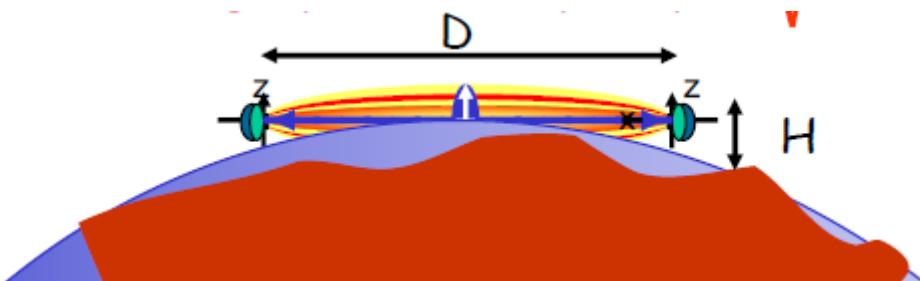
Non incide il tipo di antenna (due parabole ad esempio, non hanno una Fresnel Zone più piccola rispetto a due dipoli).

La Fresnel Zone non è rilevante per scenari indoor (a causa della riflessione)

Per calcolare il raggio della FZ bisogna lasciare un 60% di raggio non ostruito

- $R_{60\%} = 43.3 \times \sqrt{d/4f}$
- $R_{100\%} = 72.2 \times \sqrt{d/4f}$

Molto spesso la Fresnel Zone potrebbe essere ostruita dalla curvatura terrestre, in caso di grandi distanze



La soluzione è semplice, basata spostare (verso l'alto) le parabole in modo che abbiano il disco blu libero. Per questo, nella pratica, vengono utilizzati dei tralicci (ripetitori TV/telefonia)

Per calcolare l'altezza: $H = \left(43.3 \times \sqrt{\frac{d}{4f}} \right) + \frac{d^2}{8}$

7.5.4 Antenne settorizzate

Utilizzando delle antenne direzionali o semi-direzionali è possibile coprire tutti i 360° dell'ambiente intorno dividendo i raggi di emissione radio in 6 diversi raggi separati (ognuno di essi viene chiamato settore).

Ogni stecca nella foto è un'antenna patch, emette segnale su un angolo aperto di 30°, questi tralicci vengono utilizzati nella copertura della rete cellulare.

Con il cellulare quindi, quando utilizziamo la rete, usiamo solo una delle 6 antenne e se qualcuno utilizza un'antenna affianco può comunicare e ricevere i suoi dati su un'emissione radio che, anche se ha la stessa frequenza nelle due emissioni, non fa conflitto, perché sono due direzioni diverse (e quindi due settori diversi con due antenne diverse).



Le onde radio in una direzione non si sovrappongono con le onde radio in un'altra (le zone di gestione delle antenne non si sovrappongono).

La settorizzazione delle antenne permette di utilizzare più volte la stessa frequenza verso più clienti a seconda di dove sono distribuiti nello spazio.

Questo è importante soprattutto per la rete cellulare, perché chi la gestisce ha pagato lo stato per poter utilizzare quelle determinate frequenze, e con quelle poche e costosissime frequenze deve cercar di far trasmettere il massimo grado di comunicazione ai propri clienti, perché facendoli comunicare con il gestore (Provider di telefonia) ha un guadagno. In questo modo moltiplicano la possibilità di comunicare sulla stessa frequenza.

Non accade mai che due settori adiacenti siano lo stesso canale (per evitare dei problemi nel caso in cui un host si pone in mezzo a due settori). Queste sequenze di emissione vengono fatte 'colorando' i segnali in modo diverso (problema del Frequency Planning, pianificazione delle frequenze ("Stabilire i colori") all'interno del sistema cellulare, opportunità che abbiamo utilizzando questi array di antenne).

Come fa il cellulare a capire quale frequenza utilizzare?

Il telefono comunica con questa infrastruttura utilizzando una sequenza pilota che serve per fare tutta la parte amministrativa (frequenza unica che utilizzano tutti i telefoni), ascoltano quella frequenza e capiscono quali di queste sono a disposizione e in quali punti.

A questo punto, o ricevono dal ripetitore che fa lo scheduling, la frequenza da utilizzare a quel telefono, o può scegliere il telefono quale frequenza utilizzare in base alle frequenze che vengono comunicate dal ripetitore come libere. Una volta fatta la scelta, i Gb di dati trasmessi viaggiano su una frequenza verso una di queste antenne in modo univoco (utilizza solo un cellulare quel canale).

Se un host si sposta, prima cambia antenna mentre si sposta (o alla stessa frequenza, o può anche cambiare). Se un host si sposta molto dal ripetitore, si aggancia al più vicino ripetitore successivo, questa operazione si chiama Hand Off o Hand Over (Passaggio di mano).

