

# Reti II

## Index

- Supporto della rete al traffico multimediale
  - Sfruttare al meglio il best effort
  - Approccio diff-serve
    - Classificazione
    - Scheduling
      - Isolamento
    - Policing
    - Meccanismi applicati al policing
  - Per-connection QOS
- Principi architetturali di Internet
  - Sopravvivenza
  - Supporto a molteplici servizi
  - Obiettivi secondari
  - Il futuro di internet
- Attacchi alle reti
  - Classi di attacchi ad una rete
  - Sniffing
    - Sniffing tools
    - Sniffing on LAN
      - Arp poisoning
    - Detecting sniffers
  - Spoofing on IP
  - IP fragmentation and Ping of death
  - Fragmentation to avoid firewall
  - ICMP Attacks
    - Ping scan
    - Smurf attack
    - ICMP Redirect
    - ICMP Destination unreachable
  - Attacks over UDP
    - UDP Port scanning
    - The fraggle attack
  - Attacks over TCP
    - TCP Port scanning
    - IDLE Scanning
    - OS Fingerprinting
    - TCP Spoofing
    - TCP Hijacking
    - Syn flooding attack
  - State Attack
  - Attacks over HTTP
  - Attacks over Wireless
    - Sniffing over Wi-Fi
    - DDOS over Wi-Fi

- Man in the middle over Wi-Fi
  - WEP & WPA
- Protocol Security Analysis
  - FTP Vulnerabilities
  - FPT Bounce attack
  - FTP Bounce scan
  - DNS Vulnerabilities
    - DNS Spoofing
    - DNS Hijacking
    - DNS Cache poisoning
    - The Kaminsky attack
    - DNSSEC
    - DNS Amplification
    - DNS Tunneling
- Segnalazione
  - SS7
    - Private branch exchange
    - Come viene trasmessa la voce
    - Rete telefonica intelligente
    - Data plan & control plan
  - ATM
    - Virtual circuits
    - Comutatori ATM
    - Segnalazione su ATM
      - Errori su hard state
  - Rete cellulare
    - Primo hop
    - 2G – GSM
    - 3G – UMTS
    - Gestire la mobilità in una rete cellulare
    - Confronto 3G e 4G
    - QOS
  - Internet
    - Inter-domain routing
    - BGP
      - BGP insegna
    - Soft State
    - Hard State
    - Multicast su IP
      - IGMP
      - Routing Multicast
    - Spettro dei meccanismi di segnalazione
- Software defined Network – SDN
  - Openflow
  - Switch in Openflow
  - Astrazione del piano dati
  - Azione sui pacchetti
  - Controller
  - Controllo di accesso dinamico
  - Mobilità trasparente

- Bilanciamento del carico
  - Messaggi controller-switch
  - Messaggi switch-controller
  - SDN per datacenter
  - Conseguenza per gli standard
  - Sviluppi del networking con SDN
- Randomizzazione
  - CSMA/CD
  - Randomizzazione sui Router
  - Randomizzazione nelle code
    - Scarto casuale in anticipo
    - RED
  - Randomizzazione in BitTorrent
    - Peer torrent
    - Funzionamento
    - Bitmap file
    - Tit-for-tat mechanism
    - Rarest-first mechanism
    - Problema dell'ultimo pezzo
    - Dettagli
  - Randomizzazione e load balancing
    - Matrici di traffico
    - Valiant load-balancing
- Indirezione
  - Indirezione nel multicast
  - Mobilità e indirezione
    - Registrazione
    - Mobilità con routing indiretto
      - Vantaggi e svantaggi
      - Problema dell'egress filtering
    - Mobilità con routing diretto
    - IP Mobile - i3
  - Infrastruttura di internet con indirezione
    - Architettura proposta
    - Discussione
    - Gestione della mobilità
    - Gestione del multicast
    - Anycast
    - Servizi componibili
  - Distributed Hash Table – DHT
    - Obiettivi di progetto
    - Chord
- Modelli di TCP
  - Ripasso
  -

# Multimedia Networking

Al giorno d'oggi il traffico Internet è dominato dalle applicazioni multimediali, principalmente il video. E in termini di volume di traffico, quindi quantità di byte che girano in rete, il video è dominante rispetto a tutte le altre applicazioni. Si stanno diffondendo sempre di più App che permettono di vedere video anche in movimento dagli smartphone. Non sono soltanto movie, ma sono anche brevi video generati dagli utenti stessi caricati su varie piattaforme Youtube, Tik tok, etc

Esistono almeno tre le scenari di Multimedia Networking che sono completamente diversi in quanto richiedono requisiti di rete molto diversi tra loro:

- **Streaming stored audio/video:** la comunicazione è unidirezionale, avviene in streaming da una sorgente a tanti utenti in broadcast, quindi non è full duplex, non è una conversazione, è solo un punto che vuole trasmettere a uno o più ricevitori un contenuto che è già stato generato ed è già stato memorizzato in uno o più server. Gli utenti quindi chiedono la risorsa in modo on demand, quindi in modo asincrono tra di loro. Applicazioni di questo tipo: Netflix, Youtube, etc
- **Streaming Live:** ci sono anche delle applicazioni che trasmettono su Internet dei contenuti video Live: Twitch, youtube Live, etc. Questo scenario si differenzia dal primo: il contenuto è generato sul momento e la trasmissione è live (anche se i ricevitori tipicamente non lo vedono istantaneamente), si parla di un ritardo di qualche secondo (al più 10 secondi), ovviamente non eccessivo altrimenti non sarebbe più un evento live.
- **VOIP:** chiamate e videochiamate su IP dove c'è bisogno del Real Time e full duplex, quindi, il traffico viene scambiato nei due sensi e ha bisogno di requisiti di ritardo molto stringenti.

## Streaming-Stored audio/video

Nello streaming stored audio/video, c'è il processo di generazione del video (tipicamente offline) che avviene molto prima di quando il video viene servito effettivamente sulla rete. Il contenuto viene memorizzato poi su dei server (di solito su delle CDN Content Delivery Network), replicato con varie copie su tanti server, in modo da avvicinare il contenuto agli utenti. Dopodiché questi server trasmettono i contenuti agli utenti on demand.

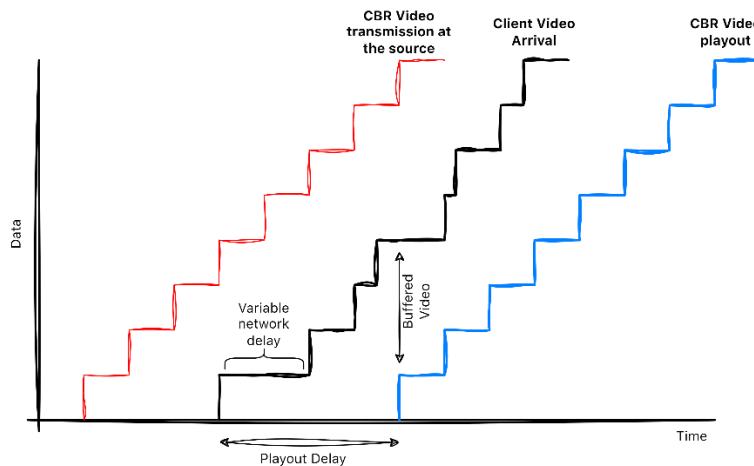
Chiamiamo "streaming" questo genere di comunicazione perché il ricevitore non scarica tutto il file e successivamente lo riproduce, ma comincia a riprodurre il video mentre sta scaricando i vari pezzi successivi del contenuto.

Gli utenti ovviamente sono molto infastiditi quando il video si interrompe, quindi quando c'è qualche problema di rete che fa sì che si blocchi la riproduzione, possono succedere due cose: o il problema si risolve o si riparte dal Frame a cui era arrivato oppure, nei casi più gravi, bisogna fare uno skip in avanti, quindi un intero pezzo di video va perso e si salta ad una ad un istante successivo, che è ancora peggio, perché si perde proprio un segmento di video. In entrambi i casi, gli utenti sono terribilmente infastiditi da questa cosa, quindi nelle applicazioni si cerca di fare in modo che quando parte la riproduzione poi non si ferma più, cioè continua pulita fino alla fine. Il problema è che su Internet la cosa è complicata in quanto la banda a disposizione varia nel tempo per congestione della rete, qualità del canale (magari anche wireless) e variano anche i ritardi, quindi il ritardo che subiscono i pacchetti dal server all'utente è molto variabile.

La variazione del ritardo con cui pacchetti raggiungono la destinazione in termini tecnici, si chiama jitter. Jitter quindi si intende la variabilità del ritardo dei pacchetti.

Come si compensa la variabilità del ritardo? Con un buffer nel client che ha proprio lo scopo di provare ad assorbire questa variabilità. Altre sfide che devono risolvere le applicazioni di video on demand sono che gli utenti spesso non si accontentano di vedere un video dall'inizio alla fine, ma saltano in avanti, lo fermano, tornano indietro, si stufano e passano ad uno successivo

infine una altro problema, inerente al fatto che siamo su Internet, è che i pacchetti possono venire persi per vari motivi e in certi casi ha senso ritrasmetterli, in altri casi è inutile ritrasmetterli se sono persi, se li ritrasmettessi arriverebbero troppo in ritardo. Le applicazioni audio e video tollerano un po' di pacchetti persi, non è una disgrazia perdere ogni tanto dei pacchetti, basta che globalmente il contenuto sia comprensibile (a differenza di altri tipi di applicazioni dove non si tollera nessuna perdita in cui i dati devono arrivare integri tutti, senza nessuna perdita).



In rosso c'è il rate con cui gli oggetti vengono trasmessi dal server (supponiamo a Constant bitrate) che genera la scalinata rossa (a scalini perché si trasmette il video a pezzi discreti).

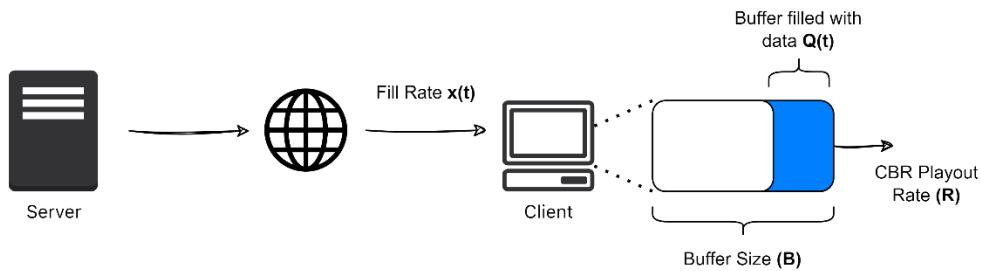
La curva nera invece è il modo in cui il ricevitore riceve i pezzi del contenuto, dove si nota che i pacchetti subiscono dei ritardi casuali, e quindi si trasforma in una scalinata dalla forma irregolare, a volte con anche delle zone piatte che significa che ci sono dei lunghi ritardi tra un pacchetto all'altro. La parte piatta dello scalino indica il tempo di delay tra un frammento del contenuto all'altro.

A fronte di questa curva nera bisogna scegliere un opportuno istante di tempo nello spazio in cui si decide di far cominciare la riproduzione al ricevitore (questo delay, che parte dal momento in cui arriva un chunk di pacchetti a quando si riproducono effettivamente prende il nome di **client playout delay**). Questo per garantire una riproduzione fluida a bitrate costante, in modo regolare e senza pause per ottenere quindi, in fase di decodifica, di nuovo una scalinata regolare (quella blu).

Il ritardo di Playout è un parametro dell'applicazione che serve ad assorbire i ritardi casuali della linea nera. Quindi sarà un parametro settato in modo da assorbire la variabilità della scalinata nera (in particolare una cosa assolutamente evitare è far incrociare la scalinata blu con quella nera). Se la linea nera si incrocia con la blu vuol dire che mancano dei pacchetti e il video al ricevitore si interrompe, perché la differenza in verticale tra la linea nera e quella blu è l'occupazione istantanea del buffer. Quindi il ricevitore ha un buffer di ricezione dove memorizza i pezzi di contenuto che gli arrivano dalla rete. Quando la differenza tra la curva nera e quella blu si azzerà, il buffer si è svuotato ed è gravissimo, in quanto il decodificatore non ha più i dati a disposizione per riprodurre il video.

Il Client playout viene dimensionato ed evidentemente bisogna scegliere un compromesso: il problema di farlo troppo piccolo o grande. Se si sceglie un playout piccolo si va incontro a un maggior rischio che il video si fermi. Se lo si fa invece troppo grande c'è rischio che l'utente si stufi

dell'attesa e cambi contenuto (magari l'utente pensa che l'applicazione non funziona). Di solito un utente è disposto ad aspettare qualche secondo.



Questo è un altro modo di vedere il diagramma temporale con le scalette: si vede il buffer di ricezione lato client, dove la porzione in blu è il livello di riempimento del buffer. Quindi il decodificatore prende dati da questo buffer ad un rate che è il payout rate del video (supponiamo **CBR - R**). Il buffer viene invece riempito da sinistra ad un rate  $x(t)$  variabile nel tempo per via delle oscillazioni della rete. Il livello istantaneo di quantità di dati nel buffer è  $Q(t)$ .

Quando quindi l'utente inizialmente richiede un contenuto, c'è il ritardo di riproduzione durante il quale il client comincia a riempire il suo buffer, si crea una specie di riserva di dati ed arrivato ad un certo livello di riempimento del buffer, decide di fermarsi ed incomincia la riproduzione. Da questo istante di tempo in poi il livello del buffer viene svuotato a Bitrate R e riempito a bitrate  $x(t)$  variabile nel tempo.

Quando il buffer si svuota è un male, perché vuol dire che il video si ferma e l'utente è infastidito da questo e di conseguenza è necessario dimensionare opportunamente il livello iniziale, quindi i dati accumulati inizialmente per evitare il più possibile questa interruzione.

Da notare che, il buffer in ricezione non è infinito, ha una sua dimensione massima **B**. Significa che anche se la rete è velocissima, si potrebbe decidere di non scaricare tutto il contenuto, ma di tenerne una quantità al più **B**. Questo sempre per il fatto che gli utenti sono imprevedibili e potrebbero a un certo punto interrompere il video perché non gli interessa più e non guardarlo fino alla fine, quindi bisogna evitare di sprecare banda e scaricare tutto inutilmente.

Cosa succede inoltre quando l'utente effettua un forward sul video (mandandolo avanti) al contenuto che era precedentemente salvato nel buffer? Dipende dall'applicazione: o viene perso o viene cachato in locale. Ma in generale quando l'utente fa forwarding sul player, se il salto è molto grande, più grande di B (dimensione del buffer) si riparte da capo con un nuovo network delay. Se invece salta in un punto che è già entro la dimensione B, vuol dire che nel buffer è già contenuto il pezzo su cui sta saltando, quindi è già disponibile almeno un pezzetto, però quello che succede è che si vuole sempre garantire di avere una certa quantità di riserva iniziale, quindi, prima di ripartire, quello che può succedere è che il video non comincerà esattamente a riprodurre quel punto lì, anche se magari i dati di quel punto sono già in B, perché bisogna comunque accumulare un pezzettino iniziale per i motivi già discussi.

Ovviamente se il video è molto lungo, non si può pensare di cachare 4GB di contenuto, dipende dal dispositivo e dall'applicazione. In generale non è conveniente memorizzare in locale. Su youtube ad esempio viene cachato il contenuto.

Supponiamo  $x$  il valor medio della banda che abbiamo a disposizione, quindi del rate di scaricamento  $x(t)$ , nel lungo termine quindi sono due le possibilità:

- $x < R$ : se il Valor medio della banda è più piccola di R, prima o poi il buffer si svuota e quindi il video si ferma e ci sarà appunto un'un'interruzione
- $x > R$ : viceversa, il caso favorevole quando x è più grande di R. Quindi la banda a disposizione nella rete è più grande di R dove va tutto bene, il buffer non si svuota mai, e si riesce ad assorbire bene le eventuali variazioni di x(t).

## Protocolli di trasporto per streaming multimedia

Non c'è una vasta scelta, o TCP o UDP.

Usare UDP è un'opzione sicuramente da considerare, UDP è stato proprio originariamente progettato avendo in mente le applicazioni multimediali a differenza di TCP che fornisce una banda imprevedibile, variabile nel tempo. Con UDP è possibile trasmettere ad un bitrate costante che è proprio quello che di solito si ha bisogno in applicazioni audio/video facendo ad esempio la scelta di trasmettere pacchetti al rate con cui ricevitore si aspetta di ricevere il contenuto. La cosa è che non è praticamente utilizzato UDP per questi tipi di contenuti.

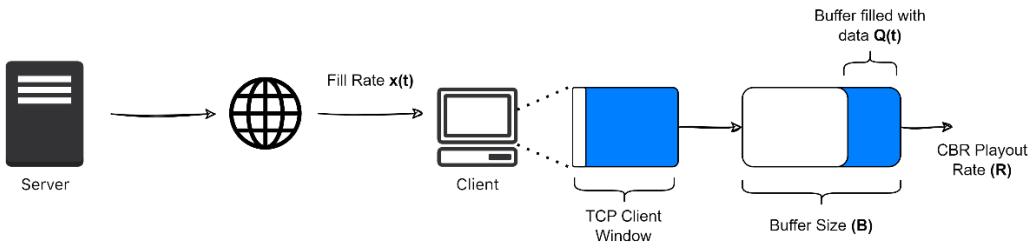
UDP quindi non è una buona scelta (nella maggior parte delle applicazioni, Netflix, Youtube, che ci sono oggi non usano UDP), ma perché?

La cosa è un po' paradossale dato che UDP è proprio stato concepito storicamente per il multimedia, però, ha dei problemi. I problemi sono i seguenti:

- si può benissimo usare UDP e sparare di trasmettere i pacchetti ad un CBR, senza tutta la complessità di TCP, del controllo di congestione, ma il problema è che se il CBR utilizzato non è sopportabile dalla rete, si andranno a perdere un sacco di pacchetti. Quindi se questo rate eccede la banda disponibile, si potrebbe arrivare ad un 10 – 20% di P/L (Packet Loss). Ma questo è solo uno dei tanti problemi di UDP per applicazioni multimediali.
- Un problema piuttosto cruciale è che il traffico UDP, spesso e volentieri, viene bloccato dai firewall, molti firewall cestinano il traffico UDP, mentre TCP non lo bloccano mai, nessuno blocca TCP.
- L'altro problema di UDP è che dato che non c'è tutta l'intelligenza di TCP, tutto l'onere del controllo in un'applicazione che utilizza UDP, lo sostiene il server perché, è il server che decide quando, come e dove trasmettere i pacchetti, se recuperare le perdite, se modificare il bitrate, etc.. Questo rende le applicazioni che usano UDP poco scalabili a popolazioni enormi di utenti.

Da notare che oltre ad UDP è anche poi necessario un protocollo a livello applicativo per gestire tutta l'intelligenza dell'applicazione. Uno di questi protocolli è RTP (Real Time Protocol).

La scelta invece che hanno adottato tutte le principali piattaforme, da Youtube a Netflix, è quello di usare TCP, ma non solo. È stato scelto di fare lo streaming multimediale basato su HTTP. Con questa modalità qua è il ricevitore, tramite get HTTP, sceglie quali pezzi del video scaricare dal server e lo può fare in modo molto flessibile grazie a tutta la flessibilità che ha il protocollo HTTP, che ad esempio, permette di specificare il byte Range, cioè all'interno di un grosso file, quello che è il video, selezionando attraverso un header (range) la porzione del file interessata.



Una cosa interessante che succede in uno scenario multimediale su TCP, se si vuole evitare di caricare troppo velocemente il file, è che quando il buffer di ricezione si satura, il buffer dell'applicazione smette di tirar fuori i dati dal buffer di ricezione TCP che quindi a sua volta si riempie, tramite un meccanismo che si chiama di back pressure di pressione all'indietro. Quindi prima si riempie il Buffer dell'applicazione al ricevitore, poi si riempie il buffer di ricezione di TCP e grazie al meccanismo di controllo di flusso di TCP automaticamente si arresta anche il trasmettitore.

Quindi usare TCP si ottengono grossi vantaggi: spostare il controllo al ricevitore e rendere i server stupidi, i server sentono solo delle HTPP get e mandano i chunk che vengono richiesti. Si sposta la parte di controllo ricevitore. L'applicazione diventa di conseguenza scalabile. E cosa non da poco il fatto che HTTP, che gira su TCP, non viene mai bloccato da nessun firewall.

## DASH

Questa soluzione di HTTP è stata raffinata ulteriormente con un meccanismo che si chiama **DASH** (**D**ynamic **A**daptive **S**treaming over **H**TTP). È dicono lo stato dell'arte, la soluzione più sofisticata che c'è attualmente per il video on demand. È di fatto la soluzione che ha adottato Netflix.

Qual è la miglioria introdotta Dash rispetto a usare puramente HTTP su TCP? È che permette di cambiare dinamicamente nel tempo il bitrate. Quindi offre i video a bitrates diversi con qualità più o meno bassa (perché i vari ricevitori sono molto eterogenei, ci sono alcuni ricevitori magari che sono dei dispositivi mobili che si accontentano di un certo bitrate, magari più basso, altri che invece si possono permettere bitrates più alto).

Oltre al fatto di poter servire il video a bitrates diversi, con DASH è possibile dinamicamente adattare il bitrate alle condizioni istantanee della rete, quindi cambiare la qualità del video nel tempo, a seconda di che cosa la rete offre. Per permettere questa miglioria il video viene inizialmente memorizzato sui server a varie qualità diverse. Successivamente quando il client richiede un certo contenuto, si inoltra il cosiddetto manifest file, cioè un file particolare dove si elencano tutti i bitrates disponibili per quel contenuto con i rispettivi URL. Sulla base di questa informazione, il client fa delle misure della banda disponibile in tempo reale, quindi mentre riceve i pezzi del contenuto misura il bitrate che ha a disposizione nella rete così può dinamicamente, nel tempo, decidere di saltare ad una codifica più elevata o più bassa consultando il manifest file.

Comunque tutto è eseguito lato client, è qui che risiede la parte di controllo ed intelligenza del flusso. Il client in particolare determina quando, cosa e dove:

- **When:** Quando, tramite HTTP GET con l'header byte, richiedere i chunk, e lo fa quando ne ha bisogno, quindi a seconda del livello corrente di occupazione del buffer.
- **What** (encoding rate): Decide di prelevare dal server dei pezzi del contenuto andando a scegliere un'opportuna codifica tra quelle disponibile nel manifest file e a seconda di misure in Real Time della banda disponibile
- **Where:** il client può ottimizzare anche la scelta di quale server andare ad utilizzare per scaricare i chunk

Ora si suppone di essere un content provider e di avere un video da distribuire agli utenti sparsi nel mondo. Una prima soluzione (naive) non funzionante è quella di noleggiare un server in rete potentissimo su cui si carica il video in modo che poi questo server lo distribuisca a tutto il mondo. Il problema di questo è, prima di tutto, quello di avere un Single point of failure, congestione su un singolo punto della rete. Ma la cosa più importante è che sarebbe impossibile gestire, è la latenza e le perdite a livello geografico: gli utenti sono sparsi geograficamente, chi si trova distante dal server centrale trova difficoltà a riprodurre il contenuto.

Per risolvere questo problema, la soluzione standard al momento è utilizzare le Content delivery networks (CDN).

# CDN

Le content delivery network (CDN) sono delle infrastrutture (dei server) specializzati nella distribuzione di contenuti.

Le CDN fanno proprio il mestiere di disseminare copie del contenuto geograficamente su dei server sparsi nel mondo, avvicinano quindi il contenuto agli utenti, ottenendo un sistema distribuito che quindi non ha più i problemi della soluzione centralizzata, ridicendo il carico in rete e la latenza che subiscono i pacchetti perché idealmente un utente che vuole il contenuto, lo andrà a prelevare dal server CDN a lui più vicino, quello che sperabilmente offre la latenza minore. Questo andrà ovviamente a vantaggio della qualità percepita.

I CDN pubblici offerti come servizio per la distribuzione di contenuti possono essere suddivisi in due sottogruppi: CDN con architettura Enter-Deep e CDN con architettura Bring-Home.

## Enter-Deep Architecture

L'architettura Enter-Deep si basa sulla distribuzione dei CDN il più vicino possibile ai client. I server sono generalmente posizionati nelle reti di accesso degli ISP, consentendo un tempo di andata e ritorno molto ridotto per i client.

I vantaggi con latenza e prestazioni inferiori sono generalmente controbilanciati quando si mantiene un numero elevato di server. La configurazione e la propagazione delle modifiche possono essere impegnative.

## Bring-home Architecture

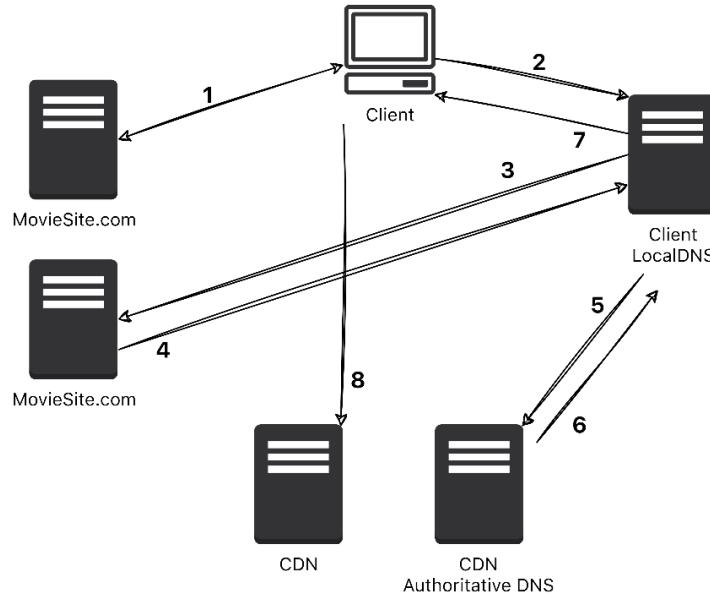
L'architettura enter-deep mira a posizionare in modo intelligente un numero minore di server CDN, ma molto più grandi, in posizioni strategiche come punti di scambio o punti di peering di grandi dimensioni (dove si attaccano gli ISP). Questi server CDN sono ad alte prestazioni con collegamenti a larghezza di banda elevata a più ISP e possono quindi ottenere buone prestazioni nei confronti dei clienti.

Con pochi server, la manutenzione e la propagazione della configurazione sono piuttosto efficaci. Tuttavia, un minor numero di server richiede posizioni attentamente pianificate per prestazioni di connessione ottimali e accordi di peering che raggiungano in modo efficiente qualsiasi client nella regione. E se uno dei pochi server fallisce, i client verranno indirizzati ad altri server solitamente molto più lontani da quello guasto, generando potenzialmente una maggiore latenza.

## Funzionamento

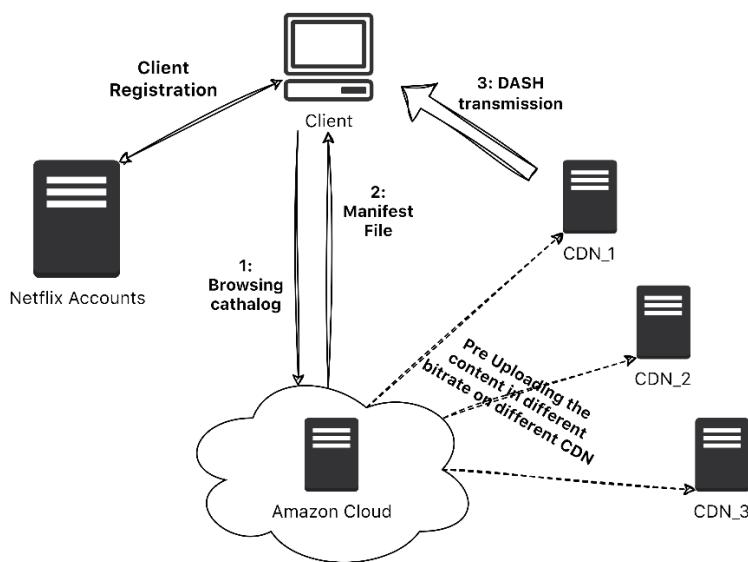
Come si fa ora ad indirizzare un utente verso un particolare server di una CDN? Per indirizzare un utente ad un contenuto situato in un certo server CDN, si utilizza il DNS.

Un client inizia a navigare il sito di contenuti multimediali (1). Successivamente sceglie il contenuto da riprodurre, e selezionandolo risolve la richiesta utilizzando il local DNS (2). Il local DNS inoltra la query all'autoritative DNS del content provider (3) che a sua volta indica al local DNS di chiedere direttamente al DNS server della CDN (4). L'autoritative DNS della CDN risponde con un server opportuno (6) al local DNS che a sua volta inoltra la risposta al client (7) che andrà a richiedere al server CDN il contenuto (8).



Quindi, tramite DNS, il content provider demanda il problema di trovare un un'opportuno indirizzo IP di un server, alla CDN che al suo interno conosce tutti i suoi server, quindi utilizzerà una qualche strategia per trovare il server opportuno vicino all'utente: per trovare il server opportuno, il CDN effettua intanto delle misure di ritardo, quindi, facendo dei ping in modo regolare, può misurare la latenza attuale tra i suoi server e chi sta chiedendo un contenuto. Più banalmente invece, potrebbe, sulla base della geolocalizzazione degli indirizzi IP, selezionare un server che è ha un noto punto geografico vicino all'utente. La seconda opzione potrebbe essere un'arma a doppio taglio: la distanza geografica non è necessariamente correlata ad avere un buon link con una buona banda. Se l'utente sta usando il suo DNS per gestire la risoluzione della sua richiesta, non ci sono problemi. Ma molte volte gli utenti utilizzano DNS pubblici (tipo quelli di google 8.8.8.8) che potenzialmente sono lontani da loro geograficamente.

Per risolvere questo problema bisogna necessariamente spostare il problema del trovare il server opportuno nei client (come fa Netflix) attraverso il manifest file.



Una soluzione alternativa a quello DNS, senza manifest, è quello di usare Anycast IP attraverso BGP.

# VOIP

L'enorme differenza rispetto al caso precedente di streaming video on demand, è che in questo scenario si stabiliscono dei flussi multimediali interattivi.

Ormai è inevitabilmente la transizione da rete telefonica classica a telefonate su IP, la rete telefonica tradizionale si sta sempre più smantellando. Effettuare le chiamate su IP però, non è un'impresa banale. IP infatti, offre un servizio best effort quindi non è certamente stato progettato e sviluppato per fare telefonate. Nel corso degli anni però, sono state sviluppate ed adottate delle tecniche a livello applicativo per permettere le chiamate su IP, senza ovviamente toccare il livello IP (immutabile a quanto pare), per avere telefonate di qualità accettabile su Internet.