7.5.5 Diversity scheme

I router moderni hanno gruppi di due o più antenne, queste sono posizionate tutte a distanza $\frac{\lambda}{2}$.

Questa distanza non è casuale, ma è stabilita in modo che a quella distanza non è possibile avere opposizioni di fase su tutte le antenne (avremo sempre un segnale che non è in opposizione di fase).

Sempre per composizione di fase, nella [Diversity scheme](#), estrarremo il massimo della componente additiva del segnale (con un processore) in modo da capire sempre meglio l'energia che arriva anche in condizioni non ideali.

In caso di rumore, ognuna delle antenne sente un rumore differente con un segnale comune (il messaggio); Estraiamo quindi da questo segnale la parte comune a tutte le antenne e questo sarà il messaggio senza rumore.

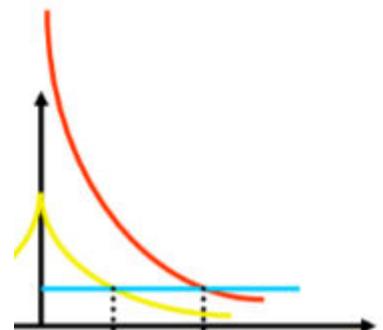
7.5.6 Perdita di percorso

$$\text{Perdita (dB)} = 36.6 + (20 \times \log_{10} F) + (20 \times \log_{10} D)$$

Questa formula ci dice in dB quanto segnale perdiamo in funzione della distanza percorsa da quel segnale, che dipende dalla frequenza e dalla distanza.

F è in Mhz, D è in miglia.

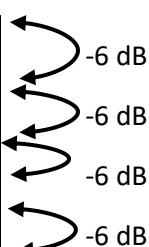
La costante 36.6 è un valore di perdita in ambiente terrestre, nel caso di valutazioni nello spazio aperto (tra Marte e Giove) la costante moltiplicativa è pari a 32.4 (valore di perdita inferiore).



Questa formula caratterizza la forma che ha la curva rossa nell'immagine (legge di propagazione) se otteniamo **il risultato in dB, qualunque potenza parta dalla trasmissione, dalla forma della curva (che dipende da frequenza e distanza)** riusciamo a capire di quanto decade il segnale.

7.5.6.1 Regola dei 6 dB

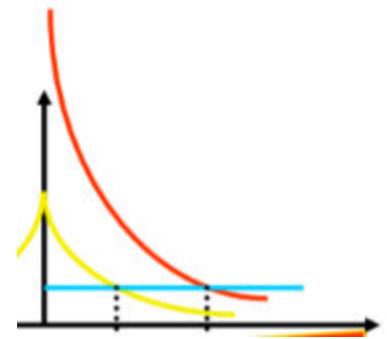
100 metri	-80.23 dB
200 metri	-86.25 dB
500 metri	-94.21 dB
1000 metri	-100.23 dB
2000 metri	-106.25 dB
5000 metri	-114.21 dB
10000 metri	-120.23 dB



Quando raddoppiamo la distanza dal trasmettitore, il segnale che riceviamo si riduce di un fattore pari a circa $\frac{1}{4}$; questo è dovuto alla legge di propagazione, che dice che per raddoppiare la distanza percorsa dal segnale dobbiamo quadruplicare l'energia del segnale. (Il fattore decadimento energia radio nell'ambiente è quadratico rispetto alla distanza)

7.5.7 Link Budget

Il Link budget è il segnale in eccesso ricevuto dal ricevente, oltre a quello necessario della Signal Detection Limit (in modo da comprendere l'informazione). Sarà quindi pari al **segnale ricevuto** (curva rossa) meno il valore della **Signal Detection Limit** (linea blu).



$$\text{received power} \text{ (in dBm)} - RS \text{ (in dBm)}$$

Es.

$$RS = -82 \text{ dBm}, \text{ received power} = -50 \text{ dBm}$$

$$\text{Link budget} = -50 - (-82) = +32 \text{ dBm}$$

Ciò significa che il segnale ha un margine di **+32 dB** prima che diventi impraticabile.

Dobbiamo progettare il sistema in modo che questa quantità sia positiva (ovvero il segnale ricevuto sia maggiore della linea blu)

Se il link budget è positivo vuol dire che il canale funziona (ho comunicazione).

Dobbiamo fare in modo che anche questa quantità sia abbastanza grande (in modo da avere un margine abbastanza ampio) poiché con un margine troppo piccolo (tipo 1dB) in caso di interferenza rischiamo di avere un segnale sotto la Signal Detection Limit.