## Requisiti del VOIP

Prima di tutto c'è un requisito di **latenza**. In particolare in una trasmissione VOIP:

- Quando la latenza è minore di 150ms, la chiamata è sostenibile, a livello percettivo non ci accorge di niente
- Tra i 150ms e i 400ms inizia a sentirsi a livello percettivo la latenza
- Sopra i 400ms la telefonata subisce un netto degrado

Sopra i 400ms, a livello percettivo, si ha come l'impressione che il ricevitore non abbia sentito o non sia in ascolto.

Altri problemi che sorgono se si vuole davvero trasmettere telefonate su Internet sono quelli legati allo stabilire una **sessione**. Intanto trovare il ricevitore, cioè la persona con cui si vuole parlare: come si trova un ricevitore? come si ottiene il suo indirizzo IP con cui aprire un flusso audio? Come si gestisce il ricevitore in movimento?

L'altra cosa che bisogna fare in Voice over IP è idealmente offrire anche tutti i **servizi** che sono disponibili nella rete telefonica tradizionale, cosiddetti servizi di rete intelligente: l'inoltro di chiamata, chiamata su occupato, il blocco, i numeri verdi, segreteria, etc...

Per via dei requisiti stringenti di latenza che ha il VOIP (max 400ms) fanno sì che questa volta, la scelta migliore a livello di trasporto, vada ad UDP. Questo perché bisogna davvero mandare un CBR in questo caso: si sa il rate a cui mandare il traffico dati di una telefonata (64Kb/s). In questo caso quindi è sciocco usare una banda variabile VBR con TCP per poi ritrasmettere anche inutilmente i pacchetti, quindi tutti i meccanismi che offre TCP di recupero dei pacchetti persi non servono alla fine.

I meccanismi specifici per fare una buona qualità del servizio vengono gestiti comunque a livello applicativo.

## Caratteristiche VOIP

Quello che succede tipicamente in un'applicazione VOIP è che il codificatore audio genera dati ad un certo bitrate (codifica standard della telefonia: PCM 64 Kbps, 8000 KB/s) ed ogni 20 millisecondi si raccoglie tutto quello viene prodotto dal codificatore e lo si pacchettizza: utilizzando la PCM 64Kbps, ogni 20 secondi l'encoder genera 160 byte di dati che vengono messi in un pacchetto UDP e poi consegnato ad IP sperando che arrivi dall'altra parte della rete (best-effort).

Per come funziona una chiamata vocale, è possibile anche fare la soppressione dei silenzi, cioè trasmettere effettivamente pacchetti dati solo quando un utente parla e durante i silenzi non

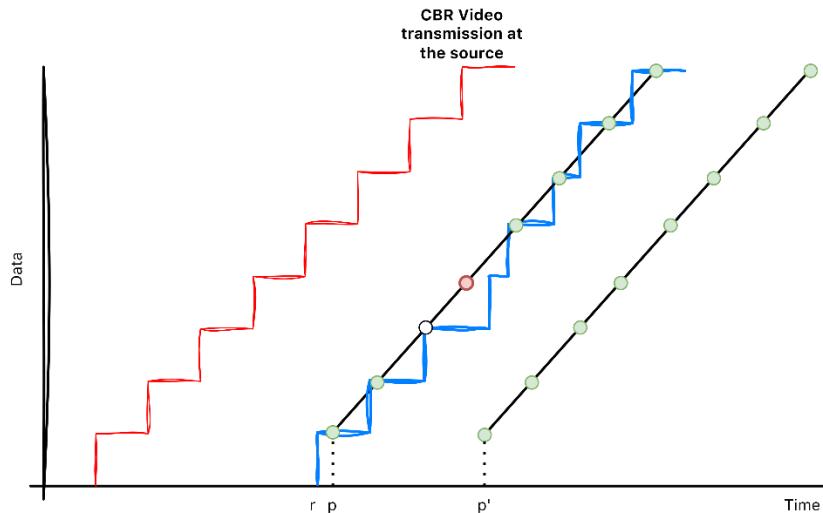
trasmettere nulla. Quindi si avranno dei periodi di attività della voce (talk spurts) e dei periodi di inattività, in quando sarebbe inutile trasmettere dei pacchetti che corrispondono a un periodo di silenzio.

### Risolvere il delay e packet loss

I pacchetti di talk spurts andranno incontro a un sacco di possibili guai, in particolare si parla di possibili perdite dovute a svariati motivi, ritardi di accodamento tipicamente nei Router e altri motivi che fanno sì che la latenza sia impredicibile e casuale. Questo è un problema perché appunto se la latenza totale supera i 400ms, il pacchetto è praticamente da buttare, anche se arriva al ricevitore, dopo 400ms a livello percettivo è da buttare.

D'altra parte avere dei pacchetti persi su trasmissioni VOIP, non è un problema grave. In particolare, si è visto che ad esempio con i codificatori attuali, avere una probabilità di perdita di pacchetti tra l'1 ed il 10% può essere tollerato. Questo significa che in questo range di valori 1-10% (che è abbastanza alto) tutto sommato non si perde tanto in qualità percepita dall'utente.

Quello che invece si dovrà fare, è attuare la stessa strategia che viene attuata nel VOD (Video-On-Demand), per assorbire il jitter delay. La differenza è che quid il playout-delay non può essere della stessa grandezza. Questa volta, nel VOIP, si ha un budget molto più contenuto: 150ms.



Se il playout delay è basso ( $p$ ) è probabile che alcuni pacchetti non vengano calcolati dall'encoder e quindi persi. Se il playout delay è più alto ( $p'$ ) non avremo perdite ma a discapito di un ritardo artificiale al ricevitore.

Questo ritardo che si introduce artificialmente al ricevitore, è possibile farlo fisso, quindi un unico valore per tutta la telefonata, da quando inizia a quando finisce, oppure farlo adattativo nel senso che cambia nel tempo e si adatta al talk spurt, cioè ogni periodo di attività dell'oratore, si usa un playout delay diverso che si può adattare dinamicamente alle condizioni della rete stimando la latenza dei pacchetti.

Per stimare la latenza si utilizza una EWMA (exponentially weighted moving average), la stessa che si usa per l'RTT in TCP:

$$d_i = (1 - \alpha)d_{i-1} + \alpha(r_i - t_i)$$

$(r_i - t_i)$  è il sample attuale. Se Alpha è uguale ad 1 in sostanza si prende in considerazione solo il sample attuale, se alpha è uguale a 0 è l'opposto, si prende in considerazione solo il sample (delay) precedente. Un valore ben dimensionato di Alpha può dare dei risultati sulla stima della latenza media molto efficiente.

Se unito alla deviazione standard:

$$v_i = (1 - \beta)v_{i-1} + \beta|r_i - t_i - d_i|$$

Si può calcolare il playout delay adattivo:

$$\text{playout}_i = t_i + d_i + Kv_i$$

Uno dei problemi che potrebbe capitare è il caso in cui un trasmettitore faccia anche la soppressione dei silenzi e quindi non trasmette i periodi di inattività. In questo caso il ricevitore non conosce questi istanti in cui la voce è stata soppressa, perché la soppressione dei silenzi la fa il trasmettitore, però non è che la comunica al ricevitore. Il ricevitore riceve una serie di campioni audio e poi deve cercare di riprodurre la voce lato ricevitore. Quello che succede è che il ricevitore può anche ricevere un buco, quindi un flusso interrotto non a causa di un silenzio che è stato soppresso al ricevitore, ma un problema della rete dovuto da perdite di pacchetti e quindi deve avere un meccanismo per distinguere un caso dall'altro.

La soluzione che è stata adottata comunemente nel traffico VOIP è quella di usare congiuntamente dei timestamp e dei numeri di sequenza, quindi ogni pacchetto audio generato al mittente è sia marcato temporalmente con un timestamp che con un numero di sequenza. Il timestamp è un tempo assoluto di trascorso dall'inizio della chiamata.

In questo modo si da un'idea di sequenza che cresce per ogni pacchetto generato "davvero" e quindi quando si ha un periodo di silenzio. Durante un periodo di silenzio, il timestamp continua a crescere, ma il numero di sequenza dei pacchetti resta fermo.

## Recovery from packet loss

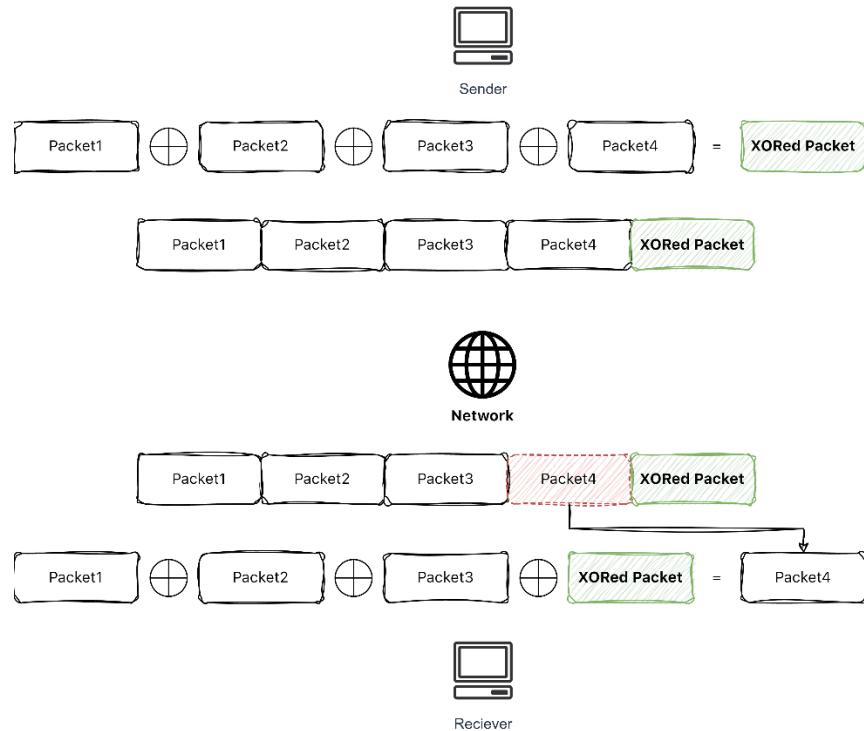
Con i flussi audio si possono benissimo tollerare perdite dall'1 al 10%. Questa cosa è resa possibile anche grazie all'uso di opportune tecniche di mitigazione dei pacchetti persi - P/L.

### Forward error correction

Uno di questi metodi è quello di usare la correzione degli errori in avanti (Forward Error Correction - FEC). In questo metodo, invece di ritrasmettere ciò che è andato perso si cerca di recuperare i pacchetti persi al ricevitore, cioè il ricevitore grazie a dei meccanismi riesce a recuperare o ricostruire i pacchetti persi.

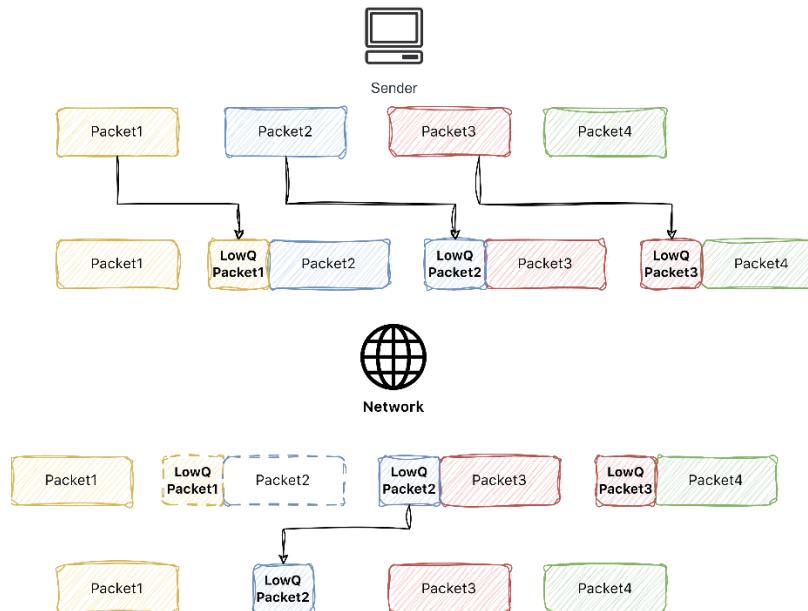
Quello che si fa è prendere n pacchetti ed effettuare l'operazione di XOR tra questi n pacchetti, generando così un n+1 pacchetto nuovo che contiene lo xor degli n pacchetti precedenti e trasmettendo pure quest'ultimo.

Si sta trasmettendo un po di ridondanza, però, se per disgrazia la rete perde uno qualunque degli n pacchetti, è possibile ricostruirlo al ricevitore grazie al fatto che l'n+1 pacchetto è lo XOR di tutti i precedenti pacchetti.



Lo svantaggio di FEC è che bisogna aspettare  $n$  pacchetti però comunque grazie al playout-delay non si nota molto. Lo svantaggio principale però è che, se si perde più di un pacchetto, non si può ricostruire più niente da quel  $n+1$  esimo pacchetto.

Si possono usare anche tecniche più sofisticate, ad esempio c'è un altro schema FEC che funziona in questo modo:



Per ogni pacchetto si genera una codifica di buona qualità e una codifica di bassa qualità. Quella di bassa qualità, che è molto piccola quindi occupa molto meno byte, la si appende al pacchetto successivo. Di conseguenza, se la rete perdesse l' $N$  pacchetto, si avrebbe la sua versione a bassa

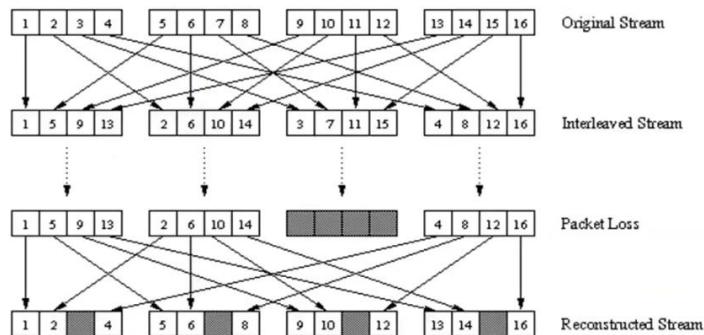
qualità, inserita nel pacchetto N+1. Ovviamente questo è possibile se il pacchetto N+1 è già disponibile. Inoltre se si perdono due pacchetti consecutivi si avrebbe comunque un buco.

Il problema che accomuna queste due tecniche è quello di avere pacchetti persi consecutivi.

### Interleaving

C'è una tecnica che mitiga il problema dei pacchetti persi consecutivi e si chiama interleaving.

Questo si fa al prezzo di introdurre un'ulteriore ritardo di decodifica, quindi la necessità di aspettare più pacchetti, di ricevere più pacchetti prima di iniziare la decodifica.



L'idea generale è quella di mescolare i dati dello Stream originale in modo da mitigare l'effetto che potrebbe essere introdotto dalla rete di perdere consecutivamente un intero blocco.

### Skype VOIP

Skype è una delle prime applicazioni che ha proposto il Voice Over IP (prima release del 2003). Quando uscì suscitò parecchio turbamento tra gli operatori di telefonia tradizionale, perché di punto in bianco si potevano fare telefonate gratis intercontinentali. Prima che uscisse Skype (in generale il VOIP), fare telefonate a lunga distanza era molto costoso. Grazie a questo fatto, acquisì subito moltissimi utenti. È stata poi acquisita da Microsoft nel 2011.

Skype è un protocollo proprietario, chiuso e criptato quindi non si conoscono esattamente tutti i dettagli di come funziona. Però una cosa molto interessante che hanno fatto vari ricercatori è provare a fare una specie di reverse Engineering del protocollo di Skype osservandolo in azione in rete. È un'operazione interessante che si può fare anche con altri protocolli proprietari, cioè farli girare, osservando come si comportano i pacchetti che viaggiano, in modo da estrapolare qualcosa sul protocollo.

Quello che si è capito di Skype è che ci sono dei nodi (utenti) quindi non controllati direttamente da Skype e dei supernodi. Questo crea quindi una struttura gerarchica a due livelli in cui gli utenti sono le foglie e poi, dei supernodi più grossi, più stabili, con maggiori risorse computazionali che aiutano il protocollo a funzionare.

Per diventare un super nodo bisogna avere queste proprietà:

- Avere un indirizzo IP pubblico
- Essere il più possibile online quindi essere attivi possibilmente 24/7 e comunque essere particolarmente stabili nel tempo
- Facilmente raggiungibili
- Avere una grossa banda sia in upload che in download.

Viceversa, i nodi utente si accendono e si spengono, sono instabili, si connettono e si sconnettono in continuazione.

Inoltre, i supernodi si conoscono tra di loro, quindi creano quella che si dice, una rete overlay, una rete logica, non fisica, a livello applicativo che li connette tra di loro, che forma una maglia complessa. Ogni supernodo ha associato tanti nodi utenti normali che sono direttamente connessi.

In più c'è un server di login, questa volta invece di proprietà di Skype, che serve per la fase di login al sistema. Però questo è il suo unico scopo, quindi il server di Skype che fa il login ha l'unico scopo di tener traccia di quali utenti sono online, quali sono spenti, etc...

### Problema del NAT traversal

Quali problemi si riesce a risolvere grazie ai supernodi? Il problema principale sono quei clienti che sono dietro ad un NAT, quindi sorge il problema del NAT traversal. La situazione più complessa quando si parla di NAT traversa è quella simmetrica, in cui i peer sono entrambi dietro un NAT. Quando invece la situazione è asimmetrica, cioè quando uno è nattato e l'altro no è relativamente più semplice da risolvere.

In Skype si è risolto il problema più complesso (simmetrico) in questo modo: gli utenti si collegano ad un supernodo in modo che il sistema entri a conoscenza dell'indirizzo IP dell'utente (anche in caso di mobilità questo aiuta).

Il problema, appunto principale da risolvere è quello di ottenere l'indirizzo IP del destinatario. Quindi se ad un certo punto il client vuole fare una telefonata verso un'altro client, deve contattarlo in qualche modo e lo fa appoggiandosi a questa rete overlay di supernodi. I supernodi scambiandosi informazioni tra di loro, implementando, probabilmente (perché non si sa certezza) una struttura dati che si chiama DHT Distributed Hash Table riescono a trovare l'IP del destinatario.

I due nodi client hanno delle connessioni TCP verso il loro supernodi. Essendo che i supernodi hanno indirizzi pubblici, i client aprono una connessione TCP verso il loro supernodo e la mantengono sempre aperta (una specie di connessione di controllo). In questo modo la rete overlay si scambia gli IP dei client.

Per far partire il traffico dati tra i due client questo non basta. In questo momento il traffico non è partito, la connessione è solo in fase di setup. Il traffico effettivo dati non necessariamente ha bisogno di attraversare questaOverlay di supernodi. La connessione dati può essere fatta attraverso un'ulteriore supernodo, diverso dai precedenti, un supernodo unico che fa solo da tramite tra i due interlocutori. Questo supernodo è soprannominato **Relay**. I due interlocutori aprono entrambi una connessione verso lo stesso relay che poi li mette in contatto tra di loro.

Il caso più semplice, è quando la persona che si vuole chiamare ha un indirizzo IP raggiungibile non Nattato. In questo caso si ottiene un indirizzo IP dal supernodo.

Si è visto che Skype, preferibilmente usa UDP.

Nel caso di multicall, di solito tutti i partecipanti mandano il loro audio al centro stella (un utente della chiamata stessa, chiamato host della chiamata). Questo centro stella mescola gli audio, quindi da  $n - 1$  flussi si passa ad 1 che viene trasmesso ai peer della chiamata, questo per ogni peer. In questo è molto più scalabile: non è  $n^2$  il numero di flussi richiesti, ma solo più nell'ordine di  $2n$ . Questo si può fare perché l'audio per sua natura può essere facilmente combinato.

Con il video non si può ovviamente combinare il segnale quindi questa tecnica non può essere applicata. La soluzione in questo caso è che ogni client manda una volta sola il proprio video ad un

server messo da qualche parte nel mondo e il server poi lo replica  $n - 1$  volte agli altri partecipanti, questo va meglio perché questi server hanno magari tanta banda (rispetto a far fare l'upload ai client).

# Protocolli per il Real Time

Questi protocolli sono utilizzati sopra il protocollo di trasporto, è necessario quindi un nuovo header che aggiunga una serie di informazioni aggiuntive da inserire nei pacchetti per gestire le comunicazioni Real Time, come VOIP ad esempio.

## RTP

RTP (Real Time Protocol) specifica il formato dei pacchetti che contengono audio e video in Real Time, quindi conversazionale. È uno standard aperto, definito nell'RFC 3550, ed è un protocollo che gira solo sugli end-system.

Tra le cose fondamentali che fa RTP è inserire quelle informazioni di cui abbiamo bisogno per marcare i pacchetti, quindi oltre a contenere i campioni di audio e video (i dati), contiene sicuramente delle cose fondamentali che sono: uno schema di numerazione, quindi un numero di sequenza progressivo, un timestamp, tipo di codifica, etc. Tutto questo si traduce in dei campi dell'header RTP.

RTP specifica solo un formato di pacchetti, non garantisce nessuna qualità del servizio al traffico che. È solo un protocollo che incapsula.

Le cose fondamentali che aggiunge l'header RTP sono:

- **Payload:** indica il tipo di codifica su 7 bit. Vale sia per l'audio che per il video, perché può anche succedere che audio e video viaggino separati, con ciascuno la propria codifica.
- **Sequence number:** 16 bit e viene incrementato ogni volta che viene mandato un pacchetto
- **Timestamp:** 32 bit. Quello che viene inserito nel Timestamp non è un vero e proprio tempo in secondi però, è equivalente ad avere un tempo assoluto. Quello che semplicemente si fa è prendere come riferimento l'istante iniziale di partenza del flusso e poi ottenere dal sampling rate della codifica automaticamente un intero che viene incrementato e che automaticamente segna lo scorrere del tempo
  - Ad esempio, PCM ha sampling rate 160 campioni. Allora il time stamp conterrà un numero di sequenza, un intero che viene incrementato di 160 ogni 20 millisecondi.
  - Notare che il timestamp scorre anche se non si trasmettendo pacchetti, quindi anche se si è in un periodo di inattività, il timestamp scorre comunque, mentre il sequence number no. È necessario avere entrambi, quindi sia un sequence number incrementato ogni pacchetto sia un timestamp per essere in grado al ricevitore di discriminare tra i silenzi e i buchi di pacchetti persi dalla rete.
- **Synchronization Source ID (SSRC):** identifica la fonte del flusso RTP. Ogni flusso nella sessione RTP ha un SSRC distinto (ogni partecipante ha un SSRC)

## RTCP

RTCP (RTP Control Protocol) lavora insieme ad RTP con scopi di segnalazione/controllo della chiamata. Quindi RTP trasporta dati ed accanto alle connessioni RTP si aprono separatamente delle connessioni di controllo della chiamata, dove gira invece RTCP.

Lo scopo principale di RTCP è di comunicare statistiche sul flusso, sul numero di pacchetti trasmessi, numero di pacchetti persi, stime del ritardo. Questo perché le statistiche possono essere utili per accorgersi di come sta andando il flusso RTP: se c'è abbastanza banda nella rete, se c'è bisogno di cambiare codifica, alzare la codifica, abbassarla; quindi avere delle misure dinamiche sul tasso di ricezione, di invio dei pacchetti è utile per eventualmente cambiare dinamicamente il flusso.

Un'altra cosa importante che permette RTCP è la sincronizzazione dei flussi quando ad esempio sono separati (audio e video).

## SIP

Il **Session Initiation Protocol (SIP)** è un altro protocollo di segnalazione utilizzato per avviare, mantenere e terminare sessioni di comunicazione VOIP.

Quello che permette di risolvere SIP è quindi il mantenimento di una sessione multimediale in tutte le sue fasi:

- Scoperta degli utenti
- Risolvere il problema della mobilità
- Call setup, quindi l'instaurazione della sessione
- Gestione della sessione durante tutto il suo svolgimento.

Inoltre anche tutte le cose che concettualmente sono possibili nel mondo della telefonia classica, quindi servizi aggiuntivi della **telefonia intelligente**, trasportati però su Internet.

### Elementi di rete

Sip funziona attraverso 3 elementi di rete:

- **User Agent**: Un agente utente è un endpoint di rete logico che invia o riceve messaggi SIP e gestisce le sessioni SIP
- **Server proxy**: è un server di rete che funge da entità intermediaria allo scopo di eseguire richieste per conto di altri elementi di rete. Un server proxy svolge principalmente il ruolo di instradamento delle chiamate; invia richieste SIP a un'altra entità più vicina alla destinazione
- **Registrar**: Un registrar è un endpoint SIP che fornisce un servizio di localizzazione. Accetta le richieste di REGISTER, registrando l'indirizzo e altri parametri dello user agent. Per le richieste successive, fornisce un mezzo essenziale per individuare possibili peer di comunicazione sulla rete.

Spesso e volentieri proxy e registrar sono due processi che girano sulla stessa macchina e quindi possono anche essere integrati tra di loro.

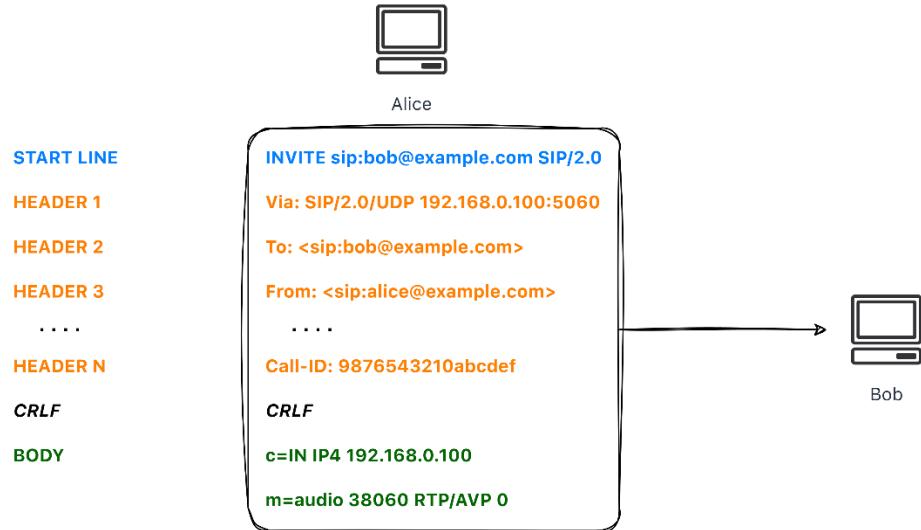
### Messaggi SIP

Il protocollo definisce il formato specifico dei messaggi scambiati e la sequenza delle comunicazioni per la collaborazione dei partecipanti. SIP è un protocollo basato su testo, che incorpora molti elementi dell'Hypertext Transfer Protocol (HTTP) e del Simple Mail Transfer Protocol (SMTP). Tra i più importanti:

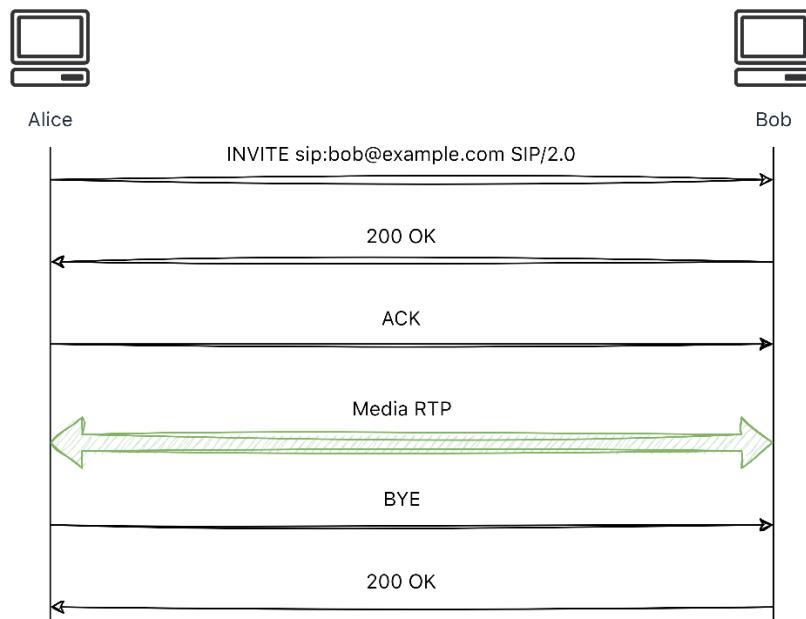
- **REGISTER**: Questo messaggio implementa il servizio di localizzazione. Memorizza l'URI indicata nel campo 'To', associandolo all'indirizzo di rete fornito in un campo 'Contact'.
- **INVITE**: Avvia un dialogo per stabilire una chiamata. La richiesta viene inviata da un client user agent ad un server proxy SIP.
- **ACK**: Conferma che un'entità ha ricevuto una risposta finale a una richiesta INVITE.
- **BYE**: Segnala la fine di un dialogo e termina una chiamata.

Un messaggio SIP (Session Initiation Protocol) è costituito da una serie di righe che includono un header e, in alcuni casi, un corpo (un po come HTTP).

Esempio di messaggio INVITE SIP:



Il caso più semplice di SIP è quando si sa già l'indirizzo IP della persona che si vuole chiamare. In questo caso non è necessario passare da nessun altro elemento di rete (Registrar e proxy), è sufficiente inoltrare il tutto al destinatario direttamente:



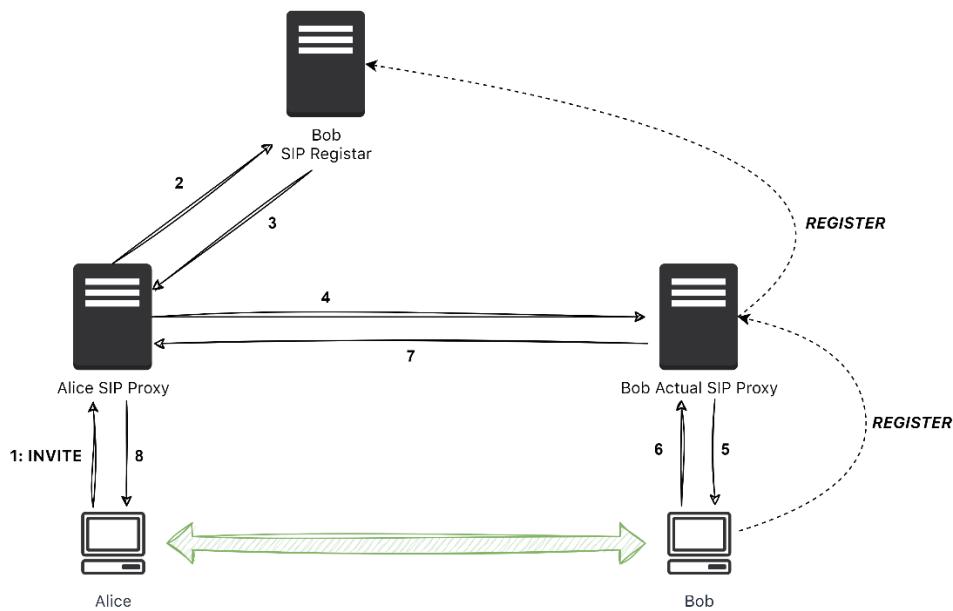
Con un semplice messaggio iniziale, gli utenti possono essenzialmente già parlare. Si cerca di tenere la cosa il più possibile semplice, si adotta un formato testuale un po simile HTTP, quindi plaintext, con comandi il più possibile nella filosofia Internet.

Questo scambio iniziale a tre vie, può essere reso un po più complicato dal fatto che magari gli utenti non supportano la codifica richiesta, quindi ci può essere una fase di negoziazione della codifica. Oppure le cose possono anche andare a non andare a buon fine perché ad esempio Bob in quel momento è occupato oppure la connessione non può essere stabilita con successo, eccetera.

Tipicamente SIP si appoggia sotto RTP però non è obbligatorio, può anche girare su qualcos'altro. La filosofia che si cerca di usare nel mondo Internet e di separare i problemi. Quindi ogni protocollo fa la sua cosa separata ortogonale dagli altri protocolli che risolvono altri sottoproblemi.

Una cosa importante da ricordare di SIP è come gestisce il problema della mobilità. In questo caso, se il destinatario cambia locazione, non è possibile chiamarlo direttamente, è necessario passare per altri elementi di rete. Questo perché non si è a conoscenza dell'IP del destinatario, dato che si è spostato. Questo viene risolto tramite i proxy SIP, (quindi tramite indirezione) e i registrar che servono a tenere traccia degli utenti e della loro posizione attuale. Il proxy SIP che è quello che al posto nostro va a gestire l'instaurazione della chiamata. È che un po come servizio di posta elettronica non sono gli utenti che da soli devono risolvere il problema del trovare il destinatario eccetera.

Uno scenario tipico:



In questo scenario, Bob, che si è spostato, ha dovuto inoltrare un messaggio di REGISTER al Registrar SIP locale, che ha informato a sua volta il Registrar di Bob che si trova in quella rete.

Quando Alice vuole conoscere l'indirizzo di Bob, invia un INVITE al suo Proxy locale, il quale contatta il Registrar di Bob per conoscere la posizione di Bob. Il registrar risponde indicando l'IP del Registrar a cui Bob si è collegato l'ultima volta. Di conseguenza, il Proxy di Alice contatta il proxy SIP attuale di Bob che a sua volta contatta Bob inoltrandogli l'INVITE. Se Bob decide di accettare, verrà inoltrata intiero la risposta verso Alice che saprà ora l'IP di Bob e potrà instaurare una conversazione.

# Supporto della rete al traffico multimediale

Approccio	Granularità	Garanzie sul servizio	Meccanismi applicati	Complessità	Utilizzo
Sfruttare al meglio il best effort di IP	Tutto il traffico è trattato il modo equo	Nessuna	Nessun supporto dalla rete, tutto sulle applicazioni	Bassa	Ovunque
Diff-service	Il traffico è suddiviso in classi	Nessuna	Marking dei pacchetti, scheduling e policing	Media	Alcuni
QOS per singola connessione	flusso per singola connessione	Leggera. Alta se il traffico è stato ammesso	Marking dei pacchetti, scheduling, policing e call admission	Alta	Nessuno o pochissimi

Discutiamo il supporto che la rete, a livello IP, può fornire al traffico multimediale con l'obiettivo di garantire delle qualità del servizio a questo tipo di traffico che abbiamo visto avere dei requisiti da stringenti in termini di qualità del servizio. Cosa può fare quindi la rete per venire incontro a queste necessità? Ci sono tre grosse filosofie di approcci possibili. Dal più semplice al più complesso.

## Sfruttare al meglio il best effort

Il primo approccio è quello di sfruttare al meglio il servizio best effort nativo di Internet. E nella tabella sopra sono indicate le caratteristiche di questo primo approccio:

- Tutto il traffico viene trattato equamente, quindi di fatto di non distinguere tra diverse classi di traffico
- L'approccio di per sé non fornisce nessuna garanzia.
- Non richiede supporto da parte della rete e nessun supporto a livello applicativo, quindi non richiede di modificare la rete esistente, di conseguenza
- La complessità è bassa.
- È la soluzione comunemente più adottata al giorno d'oggi

Quindi uno si può chiedere, ma allora come fa ad essere la soluzione più adottata se non facciamo niente a livello di rete e a livello applicativo? Qual è l'idea? L'idea è quella di agire a livello fisico. Essenzialmente, se si riesce a tenere i link sufficientemente scarichi e quindi, aggiungendo sufficiente capacità sui link della rete rispetto alla domanda di traffico, non avviene congestione e di conseguenza i ritardi e le perdite sono trascurabili. Quindi, con un'opportuno dimensionamento dei link, e lasciando che la rete offra il suo servizio best effort, nativo di fatto, offrendo qualità del servizio a tutti i flussi, si riesce a garantire un servizio di buona qualità.

Ovviamente questo ha un costo, cioè il costo di sovrardimensionare le reti, ad esempio, supponiamo che si debba far passare un traffico aggregato 1Gb/s. L'idea è che, invece di dimensionare la mia rete esattamente con una capacità di trasporto di un Gigabit, la dimensiono su 10 Gb/s. Allora avrò una rete per lo più scarica, con una bassa utilizzazione. Inoltre, mi aspetto che ci siano perdite trascurabili. Quello che costa ovviamente è che bisogna dimensionare 10 volte di più di quello che mi servirebbe. La semplicità di questo meccanismo è data dal fatto che lascio tutto intatto a livello di stack protocollore: non si implementa nessun particolare.

Questo approccio richiede di conseguenza di risolvere il problema del **network dimensioning** e cioè di capire quanta banda è sufficiente (e quindi necessaria) sui link della rete. Bisogna quindi

dimensionare opportunamente la capacità di trasporto sui vari link della rete e questo richiede ad esempio di stimare la domanda di traffico, sapere ad esempio qual è il flusso che entra in un certo punto nella rete e che esce da un'altra parte della rete. Questo problema di stima della banda e di dimensionamento della rete sono problemi relativamente semplici da risolvere ed inoltre non richiede di implementare nessun particolare meccanismo per fornire qualità del servizio, perché questa viene automaticamente garantita dal fatto che la rete è sovradimensionata. Il dimensionamento viene effettuato nella parte di dorsale di Internet, quindi non sono operazioni che si fanno comunemente. Infatti la maggior parte dei problemi di congestione o di perdita di pacchetti avvengono nella parte di accesso (reti di accesso) appunto perché la dorsale è sovradimensionata.

## Approccio Diff-Serve per il traffico multimediale

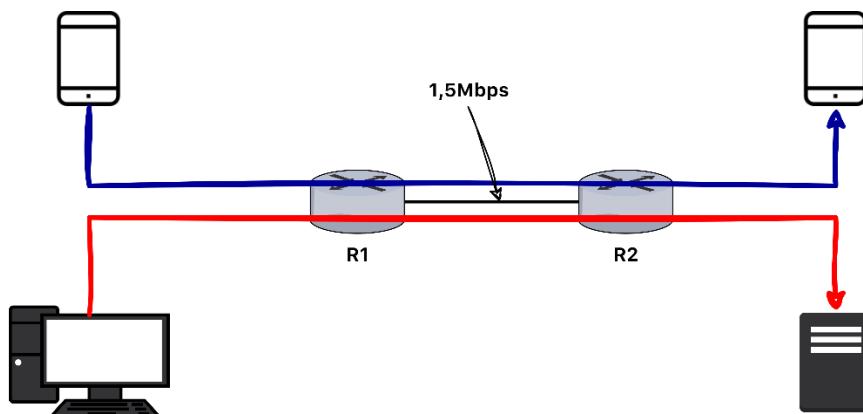
Un approccio un po meno banale è quello in cui invece si cerca di dividere il traffico in classi e di fornire qualità del servizio diverso alle varie classi (viene chiamato diff-serve, differentiated services).

Una prima cosa che evidentemente è necessario fare, è essere in grado di partizionare il traffico in classi, quindi, essere in grado di marcare i pacchetti identificando a che classe appartengono. In questo modo, si costruisce quindi un'insieme di classi che sono dei grandi aggregati di flussi di vario tipo. Attenzione che qui non si da qualità di servizio specifica ai singoli flussi ed a singoli connessioni, ma a grandi aggregati di traffico.

Per fare un esempio, si può pensare a una classe Gold, una classe Silver e una bronze che cerco di trattare diversamente, dando un po più qualità del servizio alla classe Gold e così via. Gold > Silver > Bronze.

Questa idea delle classi è vecchia, già quarant'anni fa avevano previsto un campo nell'header IP il Type of Service (ToS), riservando un certo numero di bit proprio a specificare una classe di servizio. Questo campo Type of Service non ha avuto particolare successo, è stato scarsamente utilizzato. Anche se, appunto, era stato previsto sin dall'inizio il supporto per marcare i pacchetti tramite questo campo e che è presente nativamente fin dalla versione quattro di IP.

Per capire i problemi che nascono quando si adotta un approccio Diff-Serve, consideriamo un semplice scenario in cui ci sono diverse classi che condividono uno stesso link:



Uno scenario a singolo collo di bottiglia dove la parte critica è il link che collega il Router R1 al Router R2, che va a strozzare le prestazioni dei flussi che passano attraverso questo link. Il link ha capacità di un 1,5 Mbps e si trattano due classi (blu e rossa). Inoltre, in questo scenario supponiamo che la parte di accesso abbia una capacità invece pressoché illimitata, infinita e che quindi non ci siano problemi all'infuori del bottleneck.

Immaginiamo di voler dare un servizio migliore al traffico di tipo blu, quindi offrire una qualche garanzia di servizio al traffico blu. Potrebbe succedere una cosa di questo tipo: è in atto una telefonata VOIP che viaggia sul flusso blu e contemporaneamente un trasferimento di file su HTTP che viaggia sul flusso rosso. Il traffico VOIP viaggia ad un CBR che dipende dalla codifica che usa e genera un traffico abbastanza costante (supponiamo 1Mb/s).

Se non si interviene in nessun modo, quindi se si usa il best effort, cosa succederà in questo scenario? Che cosa succede se non distinguo i pacchetti? Andrà tutto bene o succederà qualcosa di strano?

In sostanza quello che succede è che UDP non farà controllo della congestione, quindi UDP continuerà a sparare al suo CBR di 1 Mbps e non adatterà il suo tasso di trasmissione in nessun modo, quindi il VOIP manda fisso regolare 1 Mbps che è quello che gli serve. Quindi, se ci fosse solo il flusso blu non ci sarebbe congestione perché il flusso blu occupa 1 Mbps, su un link che ne trasporta 1,5Mbps, quindi di per sé il flusso blu non causa congestione. Il problema nasce quando si aggiunge il traffico TCP rosso, in quanto, TCP è molto aggressivo per sua natura: cerca di portare la rete in congestione il che satura la capacità del link.

Se si satura la capacità di un link, di conseguenza si creerà un buffer congestionato con tanti pacchetti in attesa, quindi un grosso ritardo di accodamento nell'interfaccia di uscita del router R1 verso R2 che potrebbe introdurre delle perdite e dei ritardi che vanno oltre 400 millisecondi (se si suppone che il buffer è molto grosso, diciamo che può raccogliere un sacco di pacchetti, posso tranquillamente arrivare a subire un ritardo di accodamento superiore a 400 millisecondi).

Quindi se si lascia UDP competere con TCP senza far niente, la comunicazione VOIP in uno scenario di questo tipo molto semplice, non funzionerà.

La soluzione banale di cui abbiamo parlato prima è sovradimensionare la rete. In questo modo, anche se c'è di mezzo un traffico TCP che disturba il traffico Voip, il ritardo e la perdita saranno trascurabili. Cerchiamo adesso di non ricorrere a questa situazione, ma mantenere sempre questa banda limitata di 1,5Mbps e di utilizzare l'approccio Diff-Serve.

## Classificazione

La prima cosa che evidentemente bisogna fare è la classificazione dei pacchetti. In particolare, il router all'edge della rete (R1) deve analizzare i pacchetti e classificarli, cioè distinguerli (si dice effettua packet marking). La classificazione è necessaria se si vuole che la rete tratti i due flussi diversamente (**diff-serve**).

Successivamente, non si andranno ad inserire tutti i pacchetti in un unico buffer di uscita, indistinti, mescolati tra di loro, ma si creeranno code diverse, entrambe affacciate sul link R1 <-> R2. Si creeranno quindi due buffer virtuali, uno che ospita tutti i pacchetti blu e uno che ospita tutti i pacchetti Rossi.

Le classi si possono decidere in base a vari criteri:

- Sulla base del protocollo di trasporto, quindi TCP/UDP
- Le porte (danno un'idea del dell'applicazione che sta girando)
- range di indirizzi IP
- etc.

## Scheduling

Dopo la classificazione, interviene il secondo meccanismo da attuare in una rete diff-serve che è un meccanismo di scheduling dei pacchetti, che deciderà di volta in volta se trasmetterne un pacchetto blu o uno rosso verso il link di uscita (questa politica di scheduling dei pacchetti interverrà sul link di uscita del router).

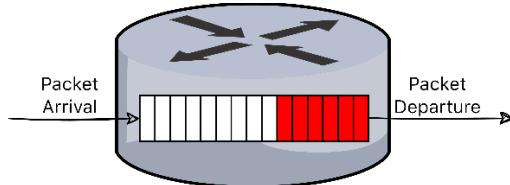
### Principio di isolamento

È facile capire perché si necessita del principio di isolamento se si analizza questo scenario: si suppone di dare **priorità secca** al traffico VOIP per cui se si hanno dei pacchetti blu da trasmettere, si trasmettono sempre per prima quelli. In questo caso è il flusso rosso che è in pericolo di congestione. Inoltre, il traffico VOIP visto che ha priorità secca, supponiamo che decida di aumentare la codifica audio e quindi il bitrate, quello che succede questa volta è che il traffico rosso viene completamente strozzato. E quindi da qui nasce la necessità di fornire anche un isolamento, una protezione di una classe dall'altra, ovvero, se una classe si comporta male e cioè, manda ad un traffico più alto di quello che ci si aspetta che faccia, la rete deve difendersi e in particolare deve difendere le classi che si stanno comportando bene, diciamo le classi deboli. In questo caso la classe debole è la classe rossa.

D'altra parte, contemporaneamente però, voglio anche fare un utilizzo efficiente della banda. Quindi non voglio sprecare risorse allocando staticamente della banda a dei flussi: può succedere che se poi quel flusso non sfrutta tutte le risorse, la banda che gli ho riservato è andata sprecata. Ad esempio, l'idea di dedicare al traffico blu una porzione fissa di banda di 1 Mbps e a quello rosso 0,5 Mbps non è efficiente.

Quindi il secondo principio che bisogna soddisfare è **l'utilizzo efficiente delle risorse**. Questo è possibile farlo se si adottano dei meccanismi intelligenti di scheduling dei pacchetti nelle code di uscita dei router.

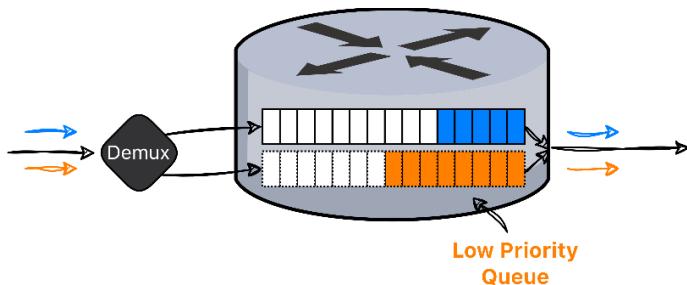
Uno degli algoritmi di default, quando non si effettua nessun trattamento diverso per classi, è la politica **FIFO** (first in first out), in cui è presente un'unica coda dove tutti i pacchetti si accodano: chi arriva prima esce per prima.



Qui i pacchetti vengono accodati in ordine di tempo trasmettendo per primo quello che è entrato per primo. FIFO è la disciplina più usata che vediamo attorno a noi (applicata alla vita quotidiana) però non è l'unica applicabile alle reti e soprattutto non è la più efficiente.

Il secondo algoritmo che vediamo è lo scheduling a **priorità secca**. Questa volta non si ha un'unica coda dove è tutto mescolato, ma N code virtuali. Dopo aver fatto packet marking, i pacchetti che arrivano ai router intermedi devono scegliere chi trasmettere sui link. Nello scheduling a priorità secca si predilige prima una coda rispetto alle altre. Ad esempio, se dessimo priorità alla blu, il router:

- Se ha pacchetti da servire nella coda blu serve sempre prima loro.
- Quando la coda blu si svuota, allora il router si affaccia sulla coda arancione e se ci sono pacchetti li serve (ma solo quando la coda blu si è svuotata).



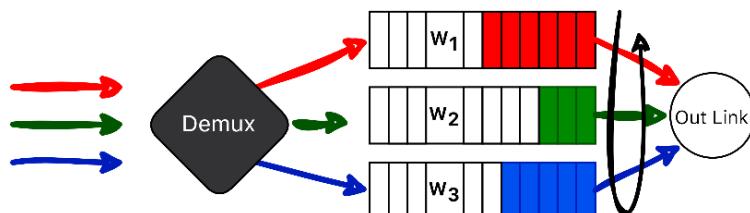
Il problema di questa politica è che se si hanno pacchetti blu che continuano ad arrivare, di conseguenza la coda arancione è sempre piena e il flusso arancione muore completamente (va in starvation). Quindi questo meccanismo di priorità secca è soggetto al problema della starvation e non rispetta l'isolamento.

L'esempio è con due classi, ma è possibile implementare il meccanismo con un numero arbitrario di classi, ordinate in ordine di priorità, quindi dalla più importante a quella meno importante.

Un'altra politica di scheduling che si vede anche in molti altri contesti è la politica **RR (round robin)**. In questa politica, si visitano ciclicamente le code e di ogni classe si estrae un pacchetto (se c'è) trasmettendolo sul link, dopodiché si passa alla classe successiva. Quando il router ha fatto il giro completo (ha visitato tutte le code esistenti), ricomincia dalla prima. Un pregio di round robin è che non permette di privilegiare una classe sull'altra (nella sua versione standard) ed inoltre risolve il problema della starvation rispettando l'isolamento. Quindi il meccanismo round robin da intrinsecamente un meccanismo di isolamento, di protezione di una classe dall'altra, perché garantisce che nella peggiore delle ipotesi (supponiamo con due flussi, uno rosso e un altro arancio) il flusso rosso si prenda sempre metà delle risorse ( $1/N$  con  $N$  code). Si nota però, che la rossa potrebbe prendersi più di metà delle risorse se ci sono degli istanti in cui non si hanno pacchetti arancioni, questo perché se una delle code è vuota, il router non si affaccia.

Il problema di round robin, per la sua versione base, è che tratta tutte le classi in modo equo non privilegiando, per dire, la Rossa sulla Arancione. Se si vuole fare Diff-Serve, questo non è sufficiente. Esiste quindi una versione più avanzata di round Robin, che permette di dare servizi diversi alle varie classi e che si chiama **WFQ** (Weighted fair queuing).

**WFQ** generalizza il round robin permettendo di dare servizi diversi alle varie classi ed evitando contemporaneamente il problema della starvation e rispettando l'isolamento.



In WFQ si assegna un peso (numero reale)  $W_i$  alle varie classi. Come nell'immagine supponiamo di dare:

- Verde ( $W_2$ ): 1
- Rosso ( $W_1$ ): 2
- Blu ( $W_3$ ): 3

Qual è l'obiettivo della politica? È quella di assegnare a ciascuna classe, una porzione di banda, proporzionalmente al suo peso. Ad esempio, la classe rossa che pesa 2, avrà una capacità di (una banda minima garantita):

$$\frac{2}{2+1+3}$$

Questo nel caso in cui tutte le code sono occupate e quindi c'è almeno un pacchetto. Nel caso in cui una coda fosse vuota non viene contata nel denominatore. Quindi in generale, la banda minima garantita per un flusso  $W_i$  è:

$$\frac{W_i}{\sum_{j:W_j>0} W_j}$$

dove al denominatore c'è la somma di tutti i pesi, ma delle code non vuote. Nel caso in cui una delle code fosse vuota e quindi non compare al denominatore, la classe riceve più di quello che era previsto. WFQ quindi offre delle garanzie sulla banda minima garantita data da questa formula.

WFQ è la soluzione più elegante di tutte quelle che abbiamo visto, la più complessa, ma quella più flessibile, dove ottengo tutto quello di cui si ha bisogno: l'isolamento dalle classi, protezione di una classe dall'altra, non c'è starvation e possibilità di dare priorità alle classi a seconda dei pesi.

## Policing

L'ultima cosa di cui si ha bisogno se si vuole implementare un meccanismo di diff-serve in una rete, oltre alla **classificazione** e allo **scheduling** dei pacchetti, c'è il problema di difendere la rete da inondazioni di traffico da parte di una o più classi che potrebbero intasare la rete. È necessario quindi introdurre delle policy.

Per questo, ai bordi della rete, si cerca di stipulare dei contratti coi clienti mettendosi d'accordo su una quantità massima di traffico che i clienti possono mandare dentro la rete e successivamente implementando dei meccanismi di policing in modo da costringere gli utenti ad adattarsi a questi limiti che sono stati stabiliti a priori con chi fornisce il servizio di connessione. Questi contratti che vengono chiamati **SLA** (service level agreement), sono basati, a livello quantitativo, da delle metriche.

Una di queste metriche potrebbe essere il **tasso di trasmissione medio (average rate)**. Ad esempio, ai bordi della rete, si potrebbe accettare un certo tipo di traffico in modo che i pacchetti arrivino con una media di massimo 100 pacchetti al secondo. Questo è un primo criterio di ammissione basato sul rate medio di trasmissione.

Il problema di accontentarsi di questa singola metrica, è il fatto che è estremamente critica la finestra di osservazione che uso per calcolare questo traffico medio, ad esempio: se la finestra di osservazione è di 1 minuto e se si suppone di essersi messi d'accordo per trasmettere 100 pacchetti al secondo, allora un flusso che manda 100 pacchetti al secondo e uno che ne manda 6000 in un minuto, hanno la stessa media giusto?

Potrebbero quindi arrivare un burst di 6000 pacchetti che arrivano tutti nel primo secondo e poi la sorgente sta zitta per i restanti 59 secondi. La sorgente sa che se comportandosi così sta rispettando il suo vincolo in trasmettere 100 pacchetti al secondo. Però dall'altra parte ha mandato un burst di 6000 pacchetti, tutti nel primo secondo, il che non è bello.

Quindi avere soltanto un riferimento medio non è sufficiente. Quello che di solito si fa è affiancare, oltre ad un valor medio di tasso di trasmissione, una di queste altre misure: **pick rate** e **burst size**.

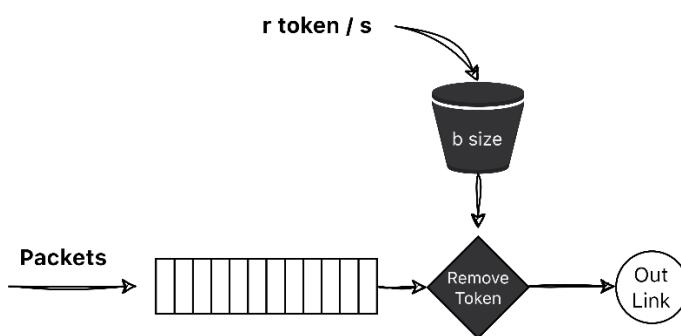
Uno è il **pick rate** (traffico di picco), cioè il massimo bitrate che la sorgente può mandare. Quindi si impone ad una certa classe che non può mai trasmettere più del pick rate in qualunque istante di tempo.

In alternativa al pick rate si può definire il **burst size**, quindi definire il massimo numero di pacchetti consecutivi che possono entrare nella rete, senza buchi, quindi uno di seguito all'altro. Un burst size di 50 vuol dire che non potrà mai capitare lo scenario che abbiamo visto prima, dove vengono sparati 6000 pacchetti nel primo secondo di questo minuto di osservazione.

Quindi unendo un'informazione di traffico medio a una che limita il traffico di picco, si cerca di limitare quella che si chiama la burstiness del traffico, quindi l'impulsività del traffico che di solito non fa bene alle reti.

### Meccanismi applicati per il policing

Una volta che ci si mette d'accordo sulle SLA, poi la rete deve identificare dei meccanismi per farli rispettare, quindi dei metodi effettivi che permettono di limitare ad esempio il traffico medio e la burstiness.



Il meccanismo più semplice che è stato inventato è quello del token bucket.

Si immagini di avere un contenitore virtuale di gettoni che viene rifornito ad un tasso 'r' di gettoni al secondo. Quindi c'è un flusso continuo costante di gettoni che entrano nel bucket. Se questo bucket, come tipicamente succede, ha capacità finita, una volta che lo si satura non entreranno più gettoni. Dopodiché ci sono dei pacchetti che arrivano ed entrano in una coda di attesa.

Il meccanismo è il seguente: un pacchetto per essere ammesso dalla rete deve consumare un gettone, quindi deve esserci un gettone libero nel bucket. Po si passa al pacchetto successivo e così via. Questo semplice meccanismo ha due parametri:

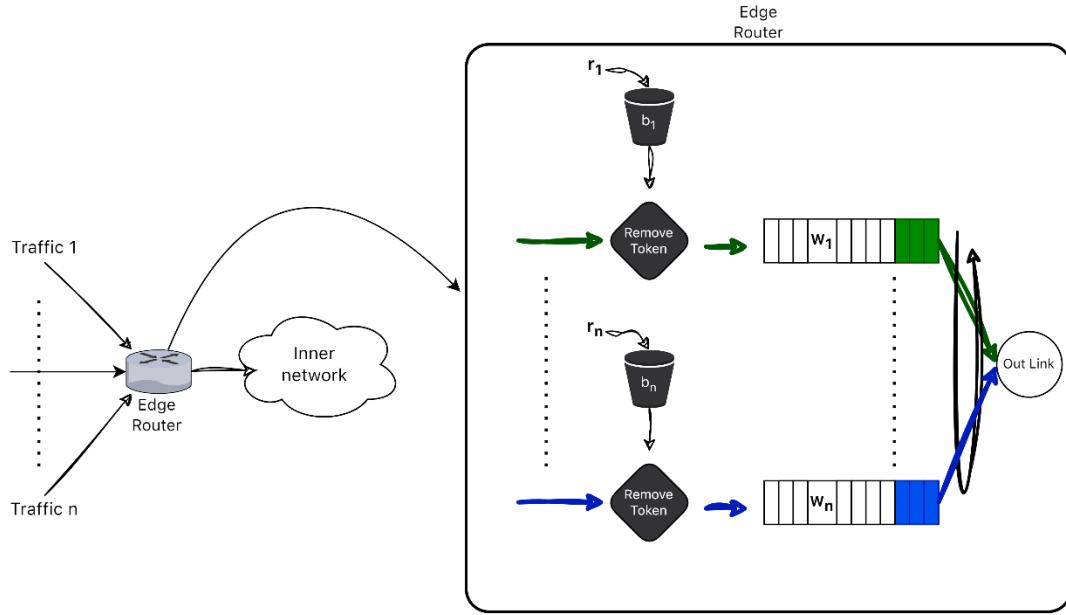
- **Il tasso 'r' con cui vengono riforniti i token** (questo è equivalente al tasso medio di trasmissione)
- **la dimensione 'b' del bucket:** il numero massimo di gettoni che possono essere messi dentro il bucket (questo è equivalente al burst size)

Avendo 'r' e 'b' si crea un meccanismo di policing che mi permette, ad esempio, stabilito un intervallo generico (t), il numero massimo teorico di pacchetti che vengono ammessi dentro la rete dentro l'intervallo 't' che è:

$$rt + b$$

Quindi in un qualunque intervallo si è certi che non ci saranno più di  $(rt + b)$  pacchetti che passano il Token Bucket. Così si è in grado di regolare il flusso, in particolare, regolandone sia il valor medio sia la burstness.

A questo punto, avendo inventato sia un meccanismo di policing sia un meccanismo di scheduling, si possono mettere assieme ottenendo un meccanismo congiunto, che è in grado davvero di fornire una qualità del servizio alle classi. In particolare, le cose migliori si notano quando si combinano il Token Bucket con WFQ.



Quindi (come da immagine) ci sono  $n$  flussi di traffico e c'è un router di frontiera dove è implementato WFQ con il meccanismo di token bucket. Si dimensionano i pesi del WFQ in modo da garantire, ad esempio, al flusso Verde una banda  $r_1$  e un bucket di size  $b_1$ . Questo viene effettuato per tutte le varie classi, ottenendo di fatto uno schema, come si vede nell'immagine (parte destra).

Quello che si dimostra è che i pacchetti che vengono ammessi dalla rete (quindi sono conformi al traffico che si è pattuito coi clienti) ricevono una banda garantita ' $r$ '.

È possibile inoltre dare una garanzia sulla latenza. Quindi la latenza peggiore che può mai capitare ad un pacchetto che supera il Token bucket ed è ammesso dentro la rete (si può dimostrare facilmente) è uguale a:

$$\frac{b}{r}$$

' $b$ ' è la dimensione del Token Bucket. Quando un pacchetto è l'ultimo di un burst, quindi quando entra dentro la rete si trova davanti una coda lunga ' $b$ '. Dovrà aspettare  $b/r$ .

Tutto questo viene implementato agli edge router. Router di frontiera dove si fa l'ammissione quindi il policing, il marking dei pacchetti e la classificazione dei pacchetti come conformi e non conformi, quindi io posso anche marcare i pacchetti di ingresso come "in-profile" o "out-profile".

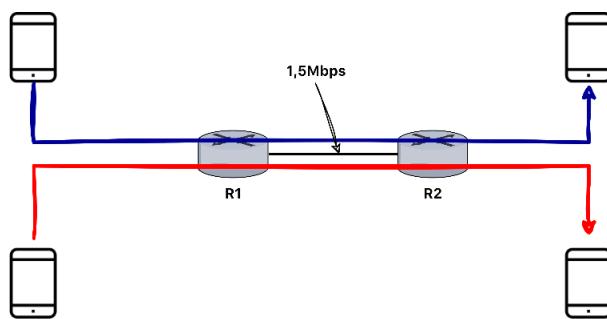
Il perché del fatto di marchare un pacchetto "in/out-profile" è che, anche se io mi sono messo d'accordo con dei clienti di una certa classe, secondo delle misure di traffico medio e di picco, poi posso anche accettare i pacchetti in eccesso che non sono conformi perché si hanno risorse sufficienti per elaborarli, risorse sufficienti per trasmetterle.

Quelli non conformi vengono marcati come out-profile, quindi questi pacchetti potrebbero essere degli ottimi candidati a essere trattati male quindi a essere scartati per primi o essere messi in una coda che è servita con bassa priorità, con basso peso.

## Per-connection QoS

L'approccio più complesso per dare supporto al traffico multimediale è quello di fornire qualità del servizio alle singole connessioni. Quindi l'approccio più ambizioso è quello che ogni singola connessione riesce a ottenere dalla rete delle garanzie di qualità del servizio.