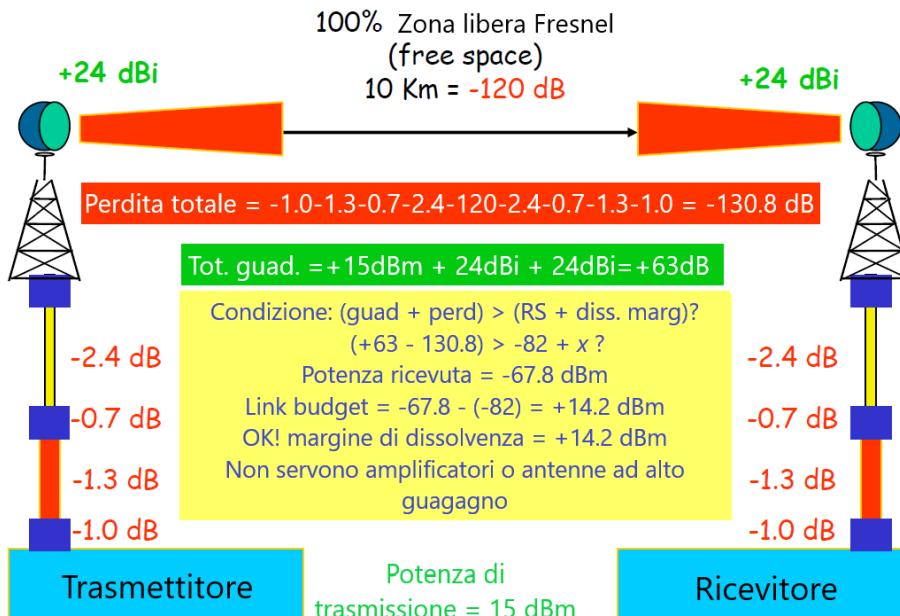
Quindi bisogna avere un Link Budget almeno pari a quelli che definiamo essere i **System Operating Margin** cioè i margini operativi del sistema.

Graficamente l'energia in eccesso (valore tra curva rossa e linea blu) deve essere almeno pari a un valore chiamato System Operating Margin che deve garantire la funzionalità del sistema (eg, in qualunque condizione atmosferica).

Il valore del System Operating Margin è almeno pari a [10,20] dB.

Il System Operating Margin può essere pari a 10 in un sistema in cui c'è poca interferenza. Altrimenti conviene lasciare un margine maggiore o uguale a 20 dB.

7.6 ESEMPIO: ESERCIZIO DI APPLICAZIONE DEI VARI ARGOMENTI DI QUESTO CAPITOLO



Abbiamo un ricevente e un trasmittente che devono comunicare tra di loro.

Trasmettitore: è una componente attiva
Possiamo decidere quante energia far trasmettere
+ aumentiamo l'energia + consumiamo
+ energia \Rightarrow + potenza di trasmissione.

L'energia del trasmittitore viaggia su tutti i cavi e i connettori, fino ad arrivare all'antenna, perdendo dell'energia.

L'energia che arriva all'antenna (IR PwOutput) sarà 15dB – tutte le perdite.

Nell'antenna abbiamo un guadagno, dovuto al tipo di antenna avente una direzione preferenziale (quindi guadagno di energia) il guadagno è di +24dBi

Il guadagno dell'antenna è un guadagno passivo e non attivo, poiché è dovuto al modo in cui è strutturata l'antenna (che concentra l'energia in un punto).

L'antenna a destra cattura il segnale che arriva da sinistra, guadagnando altri 24dBi. In questo modo si utilizza il guadagno passivo di queste due antenne per guadagnare energia. L'energia arrivata all'antenna scende verso il ricevente, e cavi e connettori intermedi comportano delle perdite.

Ricevente: ha una sensibilità pari a -82 dBm. Cioè, se il segnale ricevuto è pari a -82dBm, qualcosa riesce a capirla; in alternativa se ricevesse -83dBm sarebbe sotto la linea blu \Rightarrow non capisce.