Per fornire qualità del servizio alle singole connessioni, oltre alle cose che avevamo visto essere necessarie nell'approccio differentiated Services, ovvero il marking dei pacchetti, lo scheduling ed il policing, si aggiunge anche il **Call-admission**. Per giustificare la necessità di questo ulteriore meccanismo è sufficiente vedere un esempio:



C'è il solito collo di bottiglia con capacità di 1.5Mbps e due connessioni Voip che richiedono 1 Mbps ciascuno. Evidentemente non possono passare entrambi. Non c'è abbastanza capacità per ammetterle entrambe, quindi, la prima che inizia a trasmettere le alloca 1Mb di banda, supponiamo la blu. Quando la rossa vuole trasmettere, anche lei chiederà il suo CBR di 1Mbps di banda (ma che la rete non ha). L'unica cosa che può fare qui la rete, è bloccare proprio il flusso rosso. Questo perché se lo facessimo partire si distruggerebbero tra loro le due comunicazioni VOIP, cioè non ne funzionerebbe neanche una (troppe congestioni e ritardi). Quindi a quel punto, invece di fare un disastro per tutti, si ammettono quelle che la rete è in grado di gestire/di servire e quelle che non hanno risorse a sufficienza vengono bloccate sul nascere, cioè non vengono neanche fatte partire. Questo porta a una certa probabilità di blocco della chiamata.

La stessa cosa che succedeva (adesso un evento assolutamente improbabile) nella rete telefonica tradizionale, che è basata su commutazione di circuito, dove c'è davvero il concetto di chiamate che vengono allocate nella rete. La rete a un certo punto non poteva far partire la chiamata perché non c'erano risorse nella rete, allora dava il tono di occupato, per avvertire che la rete non avesse un circuito da dedicare alla chiamata in quel momento. L'analogia di questa cosa che succedeva nella rete telefonica, accade anche in uno scenario Internet dove vogliamo fare le cose flusso per flusso. Quindi c'è una procedura iniziale di call admission necessaria per capire se ci sono abbastanza risorse da allocare per una trasmissione.

Oltre a fare call admission, quello che è necessario fare se si vuole offrire qualità del servizio a singole connessioni è una fase di **call setup**. Quindi, prima che il traffico dati fluisca in rete, è necessario fare una segnalazione dentro la rete per prenotare le risorse richieste su tutti i link e quindi su tutti i router e i link attraversati dalla connessione.

In sostanza bisogna far girare dei protocolli che lungo tutto il cammino del flusso prenotano/riservano una porzione di banda per quel flusso, in modo che tutti i router siano informati che c'è un flusso che li attraverserà e che ha bisogno di una certa banda garantita.

Il modo con cui poi si garantisce questa banda in una rete a pacchetto è ad esempio quello di implementare WFQ che garantisce una banda minima garantita a una certa classe, ad un certo flusso di pacchetti, quindi la nostra connessione sarà trattata come una classe di pacchetti a cui WFQ garantisce una certa banda minima garantita. Questo va fatto su tutti i router però, prima di davvero far partire il flusso, in quanto, anche se solo uno dei router non dovesse avere risorse necessarie, il tutto è impossibile, la chiamata verrà bloccata.

Facile intuire che poi alla fine, quando il flusso termina, bisogna rilasciare le risorse che si sono prenotate e quindi ci sarà una nuova fase di segnalazione che prende il nome di **Tear Down**, dove vengono rilasciate tutte le risorse precedentemente prenotate.

Questo ultimo modo di differenziare i singoli flussi alla fine non è mai nato. C'è qualcuno che ha provato a standardizzare tutto questo, cioè ha provato a inventare dei protocolli per riservare banda ai singoli flussi, uno dei protocolli più famosi è **Resource Reservation Protocol RSVP**. Questo approccio è fallito perché esistono altri approcci che, con molta meno complessità, vanno decentemente meglio. Inoltre, questo approccio è impraticabile in quanto la complessità risiede nei router e non sui nodi terminali: richiede di instaurare uno stato su tutti i router, quindi tutti i router devono essere informati, mantenere un'informazione di stato dei flussi correnti. Internet però è un mondo estremamente complesso, cioè i router del centro di Internet sono attraversati da centinaia milioni di flussi al secondo e sarebbe ingestibile mantenere qualità del servizio a singoli flussi, andando a instaurare uno stato su tutti i router. Tutto ciò non scala ed è il contrario della filosofia di Internet dove il centro della rete deve essere stateless e complessità solo sui bordi.

# Principi Architetturali di Internet

[Questo articolo di Clark dell'88](#), discute dei principi architetturali di Internet mettendole a confronto tra le esigenze che c'erano trent'anni fa (quindi prima che Internet esplodesse, prima dell'avvento del web) con le esigenze di oggi. È abbastanza importante ragionare su questo, perché tanti problemi che hanno le reti oggi si capiscono, si interpretano, solo ragionando sulla storia che hanno avuto tutti i protocolli e tutte le scelte che sono fatte a partire da quando Internet è nata a metà degli anni 70. Solo tenendo conto di questa storia si capisce come mai siamo arrivati allo stato in cui siamo adesso.

Questa storia ha portato allo sviluppo dell'architettura di internet che originariamente è stata pensata alla fine degli anni 70, che poi si è consolidata nel tempo e che adesso è troppo tardi per cambiarla. Sono stati stanziati molto soldi e con tantissima fatica sviluppati un sacco di protocolli (ormai ci sono oltre 6000 RFC dalla IETF).

Anche se Internet continua a evolvere e quindi nuovi protocolli e nuove applicazioni appaiono, i principi base su cui è nata sono ormai immodificabili: ormai l'infrastruttura che è stata messa in piedi è enorme in termini di apparati, di link, etc. "Il mostro è diventato così grande che non è più pensabile partire con una nuova Internet da zero, è un mostro più grande di noi e ormai lo teniamo così com'è".

L'articolo che ha scritto David Clark è abbastanza utile in quanto lui è stato partecipe fin dall'inizio alle scelte architetturali di base di Internet.

Sotto è elencata la lista in ordine di importanza dei principi che hanno guidato l'architettura come la conosciamo oggi. Si tratta di scelte fatte cinquant'anni fa, quando il mondo era completamente diverso.

1. Connessione delle reti già esistenti
  - a. inizialmente: ARPANET, ARPA packet radio, rete a pacchetto satellitare
1. Sopravvivenza (survivability)
  - b. garantire comunicazione anche in caso di guasti a nodi/link
2. Supporto a molteplici tipi di servizi
3. Possibilità di estensione a grande varietà di reti
4. Permettere gestione distribuita
5. Permettere che un host si attacchi alla rete con minimo sforzo
6. Efficiente in termini di costi
7. Permettere monitoraggio e tariffazione delle risorse utilizzate

Internet è nata inizialmente come un esperimento, finanziato dal dipartimento della difesa americana con lo scopo iniziale di connettere unità di calcolo, centri di ricerca e università negli Stati Uniti (progetto ARPANET). Era una situazione molto sotto controllo dove i ricercatori università avevano iniziato a scrivere i primi protocolli e le prime soluzioni. Non c'era ancora l'Internet commerciale, perché a quel tempo non c'era neanche HTTP (il web è nato poi all'inizio degli anni 90 al CERN da un'idea di Tim Berners-Lee).

All'inizio quindi era solo un gruppo di universitari che provavano a connettere le loro reti locali, quindi a far viaggiare pacchetti tra un'università all'altra degli Stati Uniti. C'era l'email c'era FTP però appunto era un ambito ristretto a soli scopi di ricerca in un ambito universitario e la cosa interessante è che tutte le scelte che vediamo ancora oggi si capiscono solo pensando a questa origine.

Da notare che nella lista del 1988 non c'è neanche un accenno alla sicurezza, mentre nella lista di oggi è al primo posto. Il motivo per il quale non compare è che all'inizio (ARPANET), non c'era neanche la minima idea di un utente malintenzionato. Non c'era il concetto di un uso cattivo della rete fatta da soggetti che volessero rubare dati, fare atti maliziosi. uno dei motivi per cui è stato finanziato il programma ARPANET è stato quello di appunto inventare "un collante", un meccanismo che permette di scambiare dati tra reti già esistenti di natura molto diversa, dislocati sul territorio.

## **Sopravvivenza**

Una delle ossessioni iniziali che hanno avuto gli architetti di Internet era la survivability (sopravvivenza), cioè l'obiettivo era di creare una rete il più possibile robusta, tollerante ai guasti e che fosse in grado di autoconfigurarsi e che quindi avesse dei meccanismi distribuiti che le permettessero di sopravvivere, di continuare a funzionare anche in presenza di improvvisi guasti sui link/router. Questo è in parte dovuto al fatto che il progetto ARPANET fosse stato finanziato dal dipartimento della Difesa degli Stati Uniti e quindi realizzare una rete che continuasse a funzionare, cioè a far parlare due nodi, sempre in qualunque caso, tranne nel caso proprio disperato in cui avvenisse una partizione della rete, cioè il fatto che due nodi non fossero proprio più neanche connessi da nessun link, negli altri casi la rete doveva garantire sempre la comunicazione e il routing doveva autoconfigurarsi sempre. L'algoritmo di routing che poteva garantire una riconfigurazione delle strade in modo totalmente distribuito, robusto, tollerante ai guasti che inizialmente era tanto piaciuta agli inventori di Internet era il distance Vector (che poi è stato l'approccio meno seguito) davvero in grado di riconfigurare dinamicamente il routing in modo completamente distribuito e rimediando anche a guasti di link/router.

Per rendere la rete il più possibile robusta ai guasti, scelsero anche di mantenere lo stato solo a livello di trasporto, nei nodi terminali, e di mantenere a livello rete un'architettura stateless senza stato: i router assolutamente non dovevano installare nessuno stato spingendo il mantenimento dello stato ai bordi. Si rendeva appunto la rete intrinsecamente più robusta ai guasti perché in una rete statefull, dove i router hanno uno stato da mantenere, tutte le volte che si guasta un link / si rompe un router, c'è bisogno di ripristinare questo stato che è stato perso dentro la rete e che è quasi impossibile da recuperare. La soluzione si risolve alla radice rimuovendo qualunque soluzione statefull dentro la rete, quindi mantenendo la rete stateless. Questa era la l'ossessione legata alla sopravvivenza alla survivability.

## **Supporto a molteplici tipi di servizi**

Per quanto riguarda i tipi di servizio, all'inizio non si era previsto di avere due protocolli di trasporto diversi (TCP o UDP), all'inizio infatti c'era un'unica soluzione: TCP e IP erano integrati insieme.

Successivamente si è capito che c'era davvero necessità di supportare applicazioni diverse e che quindi certe applicazioni avevano bisogno di consegna garantita e la ritrasmissione dei pacchetti persi, mentre quelle più Real Time avevano requisiti completamente diversi. Sin dall'inizio quindi (quando non c'era Discord, Skype, non c'era nessuna applicazione VOIP come la immaginiamo oggi) gli architetti di Internet erano stati così preveggenti da capire che ci sarebbero state applicazioni con esigenze diverse, in particolare, avevano già immaginato che un giorno sarebbero state sviluppate applicazioni multimediali, audio e video e quindi decisero successivamente di separare TCP da IP, affiancando invece un protocollo più snello a livello di trasporto (UDP) che era svuotato di tutta la complessità di TCP e che offriva semplicemente un servizio best effort non orientato alla connessione.

Emerse inoltre il concetto di datagram: le unità dati che viaggiano nella rete sono dei pacchetti detti datagram e questo fa da comune denominatore su tutti i servizi di livello superiore. Questo servizio

elementare datagram doveva essere minimale, cioè non si chiede nulla alla rete se non di fare il suo miglior sforzo per consegnare un pacchetto da un nodo sorgente a un nodo destinazione.

Gli architetti di Internet sono stati poi così preveggenti che fin dall'inizio hanno inserito nel nell'header il famigerato campo Type of Service, per la differenziazione dei flussi (che appunto non fu mai usato).

Grazie al fatto che questa astrazione del datagram ha avuto un'enorme successo perché ha permesso di collegare una serie di reti sottostanti (tecnologie di livello uno e due), molto variegate, tutte unite da questo unico collante che è il livello IP, a un certo punto si è iniziato a trasportare pacchetti IP su qualsiasi cosa. C'erano queste reti primordiali tra cui ARPANET, satellitari, X.25, tutte con capacità molto diversa (breve distanza, lunga distanza, bassa capacità, alta capacità, scenari completamente diversi) Interconnessi da IP. Ancora oggi c'è una pluralità di tecnologie su cui IP gira, da ethernet a wifi, le reti cellulari, fibre ottiche.

## Obbiettivi Secondari

Tra gli obbiettivi secondari invece, c'era quello di permettere una **gestione distribuita**.

Questo è dovuto al fatto che Internet ha connesso reti diverse di amministratori diversi, ogni amministratore si gestisce il suo pezzo in modo diverso e gli amministratori interagiscono solo ai bordi delle reti (punti di peering) dove si prendono accordi tra i diversi amministratori delle reti. In questo modo Internet gestito in modo distribuito, non c'è un'unica entità che è in grado di controllarla in modo centralizzato.

Questo modello complica però l'instradamento, che diventa molto più complesso. Inoltre, diventa complesso anche fornire qualità del servizio end to end in quanto, è evidente che fare qualità del servizio significa garantirlo su tutte le reti attraversate, ma se queste sono gestite da organizzazioni da amministratori diversi bisogna metterli d'accordo, e la cosa diventa complessa, quasi impossibile. Se ci si mette BGP poi.. ggwp, betterJungWins.

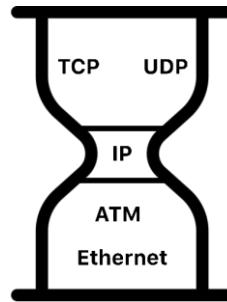
Una cosa che inizialmente era davvero in fondo alla lista era di ottenere una **una rete poco costosa**. Questo perché essendo finanziati dal governo avevano praticamente budget illimitato, tenendo conto che sono anche gli anni in cui si finanziava la missione sulla luna. Ma oltre al fatto appunto del costo economico, non era inizialmente neanche sentita un'esigenza particolare quella di ottenere delle soluzioni efficienti in termini di consumo, di banda e di risorse. Ad esempio telnet da remoto che manda un singolo carattere per volta, quindi ogni carattere digitato da tastiera, un byte, veniva trasmesso dentro un pacchetto. C'era un overhead spaventoso: con tutti gli header dei vari pacchetti che si sommano, c'erano decine di byte di Overhead per trasmettere un singolo byte.

Allo stesso modo, anche per il routing non si cercarono le soluzioni più perfette possibili per instradare i pacchetti in modo ottimale nella rete. Ci si accontentava invece di un routing più semplice, più robusto, ma magari non quello perfetto in ogni circostanza.

Un altro obbiettivo secondario era quello del **basso costo per connettere un nuovo host**. Questo non è evidentemente un punto di forza di Internet, in quanto, avendo spostato tutta l'intelligenza sugli endpoint, connettere un host a Internet era una cosa niente affatto semplice: tutta la difficoltà di configurarlo inserendo gli indirizzi, dns, gateway, etc.

Un'altra cosa, come già detto, che non fu originariamente concepita è la possibilità di avere host malevoli quindi host che sfruttassero protocolli implementati in modo sbagliato con errori, con bug. Si viveva in questa illusione che gli host, gli utenti, fossero buoni.

Altra cosa che non fu originariamente concepita è la **possibilità di fare la tariffazione del traffico**. Quindi, avendo interconnesso reti diverse tra loro, basate sull'astrazione del datagram, è impossibile, anche solo calcolare di una certa connessione che si instaura tra un client e un server, quante risorse ha consumato in Internet. Viceversa, l'architettura completamente opposta (quella telefonica) è proprio basata sulla tariffazione delle singole chiamate. Per internet è talmente impossibile farlo che l'unica soluzione ormai pressoché standard è dare un abbonamento agli utenti.



Questo diagramma è rimasto abbastanza famoso nella storia perché fa vedere la filosofia dell'architettura di Internet come una clessidra (hourglass), dove la parte centrale della clessidra è IP. Quindi IP fa da collante a tutto quello che c'è sopra, quindi TCP/UDP ed a tutte le infinite applicazioni che ci sono in cima, mentre la parte sotto che invece ci sono tutte le tecnologie di trasmissione di livello più basso. Cioè quindi tutto quello che passa da sopra a sotto ha un unico denominatore comune che è IP.

Questa architettura a clessidra, questa visione di Internet, è rimasta abbastanza nell'immaginario comune dei ricercatori che si è consolidata negli anni e che ha portato all'architettura di Internet che ormai ci teniamo, tentando a volte di cambiarne la filosofia di base, facendo visioni in cui si fa qualità del servizio per connessione, si cerca di immettere dello stato dentro i router infinite volte, ma come visto, è meglio lasciarlo così com'è.

## Il futuro di internet

Ci si può chiedere se ci terremo questa architettura per sempre oppure no. Qualcuno si è interrogato e si è posto questa domanda. In effetti, continuerà ad essere così per il resto dell'umanità oppure in futuro ci sarà bisogno di nuove soluzioni ed un'astrazione diversa da quella del datagram? In parte ci siamo già resi conto che alcune cose sono impossibili nella architettura attuale: la gestione delle risorse, la tariffazione e la qualità del servizio vanno un po in conflitto con un servizio best effort basato sul datagram. Di per sé, il datagram, non è la migliore astrazione per queste cose.

Ultimamente ci si è messi d'accordo un pò a fatica sulla nuova versione di IP (versione 6). È stata proposta, insieme alla nuova versione, una nuova astrazione: quella del flusso, andando proprio a riservare la possibilità di etichettare i flussi con un identificativo di flusso da mettere dentro l'header IPv6. Il problema è che è rimasta un'astrazione un po astratta nel senso che nessuno sa bene di preciso cosa sia un flusso: come lo si può identificare in modo univoco? Come lo si può dargli un identificativo che metta tutti d'accordo? In uno scenario complicato come Internet che è talmente esteso, talmente complicato che probabilmente il concetto di flusso è un'utopia.

La lista delle priorità, che è del 1988, è stata rivista alla luce attuale, rivotata dallo stesso autore David Clarke nel 2008. Infatti, nel 2008 David Clark ha fatto una versione nuova del suo articolo, riprendendo quello vecchio e annotando una lista diversa di requisiti che dà l'idea di quanto le cose siano cambiate. Mettendo al primo posto la sicurezza ovviamente

### 1. Sicurezza

2. Availability and resilience
3. Convenienza economica
4. Migliore gestione
5. Venire incontro ai bisogni
6. della società
7. Longevità
8. Predisposizione a supportare e sfruttare tecnologie future
9. Assolvere al suo compito

Poi availability and resilience (disponibilità): siamo diventati talmente succubi dell'uso di Internet 24/7 che non averlo disponibile anche solo per qualche minuto è un problema. È un problema ancora più grosso, non tanto per chi è viziato di internet, ma per le company, ad esempio, una banca, che se la rete non è disponibile per qualche secondo subisce dei danni economici spaventosi

# Attacchi alle reti

Vediamo una serie di esempi di attacchi classici alle reti che sono stati effettuati in passato (anche di 30 anni fa). Alcuni molto famosi che hanno causato davvero dei danni notevoli alle reti, ma che ora non si possono più fare, perché ovviamente sono state prese contromisure ed i sistemi operativi più aggiornati adesso non soffrono più di questi attacchi storici.

Quello che ci possono insegnare questi esempi storici sono dei pattern, quindi delle classi di attacco, degli schemi, che vengono sfruttati negli attacchi e che continuano a essere valide, quindi che possono continuare a ripetersi per nuovi tipi di attacchi. È come se questi attacchi avessero dei cicli, in quanto tendono anche a ripetersi nel tempo. Inoltre, alcuni attacchi, anche se sono stati scoperti in passato, non significa che sono del tutto risolti. Ci possono essere ancora dei sistemi informatici vulnerabili perché non sostituiti/non aggiornati: tanto per fare un esempio la versione WEP di Wi-Fi è facilmente hackerabile ma si trovano ancora in giro delle Wi-Fi con WEP perché per gli utenti "funziona", per loro non ha senso cambiare.

Inoltre l'introduzione dell'IOT ha reso gli attacchi ancora più frequenti. I dispositivi IOT sono particolarmente vulnerabili agli attacchi di cui parleremo in quanto utilizzano protocolli proprietari, magari senza tenere conto degli errori dei protocolli passati che hanno ricevuto dei breach. Inoltre, non hanno tipicamente dentro dei sistemi operativi veri e propri, quindi non sono protetti perché non hanno le versioni più moderne dei sistemi operativi che conosciamo.

## Classi di attacchi ad una rete

Il **Packet sniffing** è un'operazione essenzialmente passiva che consiste nell'ascoltare ed intercettare i pacchetti che girano nella rete, in modo del tutto passivo, quindi senza avere nessuna azione attiva sulla rete.

Per **spoofing**, invece, si intende l'attività per cui si fabbricano dei pacchetti per poi iniettarli dentro la rete ed impersonificarsi qualcuno. Questa è un'operazione invece attiva perché si creano dei pacchetti sulla macchina locale per poi spedirli fisicamente dentro la rete. Ci sono delle librerie che facilitano enormemente questo compito. Quella standard è **Libnet** che rende molto semplice l'operazione di Spoofing. Questa possibilità di usare queste librerie, alla fine, è alla base di tutti i problemi che abbiamo oggi: una persona con un minimo di competenze può fabbricare un pacchetto e spedirlo gratis dentro la rete, senza alcun limite. Se ci si pensa non c'è limite alle cose che si possono mandare dentro la rete e anche al rate con cui si possono spedire.

Il "**packet hijacking**" (dirottamento di pacchetti) invece è una pratica più complessa che combina elementi dello "sniffing" e dello "spoofing". In generale, si tratta di catturare una trasmissione in corso (sniffing) al fine di creare pacchetti falsi con cui inserirsi nella conversazione (spoofing). Questi pacchetti vengono inviati in modo da mescolarsi con quelli della conversazione legittima, ad esempio attraverso un attacco di tipo "man in the middle", alterandone così il comportamento.

Il **flooding** è la pratica che inonda la rete di pacchetti Spoofed in quantità enorme. È possibile farlo non solo da una singola macchina, ma da tante macchine, ad esempio, da una botnet, lanciando quello che si chiama un distributed denial of service (DDOS).

A che conseguenze si va incontro se si effettuano una di queste cose? Non c'è una risposta a questa domanda, la distinzione è molto sottile e non facile da fare, ad esempio, non c'è male nel fare packet sniffing e di sentire i pacchetti che stanno girando sulla rete. Fabbricare un pacchetto arbitrario e spedirlo dentro la rete è già una cosa che può avere delle finalità un po maliziose, però di per sé, non c'è nulla di male: usare una libreria fabbricare un pacchetto e mandarlo in rete, non è illecito. Diventa

illecito quando c'è un chiaro l'intento di nuocere, cioè di danneggiare delle cose, di impadronirsi di informazioni che normalmente non è possibile avere.

Fare denial of service chiaramente non si può e se lo si fa su vasta scala, con l'obiettivo di rendere non più fruibile un servizio chiaramente la cosa è illegittima.

## Sniffing

Perché può essere utile andare a sniffare il traffico su una rete? Ci sono alcuni protocolli classici che mandano informazioni in chiaro, quindi non usano nessuna cifratura dei dati, ad esempio: FTP, POP, HTTP, IMAP. Sicuramente in passato era piuttosto frequente vedere circolare in chiaro username, password, altre informazioni sensibili... e facendo un banale sniffing si andavano a reperire facilmente. Adesso non è più tanto così. Molti protocolli ormai non mandano più informazioni in chiaro.

Anche quando le sessioni sono criptate, però, può essere utile fare sniffing, ad esempio, una delle cose che si possono fare andando a sniffare traffico cifrato è quello che viene chiamato Fingerprinting degli host: quella operazione per il quale, osservando, ad esempio, dei pacchetti inviati da un certo host, è possibile capire come sono fatti, come reagiscono in risposta a degli eventi, capire qual è il sistema operativo che gira su quel host e magari anche la versione del sistema operativo.

## Sniffing tools

Esistono dei tool standard per fare sniffing e che si dividono grosso modo in due famiglie: quelli a riga di comando (headless) e quelli con un'interfaccia grafica.

**Tcpdump** è headless, e non fa solo sniffing di traffico, ma permette in generale di raccogliere qualunque pacchetto che si vuole, andando ad installare un opportuno filtro. Tcpdump si basa su una libreria chiamata **libpcap** che fa il mestiere base della cattura dei pacchetti: a livello kernel (tramite socket RAW) fa una copia di tutti i pacchetti in transito e una di queste copie la manda a Tcpdump che poi la analizza e la salva.

**Wireshark** è UI based, anche lui basato su **libpcap** e permette di fare operazioni di scripting, ma in modo del tutto grafico. Mette a disposizione un'interfaccia grafica molto potente, che permette di vedere dentro tutti i pacchetti. Essendo un'interfaccia grafica ha i suoi limiti.

## Sniffing on local area network

Cominciando dalle reti locali, quindi, che attacchi si possono effettuare, assumendo di essere all'interno di una rete locale switch based?

Una volta (con le reti a BUS e HUB based) era possibile utilizzare la modalità promiscua delle schede di rete, che permetteva di sentire tutti i pacchetti in transito sulla rete locale, quindi non solo quelli diretti direttamente alla propria scheda (MAC address), ma sentire tutti gli altri, anche quelli che non avevano come indirizzo di destinazione quello della propria scheda di rete.

Il problema è che la modalità attuale con cui si costruiscono le reti locali (usando gli switch) è molto difficile convincere lo switch a farsi mandare un determinato pacchetto, lo switch separa i domini di collisione. Quindi come si può fare sniffing su rete ethernet switch based?

In una rete switch ethernet, lo sniffing dei pacchetti non è affatto gratis, bisogna trovare degli accorgimenti per far sì che lo switch inoltri comunque del traffico che non appartiene a noi. Storicamente esistono una serie di tecniche, la prima è quella di inondare lo switch di pacchetti, con

vari indirizzi Mac diversi finché lo switch non è più in grado di mantenere informazioni in cache: la sua tabella interna va in overflow. Quando uno switch consuma la memoria per la cache table, si converte in un hub, quindi prende un pacchetto e lo replica su tutte le porte.

Un'altra cosa che è possibile fare è quello di ingannare lo switch facendogli credere di essere una macchina con un certo indirizzo Mac, quindi l'attaccante riconfigura il suo host in modo da avere l'indirizzo Mac della vittima. È un'operazione non particolarmente complessa, in quanto, è possibile riconfigurare l'indirizzo Mac via software a piacimento. Ovviamente clonando l'indirizzo Mac della vittima, nella rete ci saranno due macchine con lo stesso indirizzo, mandando così la rete in confusione. In particolare, lo switch sarà estremamente confuso: ogni tanto sentirà il Mac provenire da una certa porta e altre volte da un'altra porta. Quindi bisogna anche ricordarsi che in uno scenario di questo tipo ci sarà un sacco di rumore o di confusione dovuta al fatto che ci sono due macchine che hanno lo stesso Mac.

## ARP Spoofing

Un attacco un po più sofisticato che si può fare è quello di sfruttare una vulnerabilità del protocollo ARP e cioè il fatto di essere stateless (ARP Positioning attack o ARP Spoofing). Non avendo Stato, quello che è possibile fare, è mandare un arp reply senza che questa reply sia stata scatenata da una request (proprio perché non ha stato). In questo modo si "avvelena" la cache degli host, alterando il mapping degli indirizzi IP – MAC. Questo può portare fino ad una situazione in cui l'attaccante diventa man in the middle tra due host.

Un problema che nasce è che le reply legittime potrebbero restaurare gli indirizzi corretti, (quelli con la corretta associazione), quindi a quel punto diventa una guerra (race condition) tra due pacchetti che cercano di insegnare mapping diversi. Il tool che effettua il poisoning quindi deve, in modo continuo nel tempo, andare ad avvelenare la cache altrimenti potrebbero essere restaurate allo stato corretto, quindi c'è questa race tra avvelenamento e ricostituzione dello stato corretto.

Tutto questo è abbastanza automatizzato, ci sono dei tool come **Dsniff**. Alcuni sono anche così sofisticati da permettere di realizzare un man in the middle anche su sessioni SSH e HTTPS.

Allora che cosa è possibile per difendersi da un ARP Poisoning? Il problema è serio: non c'è modo di risolvere nativamente il problema, si possono attuare però delle contromisure. Una possibilità è quella di disabilitare l'ARP e di creare mapping statici tra indirizzi IP e indirizzi Mac, per lo meno per gli indirizzi IP più critici, più sensibili, tipo il server DNS, gateway. Però l'idea di fare a meno di ARP in scenari con tanti nodi è infattibile perché porta ad un costo di gestione troppo elevato.

È possibile utilizzare la tecnica dell'**ARP inspection**. Questa verifica l'integrità delle richieste ARP e delle relative risposte all'interno della rete. Durante il processo di ARP inspection, il dispositivo di rete, come uno switch, tiene traccia delle richieste ARP inviate dai dispositivi connessi e verifica se le risposte ARP corrispondono alle richieste originali. Se una risposta ARP non corrisponde a una richiesta precedente, viene considerata sospetta e può essere bloccata o gestita in modo appropriato. In sostanza, l'ARP inspection permette ai dispositivi di rete di controllare e validare il traffico ARP nella rete, identificando e bloccando le risposte ARP non valide o sospette (rende ARP stateful in un certo senso).

Un livello ancora più sofisticato è quello di utilizzare dei tool come ARPWatch. Sono Tool che si mettono in ascolto appositamente del traffico ARP per andare a scovare le operazioni maliziose, quindi traffico ARP anomalo e che quindi dall'identificazione di un attacco in corso. Altre cose che si possono fare, ad esempio, se uno adotta questo approccio di fare il monitor del traffico ARP, è tener traccia delle associazioni IP – MAC e, quando avvengono cambiamenti sospetti, scatenare degli

allarmi in quanto, è si normale che questi mapping possano cambiare, però se lo fanno con una frequenza e con un ritmo anomalo, questi strumenti se ne accorgono abbastanza facilmente.

## Detecting sniffers

Una cosa interessante è invece l'uso di strategie per individuare eventualmente qualcuno che sta facendo sniffing dentro una rete.

Se si ha in mano il terminale di cui si ha il sospetto che stia facendo uno sniffing, è banale: basterebbe eseguire il comando ifconfig che tra le informazioni di una certa scheda, mostra anche se è in modalità promiscua. Però ovviamente questa cosa è facilmente mascherabile: a livello kernel si può impedire che vengano riportate le informazioni che una scheda è in modalità promiscua (questo però in uno scenario HUB/BUS).