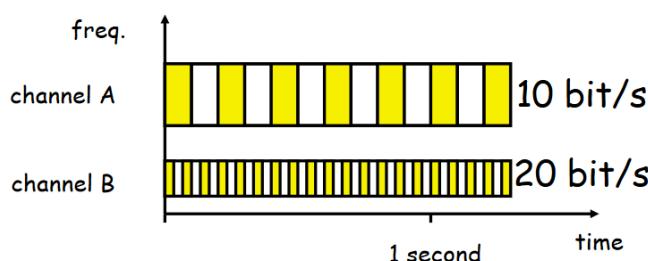
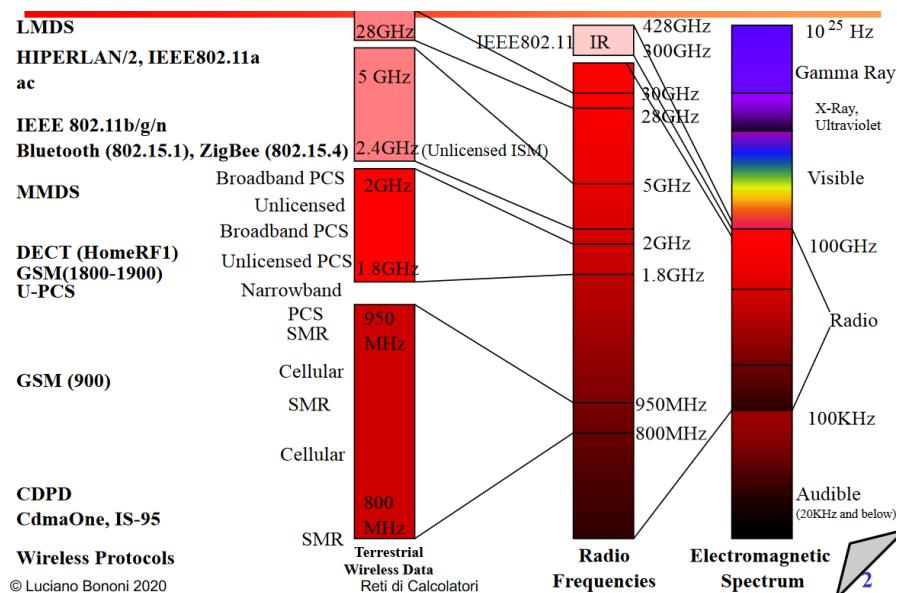
Possibile domanda esame:

- Se l'energia del segnale ricevuto dal ricevente fosse minore della RS (sensibilità del ricevente), cosa possiamo fare per risolvere il problema in modo che il ricevente possa avere un'energia sufficiente a capire l'informazione del segnale trasmesso?

Vari modi, potremmo aumentare l'energia fornita dal trasmettitore (in modo da avere un guadagno attivo). Utilizzare delle antenne che hanno un guadagno passivo maggiore e cercare di guadagnare dell'energia utilizzando le antenne, oppure potremmo cercare dei cavi e connettori che comportino delle minori perdite di energia.

8 WIRELESS

Intorno ai 100KHz inizia lo spettro radio, oltre i 100GHz siamo negli infrarossi, salendo si incontrano gli ultravioletti, raggi X e raggi gamma. Le frequenze radio utili si trovano tutte tra 100KHz e 100GHz. Esempi: tra 1.8 – 2.4 GHz abbiamo il dual band. Tra 2.4-5 abbiamo WiFi e Bluetooth. Molte porzioni di frequenza hanno più utilizzi in parallelo, l'assegnazione dello spettro alle varie tecnologie è difatti una vera problematica.



Se voglio trasmettere dei bit usando le onde radio per rappresentarne i valori, cosa fa differenza prestazionale nelle diverse tecnologie? Perché ho dei canali radio che funzionano a 100Mb/s ed altri a 1Mb/s?

Sicuramente non la velocità del segnale (le onde viaggiano alla stessa velocità, cioè quella della luce, ovvero $\approx 300.000 \text{ km/s}$);

Potrebbe essere, in alcuni casi, la quantità di spettro che uso per comunicare; il tempo durante il quale descrivo la sequenza di bit è il parametro cercato, cioè quello che fa differenza.

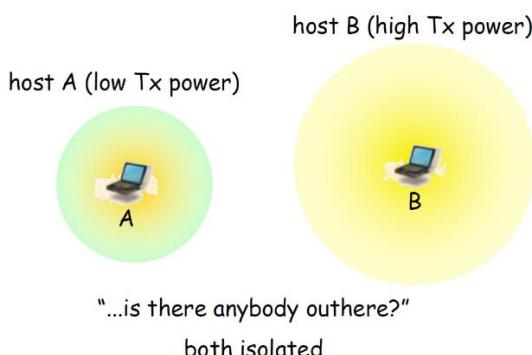
In particolare, la differenza si trova nel tempo durante il quale le onde radio descrivono la sequenza di bit. Questo parametro è noto come "tasso di trasmissione" o "bitrate".

Se rappresento i bit a velocità sostenuta ne trasmetto di più ma è più difficile riceverli (anche a causa delle altre interferenze del mondo radio).

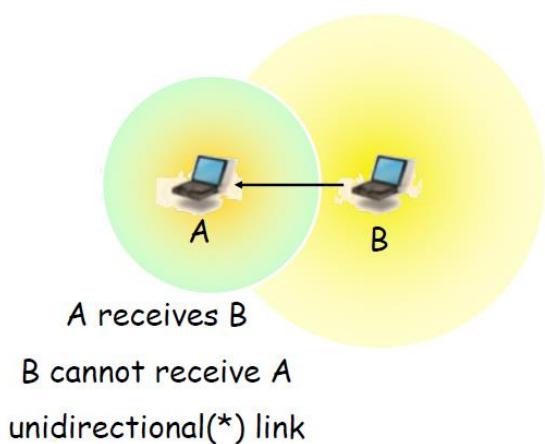
Le tecniche di codifica di bit realizzano un compromesso tra ciò che si vuole trasmettere bene e ciò che si vuole trasmettere velocemente. Ci possono essere diverse situazioni, a seconda dei range (detti dalla potenza trasmittiva).

8.1 COLLEGAMENTI POSSIBILI DURANTE LA CREAZIONE DI CANALI RADIO

Caso 1, sia A che B isolati

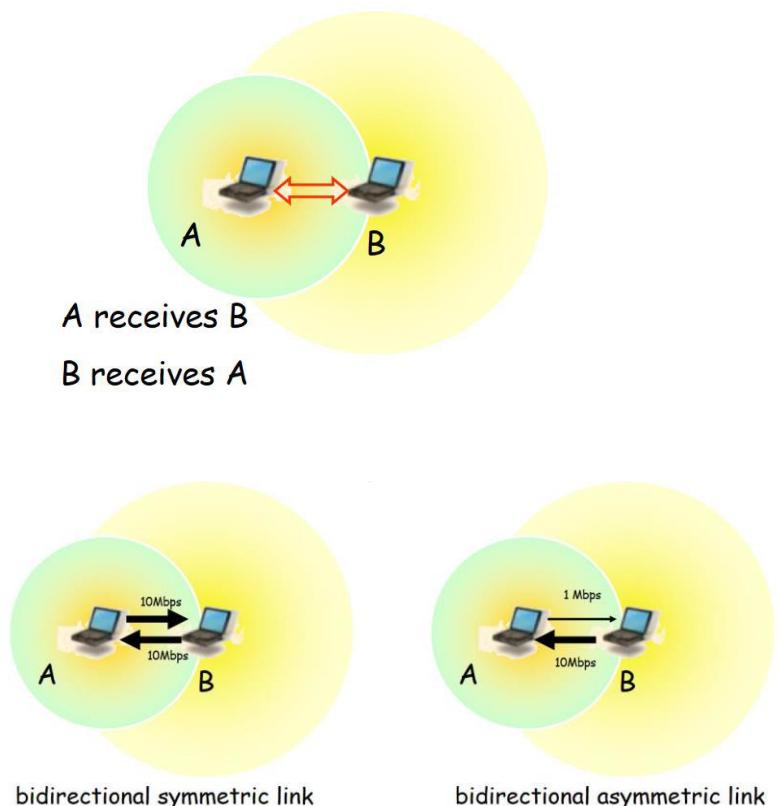


Caso 2, A riceve da B ma B non può ricevere da A, collegamento unidirezionale



Casi 3 e 4, A e B riescono a riceversi a vicenda (link simmetrico), ma possono esistere due tipi

- Se i bitrate nominali $A \rightarrow B$ e $B \rightarrow A$ sono uguali, abbiamo un link bidirezionale simmetrico
- I bitrate nominali sono diversi ed avrò quindi un link bidirezionale asimmetrico.



8.2 CREAZIONE DEI CANALI RADIO

La creazione di diversi canali radio può essere di 3 tipi:

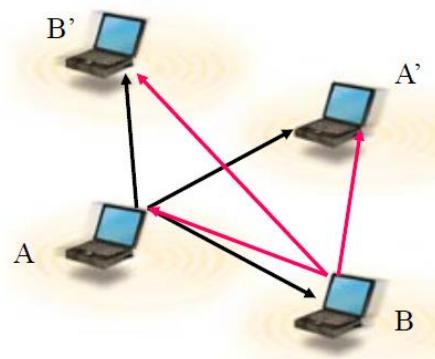
1. Narrow band, banda stretta
2. Frequency hopping spread spectrum
3. Direct sequence spread spectrum

Nel narrow band, diversi canali radio potranno coesistere nello stesso spazio/tempo perché hanno una frequenza radio diversa. La differenza tra la frequenza X e Y rende possibile, lato ricezione, di creare un oggetto detto 'filtro', capace di isolare l'energia che arriva su una certa frequenza radio.

In particolare, l'energia catturata da una certa antenna è la somma delle energie entranti, ma il filtro (passabanda) riesce ad isolare ed estrarre solamente l'energia dal segnale con la frequenza voluta.