Gli attacchi di sniffing hanno certe caratteristiche di rumorosità che rendono possibile, attraverso tool come ARPwatch, trovarli. Infatti, esistono anche Tool che individuano la presenza di uno sniffer sfruttando ad esempio la latenza. L'idea è questa: se una scheda è in modalità promiscua, essendo che lei riceve tutti i pacchetti, sarà più lenta processarli in quanto tutti i pacchetti vengono presi dalla scheda mandati al sistema operativo per essere analizzati e questo introduce una maggiore latenza, allora, si genera un sacco di traffico in rete e contemporaneamente a questo grosso invio di traffico in rete, banalmente si effettua un ping verso l'host di cui si ha il sospetto che sia uno sniffer vedendo i tempi di risposta al ping. Quindi confrontando prima e dopo: quanto era latenza prima dell'inoltro di questo scatenamento di traffico e durante lo scatenamento del traffico, confrontando la latenza si riesce a notare se una scheda è in modalità promiscua oppure no.

Poi nel caso particolare di Unix, c'è un modo velocissimo con cui si vede se una NIC è in modalità promiscua: si manda un pacchetto che ha un indirizzo MAC sbagliato, ma un indirizzo IP corretto. Il sistema operativo UNIX però, anche se il MAC è errato, processa comunque il pacchetto, quindi banalmente se si manda un ICMP request verso un host UNIX e questo risponde con ICMP reply, si capisce subito che quell'host sta sniffando la rete.

L'approccio più incisivo verso questi tipi di attacchi sulle reti locali è quello di fare un controllo stretto degli accessi alla rete: tutte le cose di cui si è parlato richiedevano che l'hacker avesse un accesso fisico ad una porta ethernet interna alla rete locale (che è la situazione più vulnerabile che esista: permettere a chiunque di girare con un portatile attaccarsi a ethernet ed avere accesso alla rete). Ormai quasi più nessuno lo fa, proprio perché la cosa è molto pericolosa. Le cose che si fanno sono di due tipi: Controllo di accesso, quindi, avere una lista di indirizzi Mac che sono legittimi e solo quelli possono scambiare traffico dentro la rete. Oppure, ancora più sofisticato, con uno standard 802.1X: usare una tecnica di autenticazione con un server e usando i protocolli standard EAP, EAPOL. In sostanza, utilizzando questi protocolli, all'inizio si ha una limitatissima capacità di scambio di traffico e solo quando si presentano dei certificati opportuni all'Authentication server, la rete abilità a mandare tutto il traffico che si vuole.

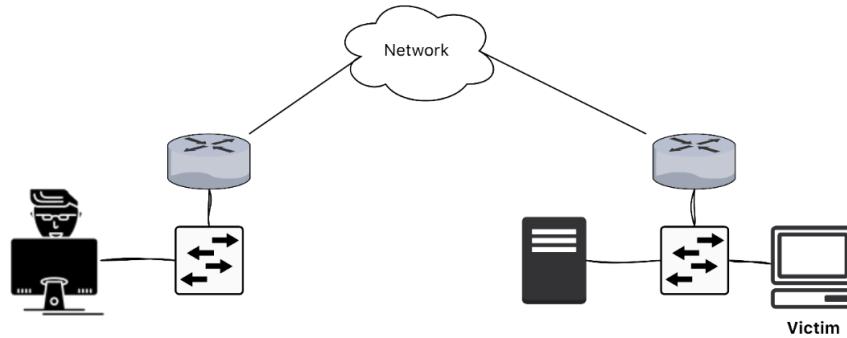
## Spoofing on IP

Ci muoviamo lentamente verso i protocolli di livello superiore. Andiamo a vedere attacchi che si fanno a livello 3.

Niente di più facile che forgiare pacchetti IP con indirizzi fasulli: con l'ip spoofing, un host si può impersonare come un'altro host semplicemente mettendo come indirizzo sorgente quello dell'host che vuole impersonare. Ovviamente solamente così non è possibile fare molto: IP nativamente non

permette di instaurare una comunicazione con nessuno: è poi necessario a livello più alto un qualche altro protocollo.

Ma allora, perché mai ci si dovrebbe impersonare qualcun altro? Perché è possibile fare dei danni seri se ci si impersonifica in alcuni HOST ad esempio, il DNS. Se ci si impersonifica come un DNS server, è possibile dare delle informazioni di routing arbitrarie e quindi gli host vittime che volevano collegarsi con un certo indirizzo, vanno poi a risolvere quell'indirizzo con un IP sbagliato.



Una cosa che succede quando si fa IP spoofing, è lo scenario seguente: l'hacker fabbrica un pacchetto con indirizzo sorgente fasullo con lo scopo di far entrare questo pacchetto dentro una rete vittima dove c'è una relazione di fiducia tra un host e un server. Quindi tutto il traffico che gli arriva da quel server la considera legittima. A questo punto, l'hacker inietta un pacchetto in cui inserisce come indirizzo IP sorgente quello del server, di cui l'host si fida e questo pacchetto potrebbe poi fare qualcosa, nel caso più peggiore, eseguire dei comandi.

Le contromisure che si adottano è quello che viene chiamato **ingress filtering**: tipicamente il router che fa da gateway per la vittima può difendere dall'ingresso di pacchetti dall'esterno di questo tipo, vedendo arrivare da fuori un pacchetto che ha un indirizzo IP sorgente, un indirizzo interno.

Al contrario, esiste anche **l'egress filtering**: cioè il filtraggio di quel traffico che non ha senso che esca da un certo punto verso fuori. Questo è un meccanismo adottato però lato mittente (da dove l'hacker agisce). Ovvio che se il cattivo ha il controllo del router...

Un'altra cosa che rende questo attacco, blind, poco utile, è il fatto magari l'hacker è riuscito ad inserire un pacchetto dall'esterno verso la rete vittima ma poi un'eventuale risposta non arriverà mai.

## IP Fragmentation and Ping of Death

Una cosa interessante a livello 3 è quello che viene chiamato Ping of death.

L'idea è questa: con un singolo pacchetto malformato, fabbricato ad arte, che arriva su una macchina vittima, potrebbe mandarlo in kernel panic, provocando un arresto anomalo del terminale.

Il primo caso di questi di tipo Ping of Death è attorno agli anni 90 e sfruttava la frammentazione IP. Il meccanismo sfrutta questa vulnerabilità della frammentazione IP: se si impone opportunamente l'offset e la dimensione dell'ultimo segmento, si effettua un buffer overflow sull'host vittima. Dato che un pacchetto IP ha una dimensione massima di 64k, l'host vittima aveva inizialmente allocato un buffer di ricezione pari alla dimensione massima che può mai avere un pacchetto IP, quindi 65.536 byte. Successivamente, l'hacker manda un pacchetto costruito ad arte, con l'ultimo frammento avente offset 65.121, con lunghezza 398. Allora  $65.120 + 39 + 20$  byte di header viene 65.538 che supera la dimensione massima allocata dell'host vittima: Ping of death.

Nel 2013 è accaduto di nuovo: un nuovo caso di ping of death ma su Ipv6. Ma come è possibile che sia successo la stessa cosa con IPv6 se con questo protocollo è stata abolita la frammentazione? Il problema è che questa versione di Ping of death sfruttava un pacchetto ICMPv6 malformato (e non la frammentazione). È stato scoperto anche nel 2020 nello stack sui sistemi Windows, di nuovo legata ad ICMPv6, in questo caso si poteva addirittura eseguire un codice da remoto sulla sul client della vittima.

### **Fragmentation to avoid Firewall**

Si può sfruttare la frammentazione anche per altri scopi, ad esempio per evitare il filtraggio operato da firewall e sistemi di intrusion detection. Alcuni firewall prendono una decisione sui pacchetti che dicevano dall'esterno senza ricostruirli tutti quanti, guardando solo il primo di questi frammenti. Quindi, in alcuni casi c'è la possibilità di bypassare i filtri proprio utilizzando la frammentazione, quindi mandando un frammento che contiene magari pochissima informazione, ad esempio, che contiene solo le porte. Il firewall, basandosi solo su questo frammento decidono di ammetterlo e di farlo passare. E dopodiché fanno passare tutti gli altri in quanto appartenenti allo stesso Datagram.

Quindi la frammentazione è vista come il più possibile da evitare, in quanto complessa, fragile e richiede dello sforzo per ricostruire i frammenti: allocare memoria, sforzo computazionale per ricostruire il datagram. Infatti nella versione più recente di IP è stata proprio abolita, non è più consentita la frammentazione.

## **ICMP Attacks**

l'Internet Control message Protocol ICMP, è quel protocollo che scambia messaggi di controllo, messaggi di errore legati alla consegna. È un protocollo che concettualmente viene trasportato dentro IP.

Esistono diversi tipi di messaggi, i più significativi per ora sono ECHO request ed ECHO reply (che è alla base del comando PING con cui si va a testare la connettività).

### **Ping scan**

Un uso che è possibile fare di ICMP è detto ping scan, si scannerizza tutta una sottorete visualizzando tutti i possibili indirizzi IP appartenenti ad uno sottorete, andando a cercare quali host sono attivi.

### **Smurf Attack**

Un'altro attacco molto famoso basato su ICMP è lo Smurf Attack.

L'idea è questa: un attaccante manda un pacchetto di Echo Request con un indirizzo IP Spoofed, cioè mettendo come indirizzo sorgente l'indirizzo un host vittima. Come indirizzo di destinazione della eco request invece inserisco intere sottoreti utilizzando destinazioni broadcast. Quando tutti gli host raggiunti dal broadcast effettueranno la ECHO Reply andranno a sommersere la vittima ottenendo un processo di amplificazione. Questo processo (di amplificazione) è pattern classico che si sfrutta negli attacchi, è molto ricorrente: con un piccolo sforzo si ottiene un risultato amplificato molto superiore in termini di volume di traffico di byte generati nella rete rispetto a quelli che si mandano inizialmente.

Questo attacco non viene effettuato per danneggiare un host, ma per negargli di operare nella rete, infatti l'host vittima negerà questi pacchetti ICMP, ma verrà indondato di un traffico talmente elevato di pacchetti ICMP che portano a saturare la larghezza di banda della sua connessione Internet o il

potere di elaborazione del suo sistema. Di conseguenza, la vittima diventa inaccessibile agli utenti legittimi.

Quando si è visto che si poteva fare questa cosa, hanno disabilitato tutti il broadcast su IP. Quindi tutti i router e tutti i firewall filtrano i pacchetti IP broadcast (era l'unico rimedio allo smurf attack).

### **ICMP Redirect**

Un altro esempio di utilizzo maligno dei pacchetti ICMP è quello del redirect. L'incidenza di un redirect normale si sviluppa come segue: all'interno di una rete sono presenti due gateway, un host invia un pacchetto al primo gateway. Dopo aver analizzato il pacchetto, il gateway però lo restituisce all'host con un ICMP redirect, segnalando che la destinazione desiderata non è nota e suggerendo di passare attraverso un altro gateway. A questo punto, l'host riceve il redirect e aggiorna la propria tabella di instradamento.

L'idea dell'attacco invece è la seguente: si fabbrica un pacchetto di ICMP redirect, che sembra provenire dal default gateway, inserendo come indirizzo IP sorgente quello del gateway (Spoofed). Questo viene interpretato come un legittimo redirect con l'effetto di dirottare del traffico dell'host (quindi del traffico che inizialmente andava al gateway legittimo viene viene dirottato da qualche altra parte) ad un host malevolo che ad esempio scarta tutti i pacchetti, effettuando Denial of service.

Ci sono ovviamente dei controlli nativi che sono stati implementati all'inizio per far sì che non si facesse un uso insensato di questa opzione e sono delle semplici controlli di correttezza, il problema è che non hanno messo abbastanza controlli nel protocollo stesso da non rendere possibile degli usi totalmente impropri di ICMP redirect e che quindi hanno portato all'ideazione di un attacco di questo tipo. Per mitigarlo però è sufficiente disattivare il redirect.

### **ICMP Destination unreachable**

Un'altro uso malizioso di ICMP riguarda il destination unreachable.

Lo scenario è il seguente: un attaccante forgià un pacchetto destination reachable, mandandolo verso una certa vittima. In questo caso l'host vittima che aveva cercato di stabilire una connessione (che era assolutamente raggiungibile) verso qualcuno (un server remoto ad esempio) non riuscirà a connettersi in quanto riceve l'ICMP destination unreachable (facendo ritornare un errore in apertura del socket al browser o all'applicativo). In questo modo si taglano fuori dei nodi dalla rete. Può ovviamente crearsi una Race condition tra risposte di server raggiungibile e destination unreachable. Quindi questo attacco è più fastidioso che dannoso, di per se non danneggia niente.

## **Attacks over UDP**

Passiamo al livello trasporto.

UDP non fa un granchè rispetto ad IP, aggiunge le porte per identificare i servizi, ma non fa niente di più, si appoggia al best effort di IP. In particolare non stabilisce connessioni, non è orientato alla connessione e così facendo si presta a vulnerabilità banali basate appunto sul forgiare dei pacchetti UDP fasulli. Ed essendo che UDP viene usato anche per servizi come DNS, causa dei problemi non indifferenti.

### **UDP Port Scanning**

Il Port scanning è un modo per curiosare in modo più o meno legittimo, quale porte sono attive su un certo host. L'idea è semplice, io mando dei pacchetti UDP senza contenuto ad ogni possibile porta

delle 65.000 varie possibili. Successivamente, se torna indietro un messaggio di ICMP port unreachable, la porta è chiusa. Se invece non torna indietro, la porta è aperta. Quindi è possibile fare uno scan delle porte attive su un host.

È un modo non particolarmente furbo di fare scanning, esistono tecniche molto più avanzate di questa. Inoltre, alcune implementazioni di alcuni sistemi operativi, limitano il numero di messaggi di errore generati ad esempio UNIX ha limite a 80 messaggi ogni quattro secondi, quindi se si vuole fare lo scanning per le 65.000 porte ci si impiega un bel po.

Il tool per eccellenza per fare port scanning è **nmap**, un tool che fa davvero tutti i tipi possibili di Port scanning, con infinite opzioni, essenzialmente non ha rivali.

### The fraggle attack

C'è una versione dello Smurf Attack che si chiama fraggle attack che usa UDP anziché ICMP, però la logica totalmente identica: quindi si usa il broadcast su IP per ottenere un effetto di amplificazione. Anche questo attacco è stato reso non più fattibile quando si è giustamente scelto di abolire il broadcast su IP.

## Attacks over TCP

Gli attacchi che si possono fare su TCP sono più complicati per il fatto che TCP è orientato alla connessione (3-way handshake, numeri di sequenza, acks).

Il primo problema che sorge quando si vuole incasinare una connessione TCP è che l'attaccante deve possedere il numero di sequenza iniziale che il server invia al client vittima. Quindi quando si esegue il 3-way handshake, il server sceglie tipicamente in modo casuale un sequence number iniziale che l'attaccante non conosce, e di conseguenza non può intrufolarsi dentro la connessione. Questo rende gli attacchi TCP molto più difficili se non impossibili.

Inizialmente, quando si era molto ingenui, ad esempio sui sistemi unix di quando è nato Internet, avevano banalmente scelto di usare un sequence number iniziale che per ogni connessione veniva incrementato ogni mezzo secondo di 64.000. Inizialmente veniva fatta questa ingenuità di aumentare sequenzialmente il sequence number, ottenendo però una cosa molto predicable, perché se lo si fa crescere nel tempo in questo modo lineare, deterministico, è molto facile azzeccare quale sarà il prossimo sequence number che un server andrà a scegliere.

### TCP Port Scanning

Un'equivalente del Port Scanning con UDP si può fare in modo equivalente anche per le porte TCP. Ci sono un sacco di servizi che utilizzano porte di default standard, quindi una cosa che è possibile fare per scoprire tutti i servizi aperti su un host è quella di aprire delle connessioni normali verso tutte le porte che si vuole esplorare e se si riesce a concludere il 3-way handshake, evidentemente il servizio è disponibile. Se la porta non è attiva verrà ritornato un ICMP Port unreachable.

Il problema di questo metodo è che mi espongo molto: i server tendono a fare logging di tutte le connessioni portate a buon fine, quindi se l'attaccante compare nei log del server a quel punto è davvero molto esposto.

Uno scanning più sofisticato e "migliore" perché non lascia tracce nei log di sistema del server, è quello basato sul fatto di non completare l'handshake: si inoltra il pacchetto di SYN, successivamente, se il server inoltra il SYN-ACK, non si conclude con un ACK ma si inoltra un RST (reset) con cui si abortisce l'handshake. In questo modo si è scoperto che la porta è attiva, però non completando l'handshake non resta traccia di questo tentativo di apertura. Molto più efficace della

prima versione di scanning, ma in questo caso è necessario avere dei privilegi di root, in quanto bisogna bloccare il terzo pacchetto dell'handshake che necessita di effettuare delle operazioni più a basso livello che necessita di root privilege.

Ogni server poi è configurato in modo diverso, ovviamente se il sistema è configurato per loggare anche connessioni resettate, si rimane loggati anche lì.

Un'altra cosa simile la si può fare con il flag FIN. L'attaccante forgia un pacchetto con FIN settato. Il problema che ci si potrebbe porre è il seguente: il pacchetto FIN è utilizzato per chiudere la connessione e basta, a cosa serve mandare un pacchetto FIN senza aver aperto prima una connessione? Si è scoperto che se si forgia un pacchetto FIN fasullo e lo si manda come primissimo pacchetto verso un host, dipendentemente dall'implementazione di TCP, si hanno risposte diverse. La maggior parte delle implementazioni hanno questa reazione: se la porta è chiusa, ritorna un pacchetto TCP con flag RST, altrimenti, se la porta è aperta non ritorna indietro niente. Si hanno quindi due comportamenti diversi a seconda dello scenario, capendo così se la porta è aperta o chiusa.

Alcuni effettuano lo scanning fabbricando dei pacchetti in cui accendo tutti i Flag TCP, quello che viene chiamato **Xmas packet**, perché come un albero di Natale si accendo tutti i Flag possibili che quindi può innescare delle risposte che con altri pacchetti non venivano innescate, però alla fine il concetto è lo stesso: scoprire se una porta è aperta o chiusa.

### **IDLE scanning**

L'idle scanning è un attacco che, di nuovo, permette di fare lo scanning di un host al fine di vedere quali sono le porte aperte. Questo metodo è talmente subdolo che permette di fare "il lavoro sporco" a qualcun altro. L'attaccante a questo punto resta davvero invisibile. L'attacco funziona in questo modo: c'è un host, un attaccante e quello che viene identificato come Relay (che è il terminale che farà il lavoro sporco per l'attaccante). La fase preliminare all'attacco, cerco di capire qual è l'identificativo IP (nell'header Ipv4 c'è un campo identification) che il Relay sta usando attualmente, di solito viene incrementato di 1 ad ogni pacchetto mandato. Per fare questo, apro una connessione verso il Relay utilizzando come primo pacchetto un SYN-ACK (che poi viene subito abortita) ed utilizzando il pacchetto di RST di risposta, si viene a conoscenza dell'identificativo IP. Dopodiché, si lancia l'attacco vero e proprio che consiste nel forgiare un pacchetto con indirizzo IP Spoofed spacciandosi per il Relay con flag SYN sulla porta, ad esempio 80, verso la vittima. Se la vittima risponde normalmente ad un SYN, con un SYN-ACK, il Relay (che non è stato lui ad aprire la connessione), risponde con un pacchetto RST, in quanto non è stato lui ad aprire quella connessione. Ed è qui che si capisce se la porta 80 della vittima è aperta: nel mandare l'RST, il relay, ha confermato che il server ha risposto con un SYN-ACK, e che quindi era pronto ad aprire una connessione su quella porta. Per capire se il Relay ha inoltrato l'RST, basta rieseguire la procedura iniziale inoltrando un SYN-ACK al Relay e controllando che abbia effettivamente incrementato il suo ID.

L'attaccante riesce a fare un idle scanning, e cioè, a dedurre che una porta è aperta schiavizzando un'host innocente per fare il lavoro sporco. Mantenendo la sua identità, il suo indirizzo IP nascosti.

La debolezza di questo attacco è che lo posso fare su un Relay che era in un certo momento quieto, che non stava trasmettendo altri pacchetti di altre connessioni, altrimenti viene tutto distorto. L'altra cosa che evidentemente manda in crisi l'attacco è se il Relay sceglie un ID per ogni pacchetto IP casuale anziché progressivo.

## OS Fingerprinting

Un'altra cosa utile per preparare un attacco, oltre a scoprire porte sono aperte, è di scoprire, di un certo host, che sistema operativo sta utilizzando e che versione del sistema operativo.

Ci sono tutta una serie di strumenti che permettono di fare OS fingerprinting. L'idea è quella che tanti protocolli ad esempio TCP, che è molto complicato, ha delle implementazioni diverse a seconda dell'OS che lo usa. Allora, fabbricando dei pacchetti strani ad esempio dei FIN, oppure inserendo opportuni Flag, e vedendo come l'host risponde, riesce a risalire a quale sistema operativo li ha generati.

Ci sono varie tecniche che si possono usare, come lo stesso **Ethercap** che addirittura a volte, in modo del tutto passivo cioè, semplicemente osservando passivamente il traffico che sta girando in una rete, riesce ad indentificare che sistemi operativi sono in esecuzione su un certo Host. È anche però possibile difendersi cambiando a livello software che tipo di OS viene visto dall'esterno.

In generale è importante l'OS fingerprinting, in quanto se si scopre che un host utilizza un certo sistema operativo, magari con una versione buggata ed ha una certa porta aperta, poi si possono lanciare attacchi mirato e fare molti danni.

## Mitnick Attack

Come si fa a stabilire una connessione con un host vittima, impersonando un altro host? Quindi l'obiettivo del TCP Hijacking è quello di stabilire una connessione con qualcuno che normalmente non si avrebbe il diritto di stabilire, ma che impersonificandosi con qualcun altro è invece possibile portare a termine e fare dei danni.

Uno dei primi che riuscì a stabilire un attacco di questo tipo, fu **Kevin Mitnik**. L'attacco prende proprio il suo nome. Nel **Mitkin attack**, ci sono tre attori: l'attaccante, e le due vittime sono un Host ed un server. Si suppone inoltre che esista una relazione di fiducia tra l'host verso il server.

C'è una fase preliminare in cui l'attaccante esegue Denial of service verso il server, in modo da tagliarlo fuori, in modo tale che da un certo momento in poi non vada a generare pacchetti. A questo punto l'attaccante manda un pacchetto di SYN verso l'host facendo finta di essere server. Il problema che sorge è che io lo posso tranquillamente forgiare questo pacchetto di SYN, però quando l'host risponde con il SYN-ACK, quel pacchetto verrà inoltrato al server, con un suo indirizzo di sequenza randomico, che è un'informazione cruciale se si vuole completare l'handshake. Kevin Mitnik, indovino il sequence number restituendo il pacchetto di ACK correttamente, aprendo una connessione verso l'host spacciandosi per il server. A quel punto è mandare un'ulteriore pacchetto, che nel caso dell'attacco storico di Kevin mitnick, apriva una backdoor, dargli la possibilità poi di accedervi dall'esterno.

Come ha fatto Kevin Mitnik ad indovinare il sequence number? Prima di questo attacco, i sequence number venivano generati in modo predicable, quindi l'attacco era possibile. Da quel momento in poi, le varie stack protocolari hanno capito che bisognava renderlo il più random possibile, e tutti hanno iniziato a patchare TCP indorducendo randomizzazione.

## TCP Hijacking

Che cosa succede quando si prova ad inserirsi dentro una connessione TCP facendo hijacking? Succede una cosa molto curiosa: se si inserisce un pacchetto fittizio all'interno di una connessione (ad handshake già avvenuto) in cui questo pacchetto appare come parte di trasferimento dati della trasmissione, succede una cosa piuttosto bizzarra e cioè che si desincronizza il meccanismo

Sequence number – Acknowledge portando ad una tempesta di acknowledgement chiamata Ack Storm.

La conversazione impazzisce e si generano messaggi che si contraddicono l'uno con l'altro, finché uno dei due viene perso per congestione della rete. A quel punto appena uno viene perso, interrompe la sequenza anomala, ma il pacchetto iniettato dall'attaccante alla fine viene comunque mantenuto.

ci sono delle varianti dell'attacco in cui si evita di generare l'Ack Storm perché si è in grado di predire esattamente la dimensione che avrà il prossimo pacchetto legittimo mandato da uno dei due host. Se io sono in grado di predire la dimensione, posso mandarlo per primo, fasullo, e in questo modo evito l'ACK Storm.

L'obiettivo non è quello di interrompere la connessione, ma anzi, mantenerla attiva ed iniettare dei dati fabbricati che quando la connessione si risincronizza, vengono riconosciuti come effettivamente legittimi e parte di quella connessione.

### **SYN Flooding Attack**

L'idea del SYN Flooding è quella di fare Denial of Service verso un host bombardandolo di pacchetti SYN che all'attaccante costa molto poco produrre e che sul server invece fanno molto male perché ogni pacchetto SYN ricevuto dal server, predisponde ad aprire una connessione TCP allocando tutta una serie di strutture nel sistema, magari facendo partire anche un thread nuovo del processo per gestire quella connessione.

Il pattern fondamentale di questi attacchi è il fatto che la simmetria nel consumo di risorse, tra chi attacca e chi riceve l'attacco. Cioè per chi lancia l'attacco non costa niente in termini di traffico generato e di consumo di risorse mentre sulla vittima è tutto l'opposto. La vittima alloca un sacco di stato per la connessione e non avendo infinite risorse, dopo un po le satura.

Nel SYN Flooding, l'attaccante manda il pacchetto SYN alla vittima che risponde con SYN-ACK, a quel punto l'attaccante non fa niente, non porta a termine la connessione lasciando la connessione mezza aperta.

Alcuni rimedi per il SYN Flooding:

- È possibile filtrare, cioè avere un rate massimo di SYN che accetto e banalmente allocare più risorse, quindi aumentare la coda delle connessioni mezze aperte.
- Ridurre il time out: non tenere le connessioni mezze aperte all'infinito, ma dopo un po che non si sono completate con successo, liberare le risorse che il sistema aveva allocato.

Un meccanismo molto astuto per mitigare davvero bene questo problema, si basa su **SYN Cookies**. L'idea è che si cerca di evitare il problema all'origine: non dedicare subito risorse del sistema per una connessione che potrebbe non completarsi mai. Quindi quando arriva un SYN, lo si gestisce, ma senza allocare dentro il sistema dello stato, quindi, in particolare, senza allocare tutte quelle strutture dati costose.

Il meccanismo è quello di memorizzare dentro una tabella hash un oggetto ottenuto combinando un certo numero di informazioni, creando un digest che identifica una specifica connessione, ad esempio  $H(\text{srcPORT}, \text{destPORT}, \text{srcIP}, \text{destIP})$ , salvando solo il risultato di questo hash e non facendo nient'altro (in particolare, non allocando stato) e mandando indietro il SYN-ACK. Se quella connessione è legittima, quindi è davvero benevola e vuole completare la connessione ad un certo punto si riceverà l'ACK con cui dimostra di essere una vera connessione senza indirizzo IP Spoofed ma un vero HOST che desidera parlare in modo benevolo. A quel punto si può iniziare ad allocare

stato. Nel dettaglio, quando arriva l'ACK finale dell'handshake, si fa un lookup all'interno della hashtable. Se lo si trova, vuol dire che quello è davvero il terzo pacchetto di Handshake e solo allora si allocano tutte le risorse preziose per quella connessione.

Quindi in questa tecnica si introduce un ritardo volontario nell'allocazione effettiva delle risorse in modo da preservare le risorse, si risolvono un sacco di problemi di sicurezza in modo molto elegante.

## State Attack

La cosa che si impara da questi attacchi (che sono anche quelli che portano più danno) è che sono basati sul fatto che alcuni servizi allocano Stato nei nodi, ad esempio, la connessione TCP. Questo significa che, se l'allocazione dello Stato, in termini di risorse consumate, è asimmetrica tra l'attaccante e la vittima, tutte le volte che questo succede si ha un serio problema di vulnerabilità.

La tendenza attuale per risolvere questi problemi fondamentali è fare le cose il più possibili stateless.

## Attacks over HTTP

Un attacco **Slow HTTP POST** è una forma di attacco DDoS (Distributed Denial of Service) in cui un aggressore sfrutta una vulnerabilità nel protocollo HTTP per sovraccaricare un server e rendere inaccessibili i servizi ospitati su di esso. A differenza di altri tipi di attacchi DDoS che mirano a saturare la larghezza di banda o le risorse di elaborazione del server, un attacco Slow HTTP POST sfrutta una limitazione del protocollo HTTP stesso.

Il funzionamento di un attacco Slow HTTP POST si basa sul fatto che, in una connessione HTTP, il client invia una richiesta al server e il server risponde con una risposta. Durante questo processo, l'attaccante invia una richiesta POST al server, ma la invia lentamente, inviando i dati in maniera frammentata o con un ritardo prolungato volontario tra i singoli pacchetti.

Il server, seguendo le specifiche del protocollo HTTP, mantiene aperta la connessione in attesa che l'intero corpo della richiesta POST venga ricevuto. L'attaccante può sfruttare questa caratteristica inviando dati a un ritmo molto lento o inviando pacchetti molto piccoli, in modo che il server mantenga aperta la connessione per un lungo periodo di tempo. Ciò fa sì che il server consumi risorse, come connessioni TCP, thread del server e memoria, mentre aspetta il completamento della richiesta.

L'obiettivo principale di un attacco Slow HTTP POST è esaurire le risorse del server e renderlo inaccessibile a utenti legittimi. Poiché le risorse del server sono limitate, il server sarà in grado di gestire solo un certo numero di connessioni simultanee. Quando tutte le connessioni disponibili sono occupate dall'attacco Slow HTTP POST, i nuovi utenti legittimi saranno respinti e non potranno accedere ai servizi ospitati dal server.

Gli attacchi Slow HTTP POST possono essere particolarmente efficaci perché richiedono solo una piccola quantità di larghezza di banda per avere un impatto significativo sul server. Possono essere difficili da rilevare in quanto non comportano un traffico intenso o insolito, ma piuttosto un rallentamento graduale delle risposte del server.

Per mitigare gli attacchi Slow HTTP POST, i server possono implementare diverse contromisure, come impostare limiti di tempo per le richieste HTTP, utilizzare server proxy per filtrare il traffico dannoso o utilizzare firewall applicativi che riconoscono e bloccano i modelli di traffico tipici di tali attacchi.

## Attacks over Wireless

Diciamo qualcosa sulla sicurezza delle reti wireless e in particolare wifi.

Ci limiteremo a fare gli esempi sulla modalità infrastruttura, quindi quella con Access point, ma è bene ricordare che esiste anche la modalità ad-hoc in wifi che permette ai nodi di comunicare direttamente tra di loro senza che ci sia un Access point connesso al resto di Internet.

Per ricordare come funziona l'associazione, gli Access point sono identificati da un cosiddetto SSID, un identificativo, che è essenzialmente il nome della rete.