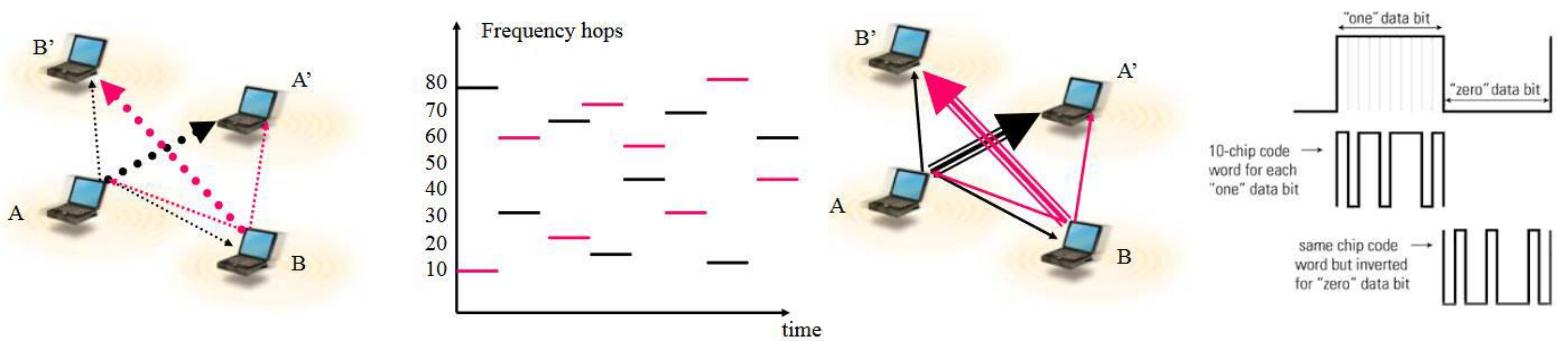
Narrow band si chiama così perché utilizza una sola frequenza. Le collisioni vengono evitate perché riusciamo ad estrarre dal canale che ci interessa le informazioni necessarie, filtrando tutto il resto delle frequenze radio presenti nello spazio/tempo. Questo rende il narrow band particolarmente efficiente e adatto per comunicazioni specifiche e mirate, evitando interferenze e collisioni con altre trasmissioni.

Le collisioni radio si hanno quando si vuole usare una frequenza per comunicare tra più di 2 interlocutori.



Nelle altre 2 tecnologie troviamo la dicitura spread spectrum, essa sta ad indicare che il canale radio non è rappresentato da una frequenza singola e fissa ma da uno spettro di frequenze. Nel frequency hopping abbiamo una sequenza di salti sincroni, generati tramite dei pattern pseudocasuali. I canali utilizzano tante frequenze narrow band soltanto tra i pattern. Qual è il vantaggio rispetto ad usare 2 narrow band? Col frequency hopping è più difficile intercettare la comunicazione, e se c'è interferenza sul canale basta saltare quella frequenza (oppure usarla poco dopo l'hop). È difficile da implementare perché bisogna fare salti perfettamente sincronizzati tra chi trasmette e chi riceve. Il pattern di salti viene generato con una funzione di hashing dell'indirizzo/chiave/parola deciso da chi trasmette. Pattern deriva da un algoritmo, la chiave va precondivisa. A deve comunicare funzione + seme ad A', stessa cosa B a B'.

Direct sequence: ogni bit che voglio comunicare è rappresentato da un pattern di chip (valori binari). Per rappresentare 0 oppure 1 si usa un pattern di chip. La codifica di un bit a uno corrisponde alla trasmissione di chip, quando si vuole trasmettere 0 si trasmette lo stesso chip ma capovolto. Ogni chip viene codificato e trasmesso su tutto lo spettro radio della banda a disposizione. Il bit è quindi su tutta la banda. L'interferenza che si crea viene annullata grazie alla presenza di pattern/sequenze di bit che rappresentano ogni bit, questa tecnica viene detta "*a divisione di codice*".



La scelta del codice (sequenza di bit usata per rappresentare i bit) da A ad A' è diversa rispetto al codice da B a B', i due diversi codici fanno in modo che chi riceve i segnali, anche se vede sovrapposizione/interferenza distruttiva, riconosca (grazie al codice), il valore del bit che a lui era dedicato. L'interferenza, pur distruggendo i segnali, non è comunque abbastanza potente da permettermi di sbagliare codifica. Il codice deve differire per ogni copia di interlocutori. Ogni codice diverso da luogo ad un canale diverso. Tutti questi canali vengono trasmessi sulla stessa finestra di frequenza. L'infrastruttura wireless creata attraverso l'utilizzo di queste tecnologie consiste in tanti AP (Access Point), cioè punti di accesso che permettono agli MT (Mobile Terminal) di entrare nel range di copertura dell'AP.

Le reti possono anche non avere AP, in questo caso si parla di reti ad hoc, cioè reti in cui i nodi si trovano in grado di comunicare tra loro senza aver predisposto la creazione di quella rete, cioè senza averne pianificato l'infrastruttura. Sono reti quindi senza infrastruttura.

Un esempio di rete ad hoc è il bluetooth; le reti ad hoc sono "antagoniste" delle reti ad infrastruttura, ovvero quelle reti che richiedono componenti cablate (backbone) e AP.

Per fondere il mondo mobile/wireless col mondo cablato bisogna realizzare il bridging

Per fare questo l'AP ha 2 stack, wireless e cablato (quindi anche 2 interfacce di rete). Per bridging si intende quando i dati nel mondo internet vengono tradotti nel formato/protocolli wireless e viceversa.

Grazie al bridging è possibile avere protocolli wireless differenti da quelli del mondo internet (tecnologie, principi di funzionamento e metodologie sono diverse).

Dialogando wireless, se ho bisogno di dialogare con nodi in Internet, ho da qualche parte una bridging func. realizzata da un nodo intermedio (AP, wireless router o simili). Integrare i due mondi richiede un adattamento dei protocolli e dei nodi che facciano da collante architettonico.

Tornando alle modalità di creazione del canale: voglio avere più comunicazioni possibili tra coppie/gruppi di interlocutori (quindi reti consistenti) Questo è possibile tramite il multiplexing, cioè più comunicazioni di più host utilizzano le stesse risorse. (Multiplexing: utilizzi multipli di un mezzo condiviso). Il multiplexing può essere fatto in 4 dimensioni:

1. Spazio

2. Tempo

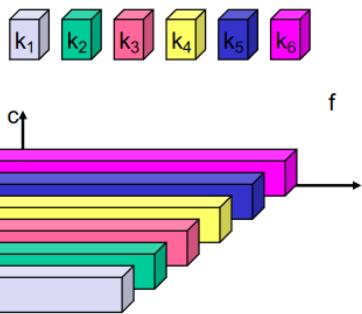
3. Frequenza

4. Codice

8.3 DESCRIZIONE DEL MULTIPLEXING

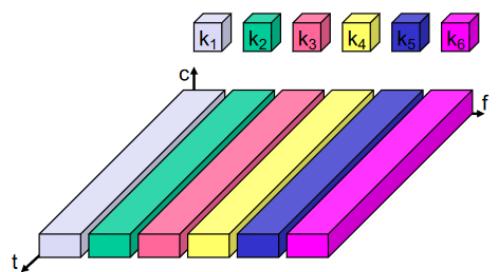
8.3.1 Per frequenza

Ogni canale ha una fetta di frequenze diverse, il canale viene scelto scegliendo le frequenze.



8.3.2 Per tempo

I canali usano tutte le frequenze ma solo per un intervallo Δt limitato. Dopo che un canale ha finito c'è un punto di vuoto e subito dopo inizia il successivo. Ogni canale di rete che usa un canale diverso avrà il proprio intervallo di tempo nel quale può trasmettere e quando può trasmettere usa tutto lo spettro radio.

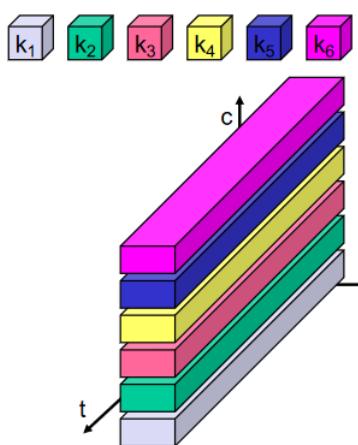


8.3.3 Tempo + frequenza

Abbiamo una certa frequenza disponibile per un certo intervallo di tempo.

Un canale (di un certo colore) è inteso come sequenza combinata di salti nelle frequenze a intervalli di tempo costante.

Salvi di frequenza (o intervalli di frequenza) nel tempo, sincroni e coordinati



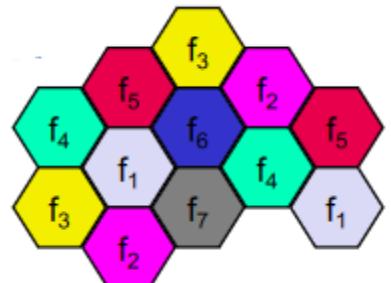
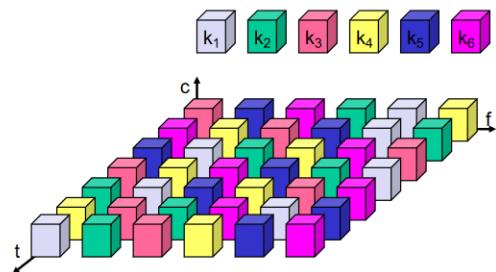
8.3.4 Per codice

← Ogni canale ha un codice diverso (chipping sequence, usata per rappresentare il valore di un bit), i canali usano lo stesso spettro sovrapposto (parlano tutti assieme nella stessa frequenza) e allo stesso tempo.