Quello che di solito si fa è di privilegiare le reti che hanno un segnale più forte: se la scheda ha in memoria diversi Access point con cui potere effettuare l'accesso, la scheda tipicamente sceglie quella col segnale più forte e lo fa in modo automatico. Questo è all'origine di un possibile problema di sicurezza. Ci sono degli attacchi basati sul fatto che un utente malevolo, si impersonifica in un access point e fa in modo di farsi sentire con la potenza migliore, così che la scheda di un altro host cerca di connettersi a questo Access point di cui l'attaccante ha clonato l'SSID.

Per proteggersi da situazioni di questo tipo alcuni AP, consentono l'accesso solo a una lista di host con un certo MAC address ma di nuovo, questo è un controllo di accesso che ha i suoi limiti, in quanto questa volta basterebbe clonare un MAC di un host a cui è permesso accedere e spacciarsi per quell'Host. Per mitigare questo tipo di attacco, sono stati sviluppati ulteriori protocolli di sicurezza, come Opportunistic Wireless Encryption (OWE) e Wi-Fi Protected Access 3 (WPA3). Questi protocolli introducono meccanismi aggiuntivi, come l'autenticazione mutua tra l'access point e il dispositivo e l'utilizzo di crittografia individuale per ogni sessione, per prevenire l'accesso a un access point clonato.

## Sniffing over Wi-Fi

In generale, le reti Wi-Fi sono particolarmente vulnerabili allo sniffing per l'ovvio motivo che il mezzo è broadcast, quindi nativamente un pacchetto mandato da una stazione raggiunge tutti quelli attorno a lei, in un certo raggio di trasmissione. Questa semplicità di fare sniffing in ambito wireless è ovviamente un grosso problema di sicurezza che le rende molto più vulnerabili ad attacchi di quello che invece era il mondo cablato (con ethernet era necessario avere un accesso fisico alla rete da attaccare, quindi con ethernet non c'è altro modo di attaccare una rete ethernet se non fisicamente entrare nell'area privata dove si trova una porta ethernet). Nel mondo wireless è possibile invece lanciare un attacco anche dal parcheggio (Parking lot Attack).

Quando si vuole sniffare traffico Wi-Fi bisogna fare distinzione tra quello che si può fare nel cosiddetto promiscous mode e quello che si può fare nel monitor mode. C'è da dire che non tutte le schede permettono di mettere l'interfaccia di rete wifi in una di queste modalità. La differenza è che nel promiscuous mode, si vedono solo i pacchetti dati, quindi non si vedono i pacchetti di controllo, che invece sono utili per fare il setup e il mantenimento della connessione, ma solo i pacchetti di tipo dati. Questo limita perché non avendo a disposizione tutti i Frame si ha meno informazione.

Alcune schede si possono invece impostare in monitor mode e allora li sentite proprio tutto (dati e frame di controllo). Un problema però di quando si mette una scheda in monitor mode è che spesso la scheda sente solo ma poi non può trasmettere (in realtà dipende molto dalla scheda).

La maggior parte dei produttori di dispositivi laptop, smartphone, non hanno la possibilità di fare né una cosa, né l'altra per motivi di sicurezza, cioè hanno impedito di fare questa operazione. Quindi gli hacker seri che davvero cercano di attaccare le reti wireless, tipicamente si dotano di schede Wi-Fi esterne, a cui mettono tipicamente un'antenna seria quindi ci collegano una antenna fisica.

Perché è importante fare già solo lo sniffing? Perché è possibile andare a sentire le reti che ci sono attorno a noi (almeno quelle che mandano i segnali di beacon) quale protocollo di sicurezza utilizzano, etc.

C'è il problema però di quelle reti che non mandano il segnale di Beacon per annunciarsi. Un modo per scoprirlle è stare ad aspettare che prima o poi un host si colleghi a loro, innescando un Probe Request / Probe Response, che permettono di scoprire anche gli SSID nascosti. Se invece non si vuole aspettare che un utente effettui l'associazione, ci sarebbe anche un altro modo che consiste di fare un attacco di disassociazione con cui si manda un disassociation Frame fasullo su un host che è si pensa essere connesso a quell'access point, staccandolo dalla rete. In questo modo l'host cerca immediatamente di riconnettersi e manda immediatamente un prob request che contiene l'SSID della rete nascosta.

### **DDOS over Wi-Fi**

Oltre allo sniffing, molto facile fare il Denial of Service. Le reti wifi sono particolarmente vulnerabili a questo perché è molto facile forgiare dei pacchetti di disassociazione, ci sono anche attacchi basati su questa idea di disassociazione in cui si forza di continuo un host a staccarsi e costringendolo quindi a doversi riattaccare di nuovo. Tipicamente quell'host poi non riesce più a connettersi.

Oppure più banale di questo, ma che richiede un hardware apposito, è possibile causare una tale interferenza sul canale radio rendendo inutilizzabile la rete. Quindi con una interferenza elettromagnetica che viene sparata sulla banda di frequenza dove opera il Wi-Fi.

### **Man in the middle over Wi-Fi**

Un'altra cosa particolarmente pericolosa che fa parte degli attacchi al Wi-Fi, è il classico man in the middle, e lo si fa in due modi, o con la tecnica di ARP Spoofing, oppure, si fa finta di essere un Access point legittimo e l'attacco consiste di queste fasi: prima l'attaccante diventa un Access point finto con un certo SSID che vuole impersonare, su un diverso canale ma l'importante è che sia ad una potenza più alta di quello legittimo. L'attaccante de-autorizza la vittima, che poi cerca di riconnettersi ma si riconnette al fake Access point, quindi all'access point fasullo. A quel punto l'host non si accorge di nulla e l'attaccante fa da tramite, da Relay, per tutti i pacchetti che quell'host vittima scambia con la rete.

### **WEP & WPA**

Per quanto riguarda i protocolli di sicurezza standard, se ne sono succeduti vari nel tempo.

Il primo, il WEP (Wired equivalent privacy) è quello iniziale con cui si prometteva di dare a una connessione wireless le stesse garanzie di sicurezza di una connessione ethernet cablata (dal nome). Però, questo standard iniziale, risultò essere fallimentare quindi facilmente crackabile, legato al fatto che si usava un Cipher RC4.

Nonostante il WEP sia stato attaccato non è infrequente trovare ancora delle reti con WEP. La cosa può sembrare sorprendente, visto che ormai sono vent'anni che si è visto che WEP è crackabile. Ci sono tool che rendono molto facile fare questa operazione, però appunto, ci sono utenti che non fanno attenzione a questo, hanno ancora installati dei vecchi Access point, continuano a usarli, ignari di questo fatto. Questo dimostra che delle falliche ben note, storiche, anche se scoperte possono ancora essere sfruttabili a decine di anni di distanza.

Dopo il WEP, è arrivato **WPA**, che funziona abbastanza bene anche se è stato deprecato, circa nel 2009, ma già molto meglio di WEP. In sostanza, per WPA, si è cercati di applicare delle patch al WEP senza necessità di modificare l'hardware preesistente, quindi, sullo stesso hardware che supporta

WEP, è stato trovato un modo di rendere più sicure le reti, basato sul fatto di usare delle chiavi diverse per ogni pacchetto, diversamente dal WEP che invece ne aveva un'unica chiave per tutti i pacchetti.

In realtà poi è stato rapidamente introdotto una versione migliorata, decisamente migliore il **WPA2** che però richiede un hardware diverso. Qui è stato anche cambiato il protocollo: si è passati dal TKIP ad AED che invece è ormai uno standard considerato più sicuro.

Ci sono due modalità in cui può operare, uno basato su password (come la maggior parte degli AP, ad esempio, quelli casalinghi), e la modalità che si appoggia sullo standard 802.1X – EAP/RADIUS, con cui ci si autentica tramite un server esterno, quindi si demandano tutti i problemi di identificazione ad un server con cui si usano tecniche classiche di certificati per l'autenticazione (questo più in ambito aziendale).

Le reti **WPA2** sono attualmente ancora robuste, cioè non ci sono particolarmente vulnerabili, tranne agli attacchi dictionary based, dove si effettua brute force delle password, e quindi indovinare per tentativi la password utilizzando database di digest, essendo che ci sono delle password particolarmente deboli e soggette a questi attacchi brute force. La tecnica di base è quella di deautenticare un utente legittimo e mettersi in ascolto del 4-step handshake che l'utente immediatamente farà partire per riconnettersi. L'attaccante registra questo 4-step Handshake e lo analizza offline, facendo appunto dictionary based, magari noleggiando risorse di calcolo in cloud potenti per un tempo limitato, ovviamente finché poi non trova la chiave.

# Protocol Security Analysis

Ci alziamo nello stack protocollare. Abbiamo visto le vulnerabilità dei livelli più bassi, dal livello due, poi il livello tre, adesso facciamo ancora un salto in su e andiamo a vedere qualche caso significativo di attacco a protocolli di livello applicativo.

È importante fare qualche premessa sul come si fa in modo serio un'analisi della sicurezza di un protocollo generico. Ci sono dei metodi formali avanzati per studiare la sicurezza dei protocolli. L'idea di base è quella di rappresentare i protocolli con dei modelli a Stati finiti, dimostrare la correttezza andando ad esaminare tutte le cose che possono mai essere possibili, su quel protocollo.

Nel fare queste cose, bisogna però partire da delle assunzioni, ad esempio: cosa è considerato affidabile? il client, il server, la rete? Che informazione può essere Sniffata, forgiata, iniettata, dirottata?

Quando i protocolli sono particolarmente complessi, diventano davvero difficili da studiare perché bisogna considerare davvero tutti i modi in cui è possibile agire. I protocolli vengono attaccati quando li si porta a operare in delle condizioni per cui non era previsto che operassero, violando le ipotesi di base che sono state fatte da coloro che hanno creato quel protocollo. Questo è un pattern abbastanza tipico degli attacchi, non solo i protocolli, ma in generale, quando si costruisce un sistema con delle premesse, facendo delle ipotesi iniziali, è qui che si trova il punto debole, cioè quando l'hacker viola le ipotesi su cui ci si è basati per costruire il sistema. Quindi aggirando in qualche modo queste assunzioni di base, si scardina il tutto. Ad esempio con la frammentazione IP c'era l'ipotesi che un qualunque pacchetto IP ha una certa dimensione massima di 65.536 Byte. Una delle ipotesi che inconsciamente si è adottato, ma che con un banale buffer overflow il tutto veniva distrutto, causando BSOD.

Inoltre, oltre appunto, alla verifica formale dei protocolli a livello teorico, ci sono tutti i problemi che nascono dal fatto che i protocolli necessariamente vengono implementati con del codice vero e proprio e nascono ulteriori problemi legati al fatto che queste implementazioni sono buggate. Ci sono infatti numerosi casi in cui, anche se il protocollo è formalmente giusto (può essere dimostrato anche matematicamente che un certo protocollo è corretto) però poi la sua implementazione è fatta male, vulnerabile, oppure c'è anche il caso in cui l'implementazione invece è fatta benissimo, però poi quando viene utilizzata davvero nel mondo reale viene configurata male. Ad esempio WPA2 è praticamente impenetrabile ma si mette come password 123456...

## FTP Vulnerabilities

Dopo le premesse, FTP è un caso interessante perché, anche se adesso non lo usa quasi più nessuno, dà l'idea di quanto fossero ingenui gli implementatori dei primissimi protocolli di Internet. FTP è vecchissimo, è uno dei primissimi protocolli che sono stati standardizzati negli RFC, infatti è descritto nel RFC 172, uno dei primissimi.

Per ripassare come funziona il protocollo: il client fa l'accesso al server con una fase iniziale dove si scambia username e password.

Nella modalità attiva, il client dice al server quale indirizzo IP e quale porta utilizzare per poi inoltrare dati, quindi aprirà una connessione dalla sua porta 20 all'indirizzo e alla porta specificati dall'host.

Il protocollo così com'è è completamente vulnerabile in quanto non c'è nessuna crittografia nei messaggi mandati, addirittura username e password sono mandati in chiaro. E quindi con un

banalissimo sniffer, è possibile sentire in chiaro username e password e usarle poi per accedere al server e farsi mandare tutti i file che si vogliono.

Inoltre, è possibile aprire una connessione con un altro host: il client malevolo ordina al server di aprire una connessione per i dati ad un certo indirizzo con una certa porta, ma la cosa è che IP e PORTA non devono essere necessariamente IP e Porta del client, ma può essere qualcos'altro, ad esempio un altro host. Questo è chiamato Bounce Attack, basato proprio sul fatto che il Client istruisce il server ad aprire una connessione verso un possibile terzo host.

### **FTP Bounce Attack & Bounce Scan**

L'FTP Bounce Attack sfrutta la possibilità di eseguire il comando PORT con un indirizzo IP di terze parti e una porta arbitraria. Inizialmente, un attaccante si connette a un server FTP che consente il reindirizzamento dei comandi. Quindi, l'attaccante esegue un comando PORT che specifica l'indirizzo IP e la porta di un'altra macchina, che può essere una macchina vittima all'interno della rete interna.

A questo punto, l'attaccante invia un comando FTP "PORT" al server FTP, indicando un indirizzo IP e una porta dell'host di destinazione all'interno della rete interna. Il server FTP invia un pacchetto FTP al client che ha iniziato la sessione. Tuttavia, poiché il client ha aperto la connessione FTP utilizzando l'indirizzo IP e la porta dell'attaccante, il pacchetto viene reindirizzato all'host di destinazione all'interno della rete interna. Ciò può consentire all'attaccante di sondare le porte dell'host di destinazione (Bounce scan) o, in alcuni casi, eseguire comandi non autorizzati utilizzando il server FTP come intermediario (ad esempio, se il client malevolo dice al server di aprire una connessione per trasferire un certo file verso un host, sulla porta 514, che è la porta di remoteShell)

Per il **Bounce Scan** invece, l'attaccante dice al server di provare a trasferire un file verso la vittima su una determinata porta che sta cercando di vedere se è aperta. Il server fa il mestiere sporco e dopodiché ritorna indietro all'attaccante il risultato: un connection refused se la porta è chiusa, successful transfer se quella porta lì era aperta.

## **DNS Vulnerabilities**

Adesso alcuni attacchi che si fanno al DNS, un'infrastruttura senza la quale non funzionerebbe internet. Le debolezze del DNS sono legate al fatto di essere utilizzate di base dal protocollo UDP, che sappiamo essere vulnerabile a spoofing e hijacking.

### **DNS Spoofing**

Un attacco di DNS Spoofing, si basa sull'idea che un host accetti connessioni in ingresso da parte di qualunque host appartenente al suo dominio il che implica una relazione di fiducia tra host.

L'attacco funziona in questo modo: un hacker dall'esterno cerca di aprire una connessione con un indirizzo interno alla rete. Essendo che la vittima non ha la minima idea di chi sia, prima di accettare una connessione in ingresso, cercherà di risolvere l'indirizzo da cui arriva la richiesta, facendo un reverse-lookup. Il server DNS locale della vittima va a chiedere al server autoritativo per quel dominio, qual è il nome per quell'indirizzo, che però è sotto il controllo dell'hacker. Il server sotto il controllo dell'hacker potrebbe rispondere che quell'indirizzo di cui si sta cercando di capire il nome di dominio, è un indirizzo di dominio di cui la vittima si fida. A quel punto la vittima si fida che quella sia una connessione che può accettare perché proveniente dal dominio fidato. In questo modo si apre una breccia di sicurezza.

La mitigazione di questo attacco è basato su un double reverse lookup, con cui la vittima verifica che davvero la corrispondenza dell'indirizzo – domain name sia effettivamente quella.

### **DNS Cache Poisoning**

Un attacco di cash poisoning riesce però a superare anche il double reverse lookup. Questo attacco sfrutta quelle che vengono chiamate Glue Data: nel servizio di DNS, a volte, oltre a tutte le risposte dirette alla query per cui è stata fatta l'interrogazione, si inseriscono delle ulteriori informazioni che si ritiene possano essere utili a chi ha interrogato il server e che vengono "incollate" (glue) assieme alla risposta vera e propria. Ad esempio, per bypassare il double reverse lookup, come glue data basterebbe indicare al server dns della vittima che l'attaccante corrisponde ad un IP di cui si fida (interno alla rete), in questo modo quando la vittima farà la controverifica, il suo server DNS avrà la cache poisoned dai glue data attaccati alla DNS Query Response.

A questo punto si sono prese ulteriori contromisure, ad esempio, un'idea era quella di accettare tutti i glue data, però a patto che chi inoltri i glue data, sia un server autoritativo per quelle informazioni che sta "incollando". Quindi si accetta di prendere informazioni addizionali a patto che si è davvero chi si dica di essere.

### **DNS Hijacking**

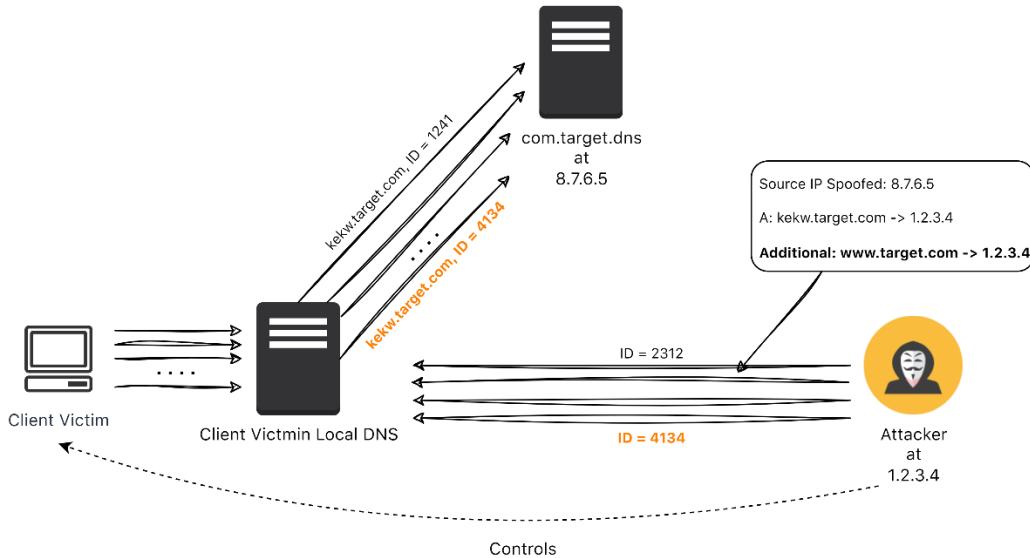
Il DNS è vulnerabile anche ad attacchi di tipo Hijacking. Il problema che sorge nel DNS Hijacking, è simile a quello che si ritrova in TCP Hijacking, e cioè il fatto che le sessioni DNS, similmente alle sessioni TCP, sono contraddistinte da un numero identificativo il quale rende difficile andare a fare DNS Hijacking perché, o si indovina l'ID della sessione DNS corrente in cui ci si vuole intromettere, oppure l'attacco fallisce.

L'idea di base del DNS Hijacking è questa: il client comincia una sessione DNS identificata da un ID. Ovviamente l'attaccante sente questa risposta e induce una DNS Reply Spoofed, con cui cerca di fornire una risposta sbagliata con l'obiettivo ovvio, di poi dirottare il cliente verso un dominio sotto il controllo dell'attaccante con cui farà phishing, o altre malizie. Quello che però succede in uno scenario di questo tipo è che la risposta dell'attaccante va in Race con la risposta legittima, in quanto il server a un certo punto risponde con la sua reply, quindi la reply falsa e la reply legittima tornano indietro più o meno nello stesso tempo, e a seconda di chi risponde per primo, ma soprattutto di chi risponde con una Reply contentente un ID di sessione sensato sarà accettato.

### **The Kaminsky attack**

Nella storia però, è stato ideato un attacco ancora più sofisticato del DNS cache poisoning, questa volta davvero diabolico, che riesce a creare dei problemi anche quando si utilizzano i sistemi di difesa presi per il DNS cache poisoning. Questo attacco, molto famoso, è chiamato il Kaminsky Attack. Attacco di tipo DNS Hijacking, che si basa sul paradosso del compleanno per intromettersi in una sessione DNS.

Per il paradosso del compleanno, trovare una collisione su 16 bit (Identificativo DNS), quindi 65.536 possibili valori, è necessario fare all'incirca 320 tentativi, che non sono tanti. Nell'attacco di Kaminsky, si suppone che l'attaccante abbia il controllo di un terminale che utilizza il server DNS che si vuole avvelenare.



L'attaccante fa sì che la vittima, faccia tante query DNS verso un indirizzo inesistente del dominio target che si vuole attaccare, ad esempio kekw.target.com (che non esiste, ma l'importante è che le richieste vadano verso il dominio che contiene il sito che si vuole impersonare). Il server vittima inoltrerà quindi altrettante query al server legittimo com.target.com, in cui ogni query ha appunto un ID casuale e per ognuna di queste, ritornerà un NX domain (non existing domain). L'attaccante, contemporaneamente a questo grosso flusso di Query, manda nello stesso intervallo di tempo un sacco di risposte con identificativi casuali (al server che si vuole avvelenare), dove si fornisce invece la risposta avvelenata che conterrà la risposta kekw.target.com e il glue data del sito che si vuole attaccare. Per il paradosso del compleanno, entro 320 query, l'attaccante dovrebbe riuscire a trovare una collisione, avvelenando la cache del server DNS target.

## DNSSEC

A fronte di queste malizie ci si è resi conto che l'unico modo con cui davvero si protegge una transazione DNS, è passare alla modalità di DNS sicura (DNSSEC) dove si fa autenticazione dell'entità con cui si parla quindi usando meccanismi a certificati, chiave pubblica, etc.. a quel punto l'attacco di kaminsky non è più possibile.

DNSSEC esiste, è stato standardizzato, però ha i suoi problemi: è molto più pesante, richiede di cambiare decisamente l'infrastruttura del servizio DNS esistente ed ha parecchio overhead.

La soluzione a livello teorico esiste, ma in pratica ci sono ancora un sacco di transazioni DNS non autenticate che girano tranquillamente su UDP di default senza la modalità sicura, e la maggior parte del traffico DNS è così.

## DNS Amplification

DNS Amplification (questo attacco usa Amplification come pattern) serve a fare denial of service in modo simile allo Smurf Attack, però sfruttando il fatto che in rete ci sono dei server DNS chiamati OPEN Resolver (o Public Resolver) che accettano (e rispondono) a query provenienti da qualunque parte del mondo. Esistono quindi server DNS del tutto aperti, che chiunque può configurare e usare, ad esempio 8.8.8.8 di google è un public resolver.

Quindi si può sfruttare l'esistenza di questi open resolver lanciando un attacco di questo tipo: mandando delle query di tipo ANY (le query ANY ritornano tutti i record relativi ad un dominio) con

cui si chiede al DNS di fornire tutte le informazioni che ha per un certo dominio, quindi generando un sacco di traffico di risposta. Se a quel punto inserisco un IP Spoofed, viene applicato il principio di amplificazione effettuando Distributed Denial of service.

### **DNS Tunneling**

Il tunneling DNS si riferisce alla manipolazione del protocollo DNS per indirizzare il traffico dannoso oltre le difese di un'organizzazione. Utilizzando domini e server DNS dannosi, un utente malintenzionato può utilizzare il DNS per eludere la sicurezza della rete e far uscire dei dati. Questo perché le organizzazioni consentono al traffico DNS di passare attraverso i propri firewall.

L'idea è che si fa finta di emettere delle query DNS, ma in realtà dentro le query DNS vengono inseriti dati che si vuole far uscire dalla rete, codificando ad esempio i dati all'interno di un nome di dominio richiesto. Non si riesce ovviamente ad ottenere un grosso volume di traffico.

# Segnalazione

Questo argomento riguarderà la segnalazione ed il piano di controllo in varie architetture di rete. Ci sono architetture diverse da quella di internet (dove la segnalazione è quasi nulla), dove invece le situazioni sono molto diverse, ad esempio la rete telefonica classica che utilizza il sistema di segnalazione che si chiama SS7.

Per segnalazione si intende generalmente uno scambio di messaggi che avviene tra diverse entità della rete per mettere in piedi, mantenere e terminare una connessione dati. Ma più in generale, nella maggior parte dei casi, si fa segnalazione dentro una rete per offrire un servizio orientato alla connessione. La segnalazione però si può fare in vari modi. Quindi uno dei degli obiettivi di questa parte è cercare di capire quali sono i vari aspetti della della segnalazione, in modi diversi con cui è possibile farlo dentro una rete.

La segnalazione quindi non avviene semplicemente all'inizio, prima di uno scambio di dati, ma può anche avvenire durante la comunicazione stessa e sicuramente avviene al termine della comunicazione, per poi rilasciare le risorse e terminare la connessione.

Quale potrebbe essere un motivo di fare segnalazione durante una chiamata, mentre i dati stanno già fluendo? Uno dei motivi è che si puo voler cambiare la qualità / la codifica della connessione, perché ci si accorge che la banda disponibile è tale per cui si debba poter aumentare la qualità o diminuirla, quindi si informerà l'altra parte di un cambio di codifica dei dati.

Oppure, pensando ad una sessione multimediale Audio / Video, ci sono tanti servizi aggiuntivi che si può voler aggiungere / rimuovere durante una sessione multimediale, ad esempio, quando si aggiunge un nuovo partecipante, un partecipante abbandona, un partecipante accende o spegne il suo audio, il suo video, etc. C'è quindi bisogno di accompagnare questi gesti con della segnalazione che evidentemente avviene durante la comunicazione.

Un'altro motivo per cui le connessioni dati sono sempre accompagnate da una connessione di controllo che avviene in parallelo ai dati, su cui girano delle informazioni di controllo contemporaneamente al flusso, è per fare misurazioni (come RTCP). Ad esempio per misurare quanti pacchetti sono stati trasmessi (perché poi se un servizio a pagamento usato sul tempo necessariamente si ha necessità di fare delle misure), volume di traffico scambiato durante la sessione, etc.

È possibile inoltre, fare segnalazione tra:

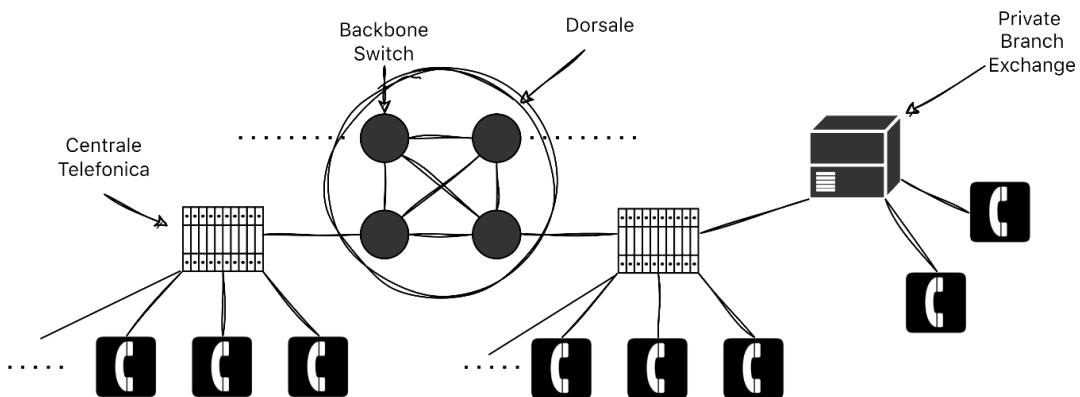
- Utente finale ed un elemento della rete.
- Utente finale ed utente finale.
- E poi esiste, in teoria, anche una segnalazione interna alla rete tra elemento di rete ed elemento di rete

## SS7

il primo caso di segnalazione, è quello della rete telefonica tradizionale su cui gira il sistema di segnalazione numero 7 (Signaling System No. 7), usato anche nelle reti cellulari.

La rete telefonica classica, è un po la mamma di tutte le reti. Si è cominciato a costruirla alla fine dell'Ottocento, ancor prima che iniziasse Internet, ed era diffusa a livello globale (avvolgeva tutto il pianeta). Quindi è stata creata un'infrastruttura capillare stendendo i doppini di rame che arrivavano in tutte le case di quasi tutto il mondo. Questa rete telefonica classica è un po in dismissione, cioè si trova ancora funzionante come un tempo, però viene sempre più rimpiazzata da VOIP.

È una rete in cui (a differenza di internet) i terminali sono stupidi. I terminali sono i telefoni che banalmente vengono collegati con un doppino telefonico ad una presa a muro che fornisce anche l'alimentazione. Una volta collegati sono già pronti, sentendo il tono classico. Quindi, a differenza di Internet, dove il cuore della rete è semplice ed i terminali sono complessi, qui abbiamo una situazione opposta, i terminali sono estremamente stupidi, semplici, mentre concettualmente la rete di backbone è sofisticata perché fa commutazione di circuito (non è una commutazione di circuito virtuale). La rete quindi viene ingegnerizzata in modo tale da poter trasportare un unico tipo di dato (chiamata telefonica) che ha una banda fissa molto nota: 64kbps.



I telefoni delle case, tramite un doppino telefonico, arrivano dentro degli armadi di distribuzione di quartiere (centrale telefonica). Queste centrali telefoniche, sono sparse su tutto il territorio, collegate poi da una dorsale centrale fatta da degli switch di backbone, dei commutatori interni alla rete.

La parte che collega gli utenti finali alla centrale telefonica è chiamata ultimo miglio (local loop).  
Questi collegamenti possono anche essere anche dell'ordine di 10 km

La rete telefonica utilizza una tariffazione completamente diversa da Internet, dove le singole linee che connettono la centrale telefonica agli utenti, vengono fatte pagare. Cioè in generale, gli utenti sottoscrivono un servizio di telefonia e pagano: una quota fissa di uso della linea più una quota tariffata tipicamente a tempo, per le telefonate

### Private Branch Exchange

Quando bisogna gestire delle organizzazioni dove ci sono tanti utenti, conviene mettere il centralino telefonico privato il che significa che l'azienda paga una sola volta la linea che collega la centrale telefonica al centralino telefonico privato e usa quest'ultima effettua multiplexing di tanti utenti che, nei loro uffici hanno ciascuno il loro telefono, ma che condividono tutti lo stesso numero fisso. Conviene molto di più attaccarli tutti a un centralino telefonico privato e poi pagare verso l'operatore telefonico un solo numero. L'obiettivo è di abbattere il costo/utente per accesso alla rete.

Quando si necessita di acquistare delle linee, si contatta il fornitore di servizio di telefonia classica, noleggiando le linee di cui si necessita. Infatti si mettono in piedi N canali fisici, non c'è interfogliamento di chiamate su un unico filo di rame.

La dorsale telefonica è composta quindi da delle centrali di transito molto grosse (c'è ne sono qualche decina per ogni nazione). Sono molto grosse e possono essere attraversati da tante chiamate contemporaneamente, sono ovviamente connessi da dei canali ad altissima capacità dove si affacciano tantissime linee da 64Kbps.

## **Come viene trasmessa la voce**

La voce è un tipo molto particolare di traffico, la sua codifica standard è la PCM che trasforma la chiamata in un flusso digitale a 64 kbps. Però fino agli anni 60 più o meno, la rete telefonica era interamente analogica. Quello che si faceva era una multiplazione a divisione di frequenza (FDM), cioè, non c'era il concetto di bit, il segnale non veniva trasformato in un flusso digitale, era tutto in analogico end to end, modulando le varie chiamate attorno a delle frequenze diverse, facendo una divisione di frequenza.