Ciò consente di estrarre i bit trasmessi malgrado il canale possa sembrare un'unica grande interferenza.

8.3.5 Nello spazio

Possiamo usare la stessa frequenza f , basta che → dove la utilizziamo la prima volta sia abbastanza (spazialmente) separato da dove la riutilizziamo. Se lo spazio che separa le colorazioni è abbastanza elevato, posso utilizzare più volte lo stesso colore. Questo è il problema del frequency planning. Qualsiasi sia il modo in cui creo i canali di diverso colore, li possiamo usare identici a condizione che ci sia in mezzo dello spazio. Lo spazio che separa è proprio lo space multiplexing (riusare gli stessi colori/canali nello spazio più volte).



8.4 FUNZIONAMENTO DI MODULAZIONE E DEMODULAZIONE

8.4.1 Modulazione

Chi vuole parlare/trasmettere, deve scegliere un canale comune a tutti. In qualche modo ci si sta coordinando per capire esattamente cosa si sta trasmettendo e filtrando ciò che si sta trasmettendo rispetto a chi sta trasmettendo su altri canali. Quindi ogni canale diventa un dominio di trasmissione a sé. Siamo su un canale, vogliamo usare le onde radio per trasmettere dei bit, abbiamo bisogno di modulare l'onda radio per trasmettere dei bit, abbiamo bisogno di modulare l'onda radio, fornire all'antenna quest'onda e a questo punto il segnale viaggia e viene riconvertito in bit, dopo essere stato catturato dall'antenna del ricevitore.

Modulazione digitale: trasmettere usando le onde radio (onda analogica, varia continuamente nel tempo) e codificare dei valori digitali (0 o 1). Questo si fa tramite ASK, FSK o PSK. Una sequenza di bit viene data in input, a livello fisico, ad un modulatore digitale, che prende il valore dei bit e cerca di rappresentarlo usando la sinusoide. Generata la sinusoide essa varierà rappresentando i valori della sequenza che vogliamo trasmettere. Dobbiamo prendere la sinusoide e copiarne le caratteristiche su un'altra sinusoide (che rappresenta il canale). In pratica il modulatore analogico prende un segnale analogico in ingresso e ne copia le caratteristiche su quello che è definito essere il nostro canale radio (il colore definito precedentemente, diviso in time, frequency, time + frequency o codice), che viene deciso sotto (nello schema dove c'è scritto "Radio Carrier (channel)", una volta stabilita però la frequenza usata per trasmettere i bit (che viene per l'appunto presa in input da sotto nel modulatore analogico). Il modulatore analogico infatti genera un segnale a quella frequenza che copia le caratteristiche della sinusoide che gli è arrivata in ingresso (sa sinistra, cioè dal modulatore digitale).

In output il modulatore analogico genera una frequenza del canale che incorpora le caratteristiche in input (che a sua volta, incorpora le caratteristiche dei bit). Ed è proprio questo che viene mandato all'antenna (traduce un'onda radio che ha le stesse caratteristiche). Se nel canale ho il canale "blu", la frequenza è quella di "colore blu", che però varia in modo analogico ma tutte le modifiche del segnale in realtà rappresentano i valori dei bit che vogliamo trasmettere. Tutto ciò viaggia nell'ambiente ed arriva al ricevitore. Lo schema di direzione è speculare a quello di ricezione è speculare a quello di trasmissione. Abbiamo un'antenna che cattura l'effetto radio (così come arriva, magari alterato) dopodiché la corrente elettrica indotta nell'antenna viene passata come input al demodulatore analogico che prende il canale radio ed estrae l'onda radio filtrando solo sulla frequenza richiesta (cioè quella arrivata da sotto). Il demodulatore controlla quale energia dell'onda radio estrarre proprio sulla base del radio carrier (la definizione comune di canale tra chi trasmette e chi riceve).

Una volta che il demodulatore sa quale frequenza ascoltare in quale istante, il suo output è "equivalente" alla sinusoide in input al modulatore analogico nel mittente, ("equivalente" perché ha subito tutte le modifiche date dall'ambiente e quindi non saranno mai uguali).

Quello che riceviamo è quindi poco simile a ciò che è partito, ma la sinusoide che è partita è in un certo senso contenuta in quella che siamo riusciti ad estrapolare dal canale. Più precisamente, la sinusoide che è partita è proprio la spina dorsale della nuvola che ci è arrivata.

Quindi anche se il segnale è disturbato, è fortemente correlato al segnale che è partito. L'ultimo blocco, quello dell'interpretazione, l'inteprete deve capire se la "nuvola" assomiglia a bit 1 o bit 0. A questo punto copia bit per bit e cerca di dare la corretta interpretazione. Se ci sono dei bit sbagliati bisogna stare attenti a riuscire a trovare una soluzione.