All'inizio, quando gli utenti erano pochi, ci si poteva permettere di utilizzare delle operatrici che commutavano i circuiti manualmente da delle cabine di commutazione. Questa soluzione poi divenne sempre più automatica, in cui l'uomo fu sostituito da dei commutatori elettromeccanici, quindi dei oggetti interamente autonomi. Poi è avvenuta attorno agli anni 60, la migrazione verso il digitale con l'idea che il local loop poteva (possono ancora ancora oggi) essere analogico, quindi l'ultimo miglio è ancora oggi analogico. Il segnale, arrivato nel primo punto della centrale telefonica, viene convertito in digitale: un modem lo converte in digitale trasformandolo in un flusso di bit e in tutto il resto della rete viene trasportato in TDM (time division multiplexing). Il fatto di aver trasformato storicamente la rete da analogica digitale, ha reso possibile l'introduzione di un sacco di servizi aggiuntivi, ad un sacco di benefici sotto tantissimi punti di vista, tra tanti quello della sicurezza.

Il mondo della telefonia ha fatto dei tentativi di trasportare non soltanto voce, ma fornire dei servizi anche per i dati. Uno dei primi tentativi è stato l'ISDN, con cui negli anni 90 gli operatori telefonici cercavano di vendere anche delle connessioni per i dati. Poi però questa tecnologia è morta.

Quella che invece ha avuto sicuramente molto successo e almeno in Europa è l'ADSL con cui si è scoperto che potevano passare sia voce che dati contemporaneamente sullo stesso doppino che era già stato steso e che arrivava nelle case.

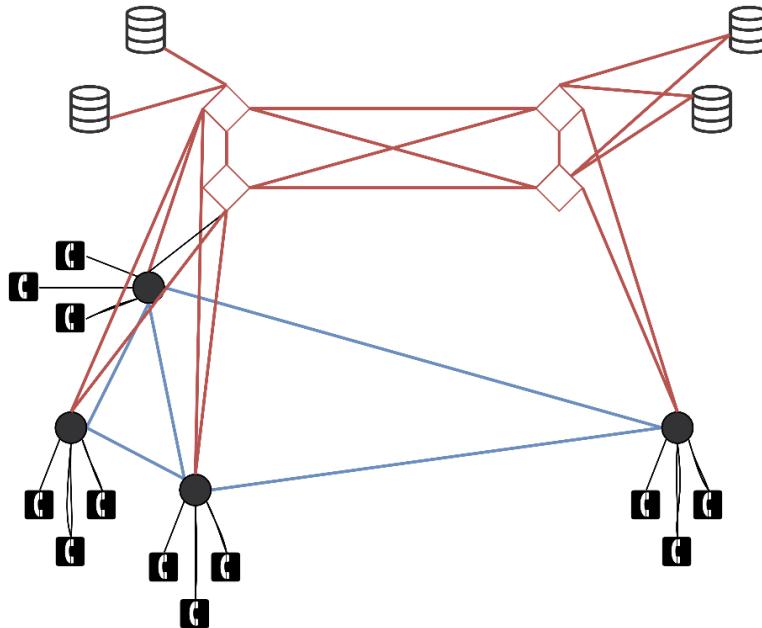
Ora ormai abbiamo le fibre ottiche che arrivano in quasi tutte le grosse città e con cui si è rimpiazzato il doppino telefonico di rame, che ha ovviamente creato un mezzo di comunicazione con una banda estremamente superiore.

## **Rete telefonica intelligente**

Quando quindi si è passati dall'analogico al digitale si è fatto un'enorme salto qualitativo enorme, in quanto, si è reso possibile aggiungere una molteplicità di servizi che nella rete analogica erano impossibili da realizzare, la cosiddetta rete telefonica intelligente.

Ad esempio: l'aggiunta dei numeri verdi, numeri speciali, numeri per emergenze, numeri per donazione (numeri dove la tariffazione avviene a carico del destinatario), possibilità di vedere l'identificativo del chiamante, la richiamata su occupato, la segreteria telefonica, la deviazione di chiamata, blocchi, restrizioni varie, eccetera. Quando si è passati al digitale, sono stati gradualmente introdotti perché si è uniti al traffico dati classico anche tutto un meccanismo di segnalazione che ha reso possibile tutta questa molteplicità di servizi.

## Data plan & control plan



L'idea è più o meno in questa immagine: essenzialmente ci sono due reti, uno è il data plane (blu) ed è quello su cui scorrono i dati, i bit di dati, quindi flusso vocale vero e proprio. Parallelamente a questa rete, ma fisicamente diversa, c'è invece il piano di controllo (Control Plane - rosso) su cui invece gira il traffico di segnalazione, il traffico SS7. Notare che per motivi di sicurezza, il piano di controllo è ridondato almeno di due unità, per far sì che sia tollerante a guasti (ogni centrale telefonica, è collegata ad almeno due elementi più interni, in modo che se uno dovesse per rompersi per qualche motivo, c'è l'altro).

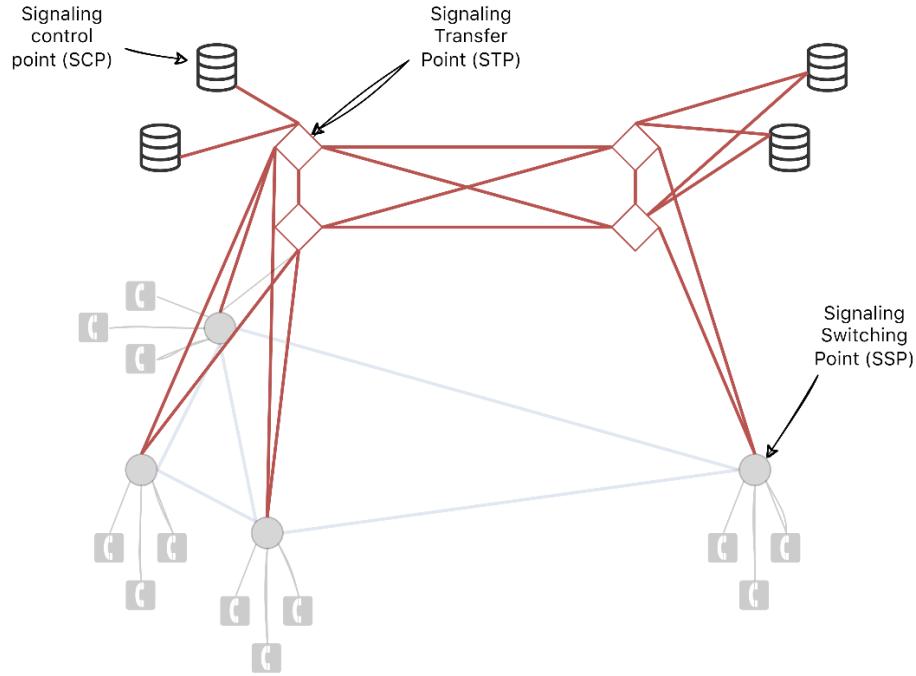
Si nota che è una filosofia completamente diversa da quella che si usa in Internet, in cui tutto mescolato: il traffico dati e il traffico di segnalazione girano sugli stessi link fisici (in-band). Qua invece la segnalazione è fuori banda (out-of-band): significa che i messaggi di segnalazione sono trasportati su una rete fisicamente separata da quella dei dati.

Questa separazione, per cui appunto il traffico di controllo è trasportato in modo completamente diverso, separato, ha un sacco di benefici perché permette di fare messaggi di segnalazione anche durante la chiamata (non vengono quindi mescolati ai 64 kbps di voce) di un host, che viaggiano appunto su un'altra rete separata, tipicamente scarica, su cui viaggiano questi pacchetti di controllo e permettendo, tra l'altro, di andare a velocità molto maggiore di quella del piano dati perché nel piano dati si è vincolati comunque ad andare a 64 kbps. Nella rete di controllo è possibile andare a velocità molto superiori, fare le cose molto più veloci, che è una buona cosa, anche perché bisogna dare, un po più di priorità al traffico di controllo.

Ma è il motivo principale per cui si è deciso di fare così, in realtà, è legato alla sicurezza. Se si permettete che il traffico di segnalazione viaggi sulle stesse linee su cui viaggiano i canali dati, si rende il tutto più vulnerabile a dei malintenzionati che possono mandare traffico di segnalazione fasullo. Infatti, intorno agli anni 70/80, c'era qualche malintenzionato che aveva scoperto che emettendo dei toni (con delle cosiddette Blue box) su delle frequenze particolari, ingannavano la rete facendole credere che fossero messaggi di segnalazione legittimi con cui si instauravano delle comunicazioni, anche a lunga distanza, in modo gratuito. Con la separazione del piano dati dal piano di controllo non è stato più possibile effettuare questo, anche perché il piano di controllo ora non raggiunge neanche fisicamente gli utenti: gli utenti hanno doppino telefonico dove scorrono solo

dati, non hanno più accesso alla rete di controllo che, si attesta solo sulle centrali telefoniche, ma le centrali telefoniche sono sotto il controllo dell'operatore, quindi non è più possibile mandare questi messaggi fasulli da parte degli utenti.

La rete SS7, sul piano di controllo è a commutazione di pacchetto, sul piano dati è a commutazione di circuito.

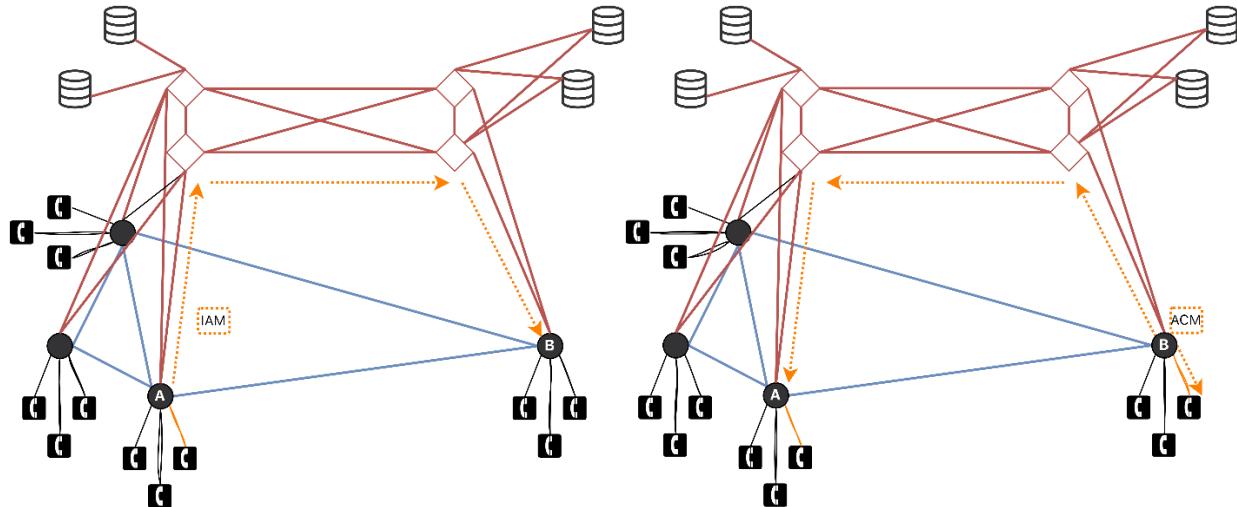


Terminologie:

- Signal switching point (SSP), che sono quegli elementi della rete SS7 del piano di controllo più vicini agli utenti finali, centrali telefoniche distrettuali.
- Signaling Transfer Point (STP), sono semplicemente dei commutatori di pacchetto, che inviano, ricevono ed instradano i messaggi di segnalazione, i messaggi di controllo della rete SS7
- Signaling Control Point (SCP), dove invece c'è la parte intelligente della rete SS7, quella che ospita i servizi, fa le funzioni di database e dove vengono implementati i servizi della rete intelligente.

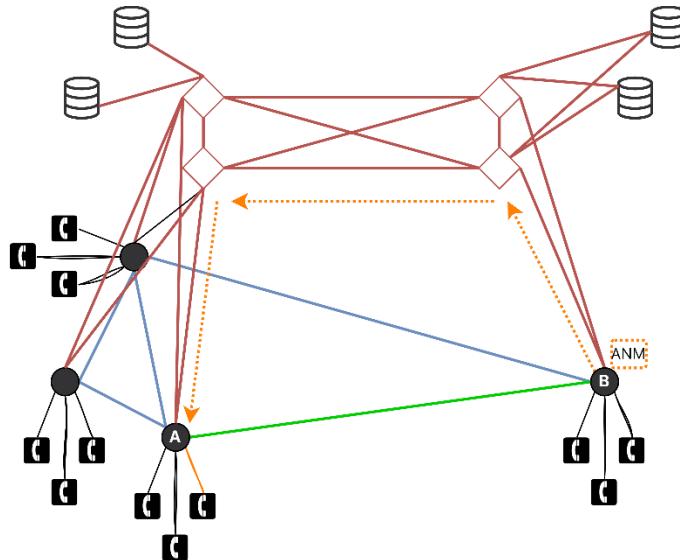
Esempio di cosa succede nel piano dati per mettere in piedi una comunicazione telefonica tradizionale tra un utente A e un utente B.

Il chiamante compone il numero del chiamato, ed il primo elemento della rete SS7 che entra in gioco e si trova dentro la centrale telefonica il quale, verifica se esiste l'utente da chiamare, e se sul piano dati ci sono le risorse disponibili, quindi c'è un canale libero, eventualmente Multi-Hop, per inoltrare la chiamata Verso B. Supponendo che ci sia questo canale (nella maggior parte quasi c'è sempre, la rete è dimensionata in modo tale che succede con probabilità di  $10^{-12}$ ) e supponiamo il caso che ci sia un singolo hop per semplicità, allora riserverò quel canale a 64 kbps (viene prenotato).



Poi nella rete di controllo, l'SSP di A, inoltra un pacchetto di Initial Address Message – IAM, che viaggia dentro la rete SS7, attraverso vari commutatori finché non arriva all'SSP di B, che si accorge che la chiamata è destinata a uno degli utenti che sono direttamente connessi a lei e costruisce un messaggio di Address Completion Message – ACM, che viene inoltrato a B facendogli squillare il telefono.

Contemporaneamente, genera lo stesso messaggio di completion message che viene però mandato ad A (lo fa passare lungo il piano di controllo) lungo la rete SS7 che fa squillare anche il telefono di A. Questa è solo la fase preliminare: i due telefoni stanno squillando, ancora non è stato fatto partire nulla.



Quando il destinatario decide di accettare la chiamata e rispondere, questa volta la centrale di B genererà un nuovo pacchetto chiamato Answer Message – ANM, che di nuovo torna indietro attraverso i vari commutatori, arrivando nella centrale di A ed a questo punto si può mettere in piedi un circuito bidirezionale vero e proprio di dati con cui si stabilisce la chiamata vera e propria.

Quando e poi la chiamata termina, girano di nuovo dei pacchetti nella rete SS7 e contemporaneamente poi si rilascia il canale sul piano dati che si era allocato per quella chiamata.

## ATM

ATM (Asynchronous transfer mode), è stato un tentativo di proporre un'architettura completamente diversa da Internet (tentativo fatto negli anni 90), sotto la spinta soprattutto degli operatori di telefonia tradizionale che avevano in mente di costruire una rete che trasportasse non solo chiamate ma qualunque tipo di traffico ad alta velocità. Inoltre c'era anche l'ambizione di fornire qualità del servizio garantita, quindi completamente diverso dall'approccio best effort di Internet, utilizzando commutazione di circuito virtuale end-to-end, stanziando delle risorse nella rete.

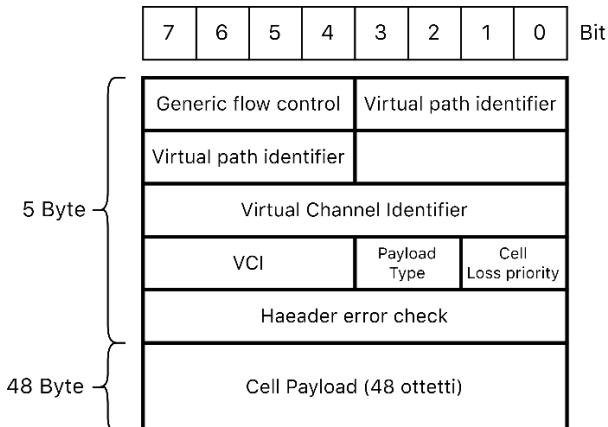
Questa alternativa ad Internet è fallita, anche se è stato fatto lo sforzo, sono stati anche investiti un sacco di risorse, però non c'è l'ha fatta a competere con Internet. Sopravvive forse in alcuni pezzi della rete ma in modalità molto ridotta.

### Circuiti virtuali

Venivano quindi stabiliti dei circuiti virtuali su cui venivano trasportati dei piccoli pacchetti chiamati **celle**, effettuando a tutti gli effetti una divisione del traffico. Questi pacchetti poi venivano trasportati su dei circuiti virtuali, ed erano di due tipi:

- VC permanenti (PVCs): pensati per le connessioni di lunga durata che quindi restavano stabili nel tempo.
- VC Commatuti (SVCs): creati dinamicamente in risposta a singole richieste di nuove connessioni.

Ed è in particolare quest'ultima modalità che è fallita, cioè l'idea di fare, per ogni nuova richiesta di traffico, un nuovo circuito virtuale commutato è risultato un fallimento. La creazione invece di circuiti virtuali permanenti è l'unica forma di ATM che ancora è sopravvissuta un po nel tempo, in parte fino ai nostri giorni in alcune funzioni limitate della rete.



I pacchetti, chiamati appunto celle, erano davvero piccolissime, erano solo di 53byte. In cui, nei primi 5 byte di Headeder c'è l'identificativo del circuito virtuale di appartenenza. Quindi ognuna di queste celle aveva come parte fondamentale dell'intestazione gli identificatori (2 identificatori):

- identificatore di percorso
- identificatore di channel.

Con l'idea di creare vari channel all'interno di uno stesso path. Poi un payload di 48 byte.

L'idea che c'era dietro ad una scelta di celle di dimensione fissa, è che si pensava di poter realizzare una commutazione veloce, cioè di costruire degli switch all'interno della rete molto veloci, che riuscissero a smistare queste celle in modo estremamente efficiente per il fatto che erano di dimensione fissa, il che aiuta molto la costruzione di matrici di commutazione, e poi perché essendo di dimensione ridotte, venivano davvero commutate in modo rapido.

## Commutatori ATM

I commutatori ATM facevano l'instradamento delle celle sulla base di una forwarding Table interna, che era fatta in questo modo:

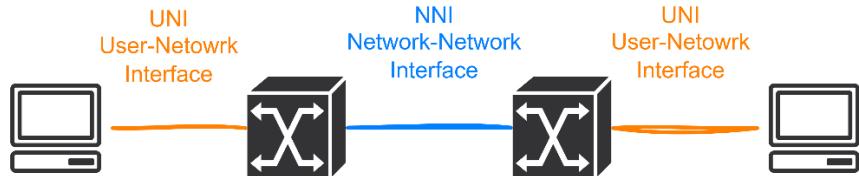
- Interfaccia in ingresso (da cui arriva la cella)
- Numero di circuito della cella in ingresso
- Interfaccia di uscita (da cui esce la cella)
- Numero di circuito della cella in uscita

Quindi sull'interfaccia di uscita una cella eventualmente usciva con un'altro identificativo di circuito virtuale: era quindi permesso agli identificatori del circuito virtuale di cambiare tra ingresso e uscita di ogni switch.

Il problema era che queste tabelle andavano costruite e mantenute aggiornate, quindi è evidente che ogni switch all'interno della rete doveva mantenere un'informazione di Stato su tutte le connessioni che lo stava attraversando e che potevano essere tantissime: centinaia di migliaia milioni di connessioni in contemporanea che attraversano lo stesso switch.

## Segnalazione su ATM

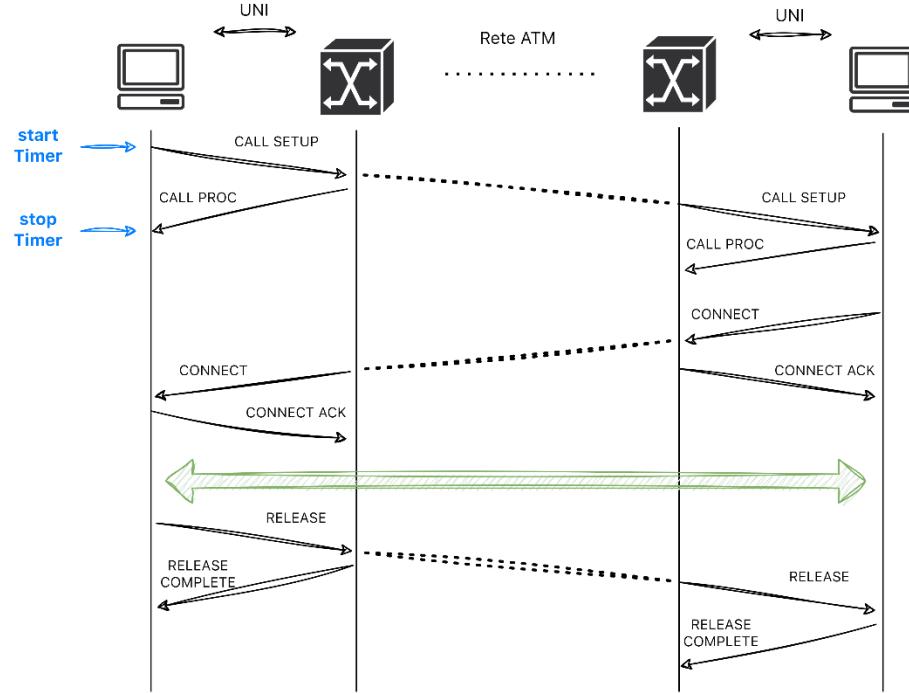
Per attivare questi circuiti virtuali c'era necessità di fare segnalazione. Quindi una fase di call setup, di instaurazione della connessione e poi di rilascio della connessione stessa. Accanto ad ATM c'era un protocollo di segnalazione piuttosto sofisticato che si chiamava Q.2931.



Lo standard gestiva tutte le fasi della segnalazione varie che ci sono, quindi:

- Tra utente della rete ed elementi interni alla rete (UNI)
- Tra elementi della rete ed altri elementi della rete (NNI)

Era uno standard molto ambizioso che aveva addirittura previsto di poter fare sia connessioni punto punto sia connessioni punto multipunto (multicast) negoziando requisiti di banda, quindi garantendo una qualità del servizio con anche i requisiti diversi tra i due host (simmetrici o asimmetrici) lungo il percorso. E poi tutta una parte di gestione di recupero degli errori che è un aspetto critico di Q.2931.



C'era una fase di connection setup che girava su un particolare circuito virtuale preconfigurato che era un canale di segnalazione che veniva usato per la fase di setup, in questa fase il chiamante specificava un indirizzo di destinazione e dei suoi requisiti di qualità del servizio. La rete gli rispondeva se era in grado di ammettere quella chiamata, segnando un certo circuito virtuale e riservando le risorse necessarie. La stessa cosa avveniva poi lungo tutto il percorso della rete, dove si creavano via via le tabelle di inoltro sui commutatori finché non veniva informato l'utente finale di questa chiamata in arrivo. Se l'utente accettava la richiesta di chiamata, un messaggio di Connection Established, tornava indietro all'utente e alla fine di questa fase preliminare di instaurazione della connessione, finalmente, si poteva cominciare a trasmettere dati.

Al termine della chiamata, una fase finale chiamata connection release, rilasciava tutto quanto, quindi di nuovo end to end bisognava rilasciare tutte le risorse prenotate e cancellare quindi le entry nella tabella di inoltro di tutti i commutatori attraversati.

In tutte le fasi di segnalazione, eventuali errori, perdite di pacchetti e problemi vari venivano gestiti con tutta una serie piuttosto sofisticata di **timers**, che funzionavano in questo modo: ogni volta che si mandava un messaggio di segnalazione di un certo tipo, si faceva partire un timer associato a quel messaggio, con l'idea che doveva tornare prima o poi un ack, una risposta dall'entità con cui stavo parlando. Se non si riceve risposta si riprovava, quindi di nuovo si faceva un nuovo tentativo di mandare il pacchetto aspettando che entro una scadenza fissata tornasse indietro una risposta (questo veniva fatto su tutti gli elementi della rete e per tutti i pacchetti venivano fatti partire, poi arrestati tutta una serie di timer specificati nel protocollo).

Questa modalità di effettuare segnalazione, è chiamata **hard state** che prevede quindi che ogni comunicazione di segnalazione sia riscontrata con successo e quindi ci sia davvero una garanzia che la segnalazione tra gli elementi abbia avuto successo grazie a questo meccanismo di conferma.

Sembra un sistema robusto, in quanto in cui si effettua la segnalazione, garantisce che la comunicazione venga avviata con successo e riserve tutte le risorse, però un meccanismo di questo tipo, hard state, ha le sue debolezze.

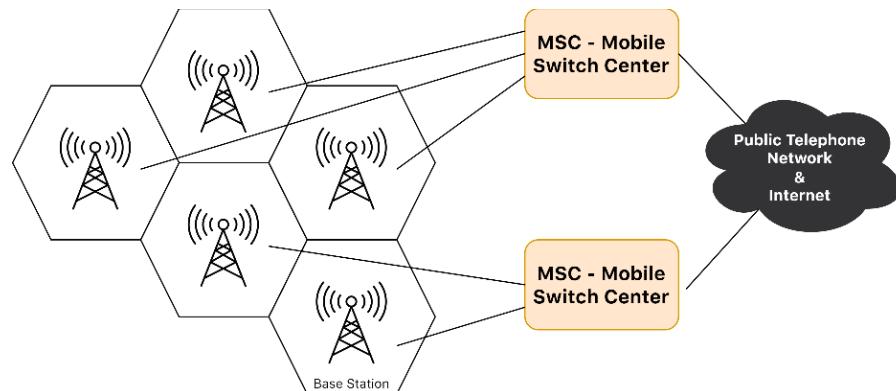
Si suppone ad esempio, che durante la connessione, uno dei due partecipanti, vada in crash. Cosa fa la rete? È un problema per la rete? La risposta è sì: la rete ha instaurato tutta una serie di Stato nei vari commutatori, risorse e banda, etc. E la rete deve accorgersi che c'è stato un malfunzionamento da qualche parte e andare a risolverlo, rilasciando tutto lo stato allocato. Ma un meccanismo di questo tipo basato su una call-setup e timers, da solo, non recupererà il problema. Non recupera il problema perché una volta realizzata la connessione con successo e si è durante la fase di traffico dati utenti, il meccanismo di timers non esiste più.

Quindi se succede qualcosa durante la fase di connessione vera e propria che cosa salva il tutto? Non esiste nulla all'interno del protocollo di segnalazione Q.2931 che può salvare questa situazione, deve esserci qualcosa di esterno, che avvisi il sistema che c'è stato un problema e che bisogna quindi rilasciare tutto lo stato allocato. Però, di per sé, per come è definito un sistema hard state, anche se è garantito instaurare con successo la connessione, da solo non riesce a recuperare facilmente problemi di malfunzionamento dei link o i nodi.

## Rete cellulare

Le reti cellulari, sono al giorno d'oggi assolutamente integrate a Internet, al punto che noi usiamo i nostri telefoni per fare più o meno tutte le cose che facciamo da un host fisso.

Hanno però il problema fondamentale della parte di ultimo miglio (il tratto radio), la parte wireless, dove la rete deve affrontare il problema principale che è quello della gestione della mobilità, cioè il fatto che gli utenti si spostino da una zona all'altra.



Nella rete cellulare, il territorio è diviso in una serie di celle che tappezzano, senza buchi, la superficie di una nazione. Ogni cella ha una stazione base (base station) che irradia la zona predisposta. Quando gli utenti hanno necessità di spostarsi, devono disconnettersi da una base station e attaccarsi ad una nuova (meccanismo di hand over) che stressa molto la rete dal punto di vista della segnalazione, in quanto il tutto deve essere fatto in tempi rapidissimi e garantendo la continuità di tutte le connessioni in corso.

Ci sono poi dei commutatori dentro la rete cellulare, che si chiamano MSC (mobile switching center) che gestiscono un'insieme di celle.

### Il primo hop

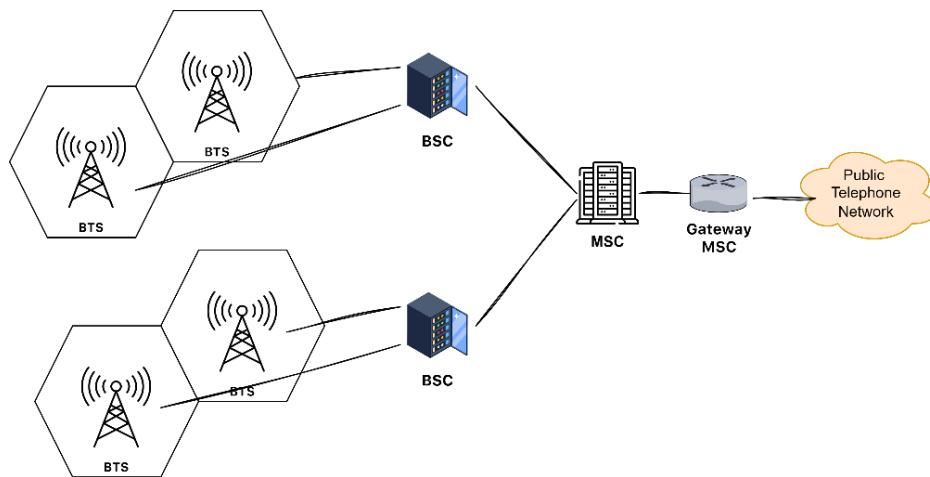
Il primo hop, è quello che collega gli utenti mobili alle loro base station, ed è l'unica parte effettivamente wireless della rete su cui si usano tecniche diverse di condivisione dello spettro, che è una risorsa molto preziosa che si contendono gli operatori telefonici e che è soggetta a licenze molto costose, quindi bisogna utilizzarla al meglio.

Per dividere il canale si utilizzano tecniche varie. Nelle prime generazioni di reti cellulari digitali, quindi il GSM (2G), si utilizzava una tecnica mista FDMA/TDMA, quindi si faceva multiplazione di frequenza e di tempo. Dal 3G (UMTS) in poi si è invece fatto largo uso di una tecnica diversa, basata su CDMA.

## 2G - GSM

La prima generazione digitale è stata quella del 2G (GSM). È uno dei pochi standard che ha avuto un successo planetario che è stato inventato dagli europei ed è stata l'unica sopravvissuta che è ancora attualmente in uso in tutto il mondo. È un po' l'ultima spiaggia a cui si arriva quando tutte le altre generazioni di livello successivo non sono utilizzabili, quindi quando non c'è la copertura di generazioni successive, per retrocompatibilità può ancora succedere di tornare indietro a utilizzare gli schemi del GSM per i dati che si chiama EDGE. La banda che però mette a disposizione per i dati è davvero bassa, insufficiente per la maggior parte delle applicazioni moderne.

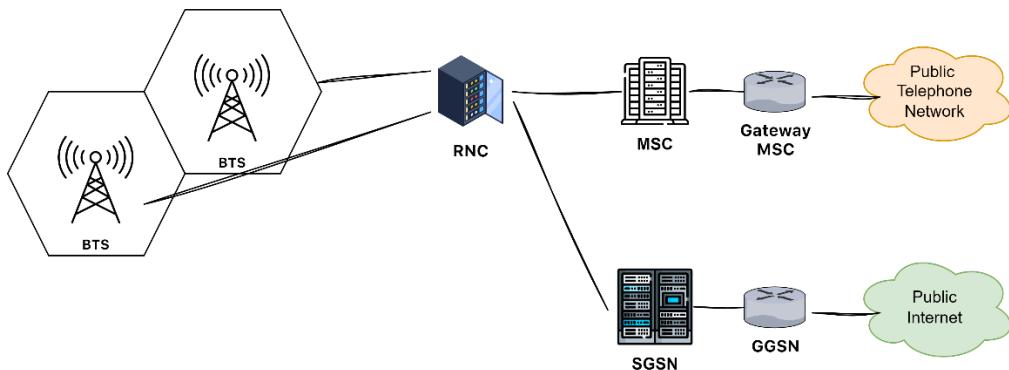
In effetti era una rete nata principalmente per le chiamate, infatti per le chiamate funziona ancora bene. Anche perché ormai con i metodi di compressione che abbiamo, la voce viene compressa in modo estremamente efficiente, anche fino a 13 - 6Kbps, quindi trasportare la voce non è un problema.



Nella rete 2G ogni Base station è collegata ad un Base Station Controller – BSC, che a sua volta è collegata al Mobile Switching Center – MSC. Ogni MSC è interfacciato con il gateway MSC che ha accesso diretto alla rete telefonica tradizionale e quindi da lì in poi ad SS7, permettendo a tutti gli effetti telefoni mobili di usufruire di tutti i servizi di rete telefonica intelligente a cui si è abituati da un telefono fisso.

## 3G - UMTS

Solo con il 3G (UMTS), si è cominciato a fornire un traffico dati decente. Lo si faceva però separando completamente i due tipi di traffico.



Quindi nella parte wireless, voce e dati viaggiavano assieme, poi alla base station controller (Che nel 3G viene chiamata **Radio Network Controller – RNC**) venivano immediatamente separati. La voce viaggiava come prima attraverso dei commutatori che si interfacciavano con la rete telefonica pubblica (la parte voce rimane invariata). I dati, invece, viaggiavano attraverso degli oggetti chiamati **Serving GPRS Support Node – SGSN** e poi al Gateway **GPRS Support Node – GGSN**. Le due reti in questo modo riescono ad operare in parallelo.

Il 3G poi ha avuto delle migliorie, versioni intermedie diciamo. Quindi se si sentono acronimi del tipo HSPA, HSDPA, sono delle migliorie che sono state apportate nella parte wireless per fornire bande sempre crescenti agli utenti. Quindi, sui telefoni se si vede comparire, ad esempio H+, si è sempre all'interno di UMTS dove andiamo però a utilizzare delle interfacce radio più sofisticate.

### Gestire la mobilità in una rete cellulare

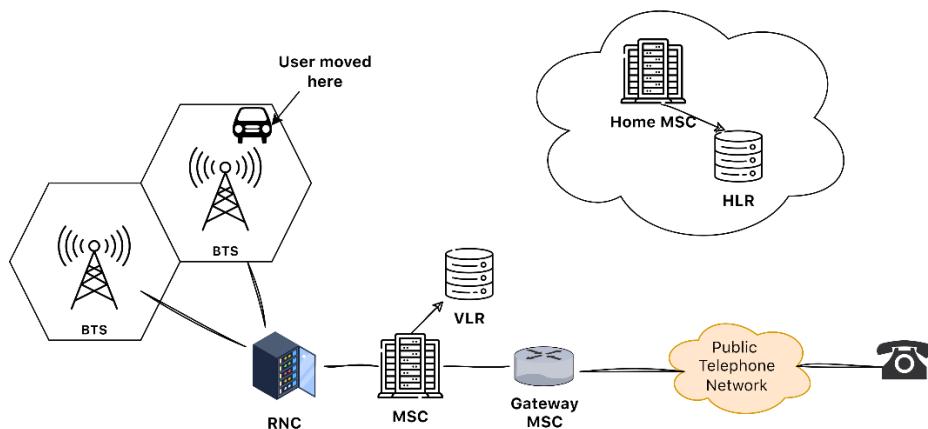
indipendentemente dalla generazione a cui appartengono, la gestione della mobilità è la medesima, i principi sono gli stessi in tutte le generazioni.

Per motivare la gestione della mobilità si consideri questo scenario: Un utente fisso che cerca di chiamare un utente mobile, come viene raggiunto quell'utente mobile? Come si gestisce il caso in cui l'utente mobile si sposta (anche da una nazione all'altra)? In uno scenario reale, l'utente è immerso in reti appartenenti a operatori diversi, quindi c'è anche il problema di gestire la mobilità tra operatori diversi, ma in questo caso supponiamo esista un solo operatore.

Il tutto è gestito con dei cosiddetti location register, quindi delle basi di dati, dei registri che tengono traccia della posizione degli utenti. Ci sono due tipi di questi registri:

- **Home Location Register (HLR)**: un database (ogni operatore di telefonia ha uno (o più) di questi oggetti) gestito dall'operatore a cui l'utente è abbonato e serve a gestire tutte le informazioni di un certo utente, in particolare quelle del suo profilo. quindi sono informazioni statiche di profilo: che tipo di contratto ha, che servizi ed abilitati, il traffico residuo, il credito residuo, etc
- **Visitor Location Register (VLR)**: sono invece quei registri che tengono traccia di quali sono gli utenti in visita, cioè quelli che si sono spostati e che sono finiti in un certo momento dentro una zona della rete gestita da un certo MSC (quindi ogni MSC ne ha uno).

Quello che succede è che all'interno dell'home location register viene aggiunto, oltre alle informazioni statiche degli utenti, un puntatore al VLR, quindi un puntatore al visitor location register su cui l'utente si trova attualmente (e quindi a quale MSC è collegato).



Quando quindi l'utente di rete fissa chiama l'utente mobile succede che: la rete telefonica tradizionale dove c'è il chiamante, in base al numero che chiama, comincia ad instradare la chiamata verso l'Home location register, quindi verso l'operatore a cui è abbonato l'utente mobile e lo fa attraversando la rete telefonica pubblica ed entrando nella rete dell'operatore attraverso il GMSC che fa appunto da gateway verso la rete telefonica pubblica. Accanto al GMSC tipicamente è posizionato l'HLR (home location register). L'operatore a quel punto cerca dentro l'HLR per individuare il puntatore al VLR dove l'utente mobile dovrebbe trovarsi in quel momento (in realtà, l'ultimo VLR che l'utente mobile ha visitato). Tramite un roaming number, la chiamata viene instradata verso l'MSC associato al VLR puntato (VLR è tipicamente collocato accanto al mobile switch in center MSC). Il mobile switching center, non sapendo in che punto preciso della zona coperta si trova l'utente mobile, irradia in broadcast la zona con un segnale chiamato di Paging. Se l'utente è lì e risponde, la chiamata viene completata.

Manca ancora la parte in cui l'utente mobile si sposta. Quando un utente mobile cambia MSC oppure spegne il telefono / modalità aereo, e poi si riconnette alla rete in un'altra location area, è l'utente stesso che dovrà far partire una procedura cosiddetta di location Update: l'utente mobile che si accorge di aver cambiato location area (perché riceve dalla rete, un location area identifier diverso da quello che aveva prima), informa la rete di aggiungerlo nel VLR. Oltre a questo, dopo aver aggiornato il VLR, si aggiornerà anche il puntatore all'HLR.

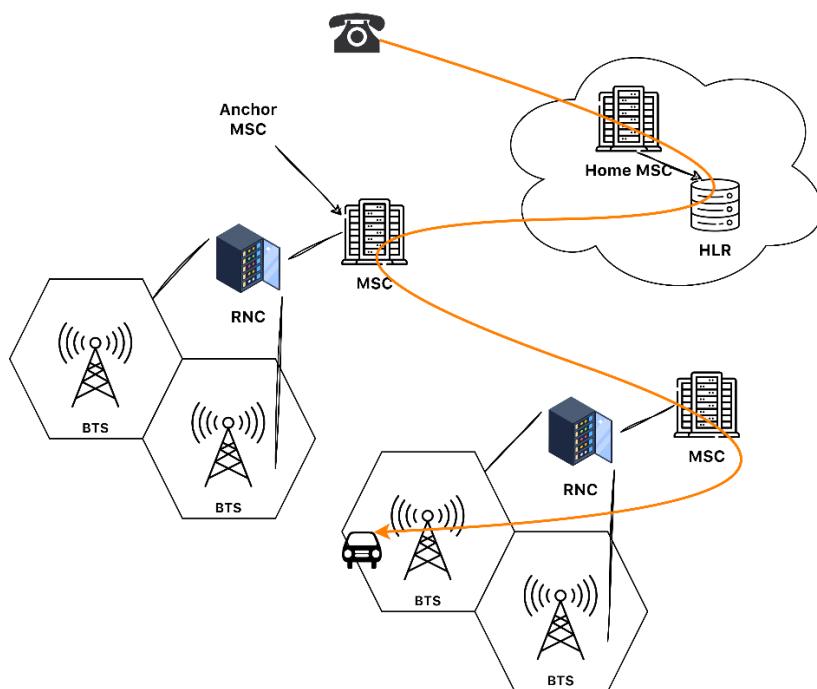
Diverso invece è il discorso di quando si ha una comunicazione in corso, e successivamente si cambia cella (processo di Handoff). Qui lo scopo è trasferire la connessione verso la nuova base station senza interrompere il servizio, quindi in termini rapidissimi e mantenendo le comunicazioni attive. Ci sono vari motivi per cui si fa Handoff, il più banale, è quello in cui un utente si sposta e quindi non sentendo più bene la stazione vecchia deve attaccarsi ad una da cui riceve un segnale più forte. Un altro motivo è invece deciso dalla rete che potrebbe decidere di assegnare una nuova base station ad esempio per cercare di bilanciare il carico della rete (se c'è una cella troppo carica e le altre sono scariche, la rete può decidere segnare quelle più scariche).

In generale la scelta di quando e perché effettuare Handoff è a discrezione dell'operatore telefonico, lo standard definisce la tecnica con cui viene effettuato l'handoff: per capire quando cambiare base station, il dispositivo mobile raccoglie delle misure di qualità del segnale nel tempo ricevute dalla base station (e anche da tutte quelle attorno), le manda alla base station. La base station le raccoglie e sulla base di queste misure può decidere che è arrivato il momento di trasferire l'utente mobile ad una nuova base station. La cosa è resa così veloce, robusta e trasparente, che neanche ci si accorge di ciò che accade.

Per rendere la cosa più veloce, la cella a cui si è collegati sceglie una nuova cella, quella su cui l'utente dovrà andare, e prepara la strada in anticipo (quindi tutto il traffico di segnalazione per l'instaurazione delle risorse) preparando già un nuovo percorso sulla nuova base station, anche se ancora quel percorso non è ancora utilizzato. Una volta messo in piedi, si fanno partire tutte le operazioni di sincronizzazione (per cui a un certo punto l'utente possiede potenzialmente due canali verso due diverse base station, ma sta ancora usando soltanto quella vecchia). Quando è tutto pronto per trasferire la comunicazione su quella nuova, la rete di ordina all'utente di trasferirsi su quella nuova, rilasciando il canale vecchio.

Ci sono in realtà dei casi di handoff più o meno complicati a seconda della cella a cui ci si sposta. Lo scenario appena visto è il caso più semplice, dove un utente si sposta verso una nuova cella che però è servita dallo stesso MSC. Per questo motivo, essendo che si resta sempre sotto il controllo dello stesso MSC, non c'è bisogno di aggiornare il VLR. Il caso in cui spostandosi si finisce sotto il controllo di un altro MSC, è leggermente più complicato. In questo caso se si è in un mezzo veloce (auto, treni), è possibile addirittura attraversare una catena di MSC diversi, creando quindi una situazione che per la rete è molto complicata, in quanto deve instradare la chiamata lungo una catena di MSC

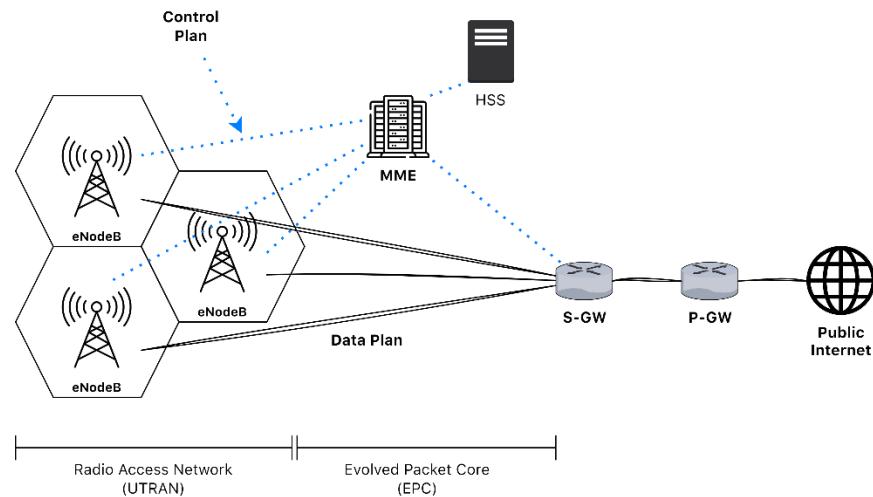
Il caso più complicato in cui l'utente, cambiando cella, finisce sotto il controllo di un'altro MSC è descritta qui:



Quello che si attua in questo caso è il seguente: la prima MSC, quella da cui è partita la comunicazione, fa da "ancora" e quindi assume un ruolo di anchor MSC. Man mano che l'utente si sposta verso delle MSC diverse, vengono aggiunti degli HOP ad una catena di MSC attraversati fino a all'ultimo dove l'utente si trova in un certo momento. Quindi la chiamata viene instradata costruendo questi catena di HOP successivi.

Un miglioramento che si può fare per evitare di creare delle catene lunghissime e di tenere sempre questo concetto di ancora MSC aggiornato ma mantenere solo l'ultimo visitato, tagliando fuori tutti gli altri (che a quel punto sono inutili).

## 4G - LTE



Con il 4G (LTE), si è deciso di fare un cambiamento infrastrutturale pesante, per cui adesso, tutto il traffico sia dati sia voce, transita assieme e dentro una infrastruttura nuova che ha richiesto di introdurre nuovi apparati, con un costo infrastrutturale notevole. Questa nuova infrastruttura fissa si chiama **Evolved Packet Core - EPC** in cui tutto è su IP, voce e dati sono trasportati dentro questo EPC con tecnologia IP. La parte radio continua a essere la UTRAN. Con l'EPC, dato che si trasporta tutti su IP, l'infrastruttura è collegata soltanto più alla public Internet, quindi di fatto la rete LTE non è più attaccata alla rete telefonica tradizionale.

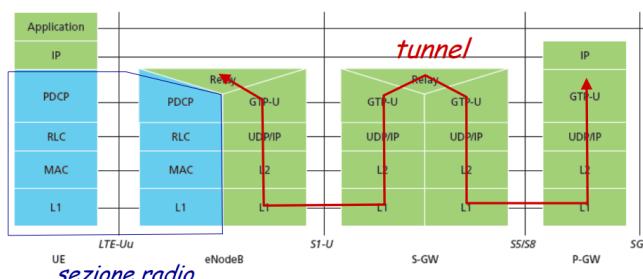
Nel 4G alla base station viene dato un altro nome, che è quello di eNodeB. La parte di EPC si interfaccia con un Serving Sateway (S-GW) che a sua volta si interfaccia con il resto di internete attraverso un Packet Network Gateway (P-GW). Nella rete LTE gli oggetti che sono nel piano di controllo e che gestiscono la mobilità degli utenti sono chiamati HSS (Home subscriber server) e contengono sia HLR che VLR. Le eNodeB sono coleggiate a quello che viene chiamato Mobility Management Entity (MME).

### Qualità del servizio

In che modo si gestisce la qualità del servizio dentro la rete LTE, essendo che a questo punto tutto viene trasferito su IP (EPC)? Ovviamente se ci si affida solo ad IP non si ottiene nessuna qualità del servizio.

Il modo con cui lo si fa è tramite dei tunnel, all'interno dell'EPC, vengono creati dei tunnel tra la base station ed il Gateway.

### Radio+Tunneling: UE – eNodeB – PGW



Una volta che raggiunge il gateway verso l'esterno, esce dal tunnel e viaggia sull'Internet normale. Perché viene fatto tutto questo? In LTE si vuole fornire qualità del servizio diversi in base al traffico. Sono stati concepiti 12 possibili indicatori di qualità del servizio.

<b>QCI</b>	<b>Resource Type</b>	<b>Priority</b>	<b>Packet Delay Budget</b>	<b>Packet Error Loss Rate</b>	<b>Example Services</b>
1	GBR	2	100ms	$10^{-2}$	Conversational Voice
2	GBR	4	150ms	$10^{-3}$	Conversational Video (Live Streaming)
3	GBR	3	50ms	$10^{-3}$	Real Time Gaming, V2X messages
4	GBR	5	300ms	$10^{-6}$	Non-Conversational Video (Buffered Streaming)
65	GBR	0.7	75ms	$10^{-2}$	Mission Critical user plane Push To Talk voice (e.g., MCPTT)
66	GBR	2	100ms	$10^{-2}$	Non-Mission-Critical user plane Push To Talk voice
67	GBR	1.5	100ms	$10^{-3}$	Mission Critical Video user plane
75	GBR	2.5	50ms	$10^{-2}$	V2X (Vehicle-to-everything) messages
5	non-GBR	1	100ms	$10^{-6}$	IMS Signalling
6	non-GBR	6	300ms	$10^{-6}$	Video (Buffered Streaming) TCP-Based (for example, www, email, chat, ftp, p2p and the like)
7	non-GBR	7	100ms	$10^{-3}$	Voice, Video (Live Streaming), Interactive Gaming
8	non-GBR	8	300ms	$10^{-6}$	Video (Buffered Streaming) TCP-Based (for example, www, email, chat, ftp, p2p and the like)
9	non-GBR	9	300ms	$10^{-6}$	Video (Buffered Streaming) TCP-Based (for example, www, email, chat, ftp, p2p and the like). Typically used as default carrier

<b>QCI</b>	<b>Resource Type</b>	<b>Priority</b>	<b>Packet Delay Budget</b>	<b>Packet Error Loss Rate</b>	<b>Example Services</b>
69	non-GBR	0.5	60ms	$10^{-6}$	Mission Critical delay sensitive signalling (e.g., MC-PTT signalling)
70	non-GBR	5.5	200ms	$10^{-6}$	Mission Critical Data (e.g. example services are the same as QCI 6/8/9)
79	non-GBR	6.5	50ms	$10^{-2}$	V2X messages
80	non-GBR	6.8	10ms	$10^{-6}$	Low latency eMBB applications (TCP/UDP-based); Augmented Reality
82	GBR	1.9	10ms	$10^{-4}$	Discrete Automation (small packets)
83	GBR	2.2	10ms	$10^{-4}$	Discrete Automation (big packets)
84	GBR	2.4	30ms	$10^{-5}$	Intelligent Transport Systems
85	GBR	2.1	5ms	$10^{-5}$	Electricity Distribution- high voltage

GBR: min e max bit rate garantiti.

Ci sono dei tipi di traffico più pregiati e altri meno pregiati. Un valore più basso di priorità, indica una classe più pregiata, una priorità più alta.

Ognuna di queste 12 classi ha un'un'indicazione di budget e di delay, quindi un ritardo in millisecondi, massimo introdotto ed un packet error rate. Da notare che tutti questi valori sono in realtà un po' fittizi, nel senso che sono delle garanzie di qualità del servizio nel momento in cui si transita nella rete cellulare, poi di quello che succede fuori della rete cellulare nel resto di Internet non dà nessuna garanzia. La scelta delle priorità è a discrezione dell'operatore.

Comunque ormai quasi tutti gli operatori sono migrati a Voice over IP abbandonato la modalità commutata vecchia e quindi le telefonate di qualunque tipo viaggiano tutte sui IP, e quando non ci sono i requisiti, quando non ci sono le risorse infrastrutturali o banda sufficiente per garantire una qualità decente al Voip, si fa backward compatibilità e quindi si utilizza la tecnica standard a commutazione di circuito.

## **Segnalazione su Internet**

Dentro Internet la filosofia che è stata usata per fare segnalazione è di tipo soft state, quindi diversa dall'hard state che abbiamo visto invece più pertinente al mondo della telefonia. In Internet esiste un piano di controllo ed è quello degli algoritmi di instradamento, quindi dei protocolli di instradamento che possono essere visti come quelli che agiscono sul piano di controllo di Internet per costruire le tabelle instradamento dei router.

All'inizio di internet, erano stati standardizzati due protocolli uno di tipo Distance Vector e uno di tipo Link state. All'inizio piaceva l'idea del distance Vector, quindi quegli algoritmi di che usano Bellman-Ford per convergere verso tabelle di cammini minimi, quindi un algoritmo totalmente distribuito che funziona in modo sorprendente, in un certo senso, basandosi solo su scambi locali di informazioni di instradamento.

Quindi le stesse tabelle che si ottengono in modo centralizzato con dijkstra si ottengono piuttosto sorprendentemente con l'algoritmo di Bellman Ford che è la base teorica dell'instradamento distance vector. La cosa funziona, bisogna solo aspettare un po, cioè, questi algoritmi convergono lentamente e possono anche dare dei problemi di oscillazioni e inconsistenze. Anche se crea un'infrastruttura assoluta, robusta ai guasti, completamente distribuita, cioè il meglio che si può mai pensare, però soffre un po di questa lentezza di convergenza, per cui la preferenza è stata data agli algoritmi di tipo Link state e quindi OSPF, perché sono più affidabili, una volta fatto girare dijkstra si ottengono direttamente i cammini minimi senza aspettare convergenze e possibili oscillazioni degli algoritmi distance Vector.

### **Inter-Domain routing**

Questi servono per l'istramento intra-domain (OSPF), quindi dentro l'autonomous system sotto la gestione di un operatore. Per mettere assieme poi tutti gli Autonomous System e far funzionare il routing su scala planetaria bisogna anche integrare un algoritmo Inter-domain (ce ne solo uno) ci si è messi tutti d'accordo nel mondo delle reti di usare BGP (Border gateway Protocol) che è un protocollo purtroppo molto complicato. È un protocollo che nasce attorno agli inizi degli anni 80 e che è passato attraverso tre versioni successive. È complesso in quanto possiede un sacco di opzioni, di parametri di configurazione che sono lasciate a discrezione degli amministratori degli Autonomous System.

Per funzionare, questi protocolli di routing si scambiano messaggi, usando una forma di segnalazione sul piano di controllo, che viaggiano assieme ai dati stessi. Vedremo proprio tra un attimo che questo ha generato nella storia dei problemi, cioè il fatto che dati e segnalazione viaggiano insieme su gli stessi, sulla stessa topologia, sugli stessi link.

È un protocollo basato su Distance Vector. Non è proprio un protocollo distance Vector come nei protocolli intra domain, ad esempio in RIP, dove si annunciano ai router vicini delle informazioni di raggiungibilità di certe indirizzi di destinazione con un certo costo, ma la cosa è più complessa. Con BGP ciò che si scambiano i router sono degli interi vettori di percorsi, quindi delle sequenze dei Path di Autonomous System che i router usano per raggiungere certe destinazioni.

### **Problemi di BGP**

Il fatto è che (per motivi politici) non viene proprio usato per trovare cammini minimi in Internet, ma cammini che soddisfano certi accordi che i service provider prendono tra di loro per smistare il traffico. Questo ha dato luogo a un sacco di problemi.

Ad esempio, alla fine degli anni 90, delle quantità enormi di traffico generato da BGP (allora BGPS), iniziarono a scambiarsi messaggi (usando TCP) per il traffico di segnalazione. Quello che successe è che si osservarono volumi di traffico molto superiori al previsto, perché i router avevano dei problemi a convergere continuando a scambiarsi i dati senza raggiungere un punto di equilibrio, quindi gran parte di questo traffico era ridondante e successe anche una cosa molto spiacente, cioè il traffico di segnalazione, mescolandosi con il dati di traffico normali, si creò una congestione di dati più segnalazione che portava anche a problemi di oscillazione. Quindi traffico che prima veniva mandato da una parte, poi troppa congestione, e veniva spostato da un'altra parte e così via.

Inoltre molti dei problemi di BGP sono dovuti al fatto di tutta una serie di parametri che vengono talvolta mal configurati, e che gli amministratori locali non sanno bene come impostare, che causano effetti di convergenza globali. Si sono osservati anche degli effetti piuttosto patologici di Black Hole.

### **BGP Insegna**

Si sono imparate delle lezioni da BGP:

- il primo è che la convergenza del routing, la consistenza delle tabelle di routing in un sistema distribuito è complicato da mantenere che a volte sfugge alla capacità di comprensione umana.
- La seconda è che gli errori di configurazione si propagano, quindi è sufficiente che un singolo router si comporti in modo anomalo da generare effetti su scala molto più ampia.
- La terza è che i pacchetti di controllo e pacchetti dati possono arrivare a interferire tra loro generando volumi di traffico complessivo anomalo.

## **Soft State & Hard State**

Il problema è quello di mantenere dello stato nei nodi di una rete in forma distribuita e farlo nel modo più possibile flessibile e tollerante a guasti possibili.

Cosa si intende per mantenere stato nella rete? Per stato si intende una generica informazione che viene memorizzata nei nodi di una rete da parte di un qualche protocollo di rete. Questa informazione di Stato è conservata in più di un nodo, quindi in forma distribuita, e può essere variabile nel tempo, quindi, mutare quando le condizioni della rete mutano: è un'informazione che viene aggiornata nel tempo. Tipicamente l'informazione di stato è associata a chiamate o sessioni generate dai sistemi terminali.

Ad esempio, nel mondo ATM è legato al mantenimento dei circuiti virtuali, quindi tutti i commutatori avevano una tabella di inoltro delle celle. Anche TCP come protocollo gestisce dello stato nei sistemi terminali, quindi in TCP c'è un concetto di stato legato alla connessione in corso dove però lo stato è presente solo alla sorgente e alla destinazione e che consiste nei numeri di sequenza oltre ai timer che gestiscono i problemi dei pacchetti persi.

Parleremo di un mittente e di un ricevitore riferendoci ad oggetti nel piano di controllo, da non confondere con mittente e ricevitore di una connessione dati normale.

### **Hard State**

Nella modalità hard state lo stato viene installato nel ricevitore alla ricezione di un messaggio di setup, mandato dal mittente. Similmente lo stato viene rimosso dal ricevitore quando si riceve un messaggio di tear down, di rimozione dello stato, sempre inviato dal mittente.

L'assunzione di default è che lo stato è valido a meno di comunicazione esplicita: l'ipotesi in hard state è che, in assenza di messaggi di setup / tear down, lo stato attualmente presente è valido.

Già parlato del fatto che una debolezza di questo approccio è legato al fatto che malfunzionamenti nella rete, nodi che vanno in crash e link che si guastano che portano al cosiddetto Stato Orfano: delle informazioni di stato pendenti nella rete, che nessuno è più in grado di rimuovere.

Nell'approccio soft state invece si vuole proprio evitare il problema degli stati orfani.

### **Soft State**

Per soft state si intende una modalità di mantenere un'informazione di Stato nei nodi della rete che è gestita dagli host, dai sistemi terminali.

L'idea proviene da Clark che aveva intuito questo concetto, appunto, coniando il termine di soft state.

Secondo Clark, per soft state, si intende un sistema in cui avvengono determinate cose:

- Lo stato è mantenuto probabilisticamente: Questo si contrappone alla modalità hard state in cui invece lo stato è mantenuto con certezza.
- Lo stato non è necessariamente consistente: ci possono anche essere delle situazioni in cui lo stato non è quello che dovrebbe essere, quindi una frazione di tempo in cui lo stato è diverso da quello che vogliamo raggiungere.
- Gli utenti si assumono la piena responsabilità di mantenere lo Stato: i sistemi terminali hanno il compito di gestire la creazione e il mantenimento dello stato nella rete
- La rete, invece, ha la responsabilità di rimuovere lo stato.

L'idea del soft state è la seguente: nel soft state c'è sempre il messaggio di setup, quindi è sempre vero che lo stato è installato dentro il ricevitore in risposta ad un messaggio di setup inviato dal mittente, ma il mittente non manda semplicemente un singolo messaggio di setup. Successivamente al primo messaggio di setup, inoltra una serie di messaggi di refresh periodici. Quindi per tutta la durata in cui il mittente vuole che lo stato sia mantenuto nel ricevitore, a intervalli periodici, manda dei messaggi di refresh al ricevitore che hanno significato semantico di mantenere lo stato (eventualmente aggiornarlo). Quando il ricevitore non sente più refresh per un certo tempo (il ricevitore mantiene un time out, che viene refreshato ad ogni messaggio di refresh che riceve), elimina il suo Stato. Quindi in soft state, l'assunzione di default e che lo stato è sempre valido, a meno che non venga rinfrescato.

Quindi è un approccio di tipo best effort che non garantisce in modo deterministico che lo stato venga installato nella fase di setup. Per questo motivo si parla del fatto che lo stato è installato in modo probabilistico, perché il messaggio di setup iniziale potrebbe anche venire perso con una qualche probabilità e di conseguenza posso anche non avere la garanzia assoluta che lo stato sia stato installato nel ricevitore nella fase di Setup.

Questa è la modalità soft state cosiddetta pura, che non fa mai uso di messaggi esplicativi di rimozione dello stato. Nella pratica si può arricchire questa idea del soft state con una rimozione esplicita: invece che aspettare che scada il time out nel ricevitore, il mittente può rimuovere direttamente in modo esplicito lo stato con un messaggio di tear down che si può aggiungere, volendo, al protocollo.

L'idea del soft state è l'idea che è piaciuta di più nel mondo Internet, che porta ad un'intrinseca robustezza e flessibilità dei sistemi nei confronti di errori di vario tipo.

Quindi con questo meccanismo si risolve il problema degli Stati orfani grazie ai timer che lo rimuovono in automatico.

Nel corso del tempo ci si è resi conto che il soft state, anche se ha dei difetti (gestione probabilistica dello stato, possibilmente inconsistente con frazioni di tempo in cui lo stato non è quello che dovrebbe essere ed il refresh periodico che non è gratis, quindi l'invio di messaggi di refresh periodici ha un costo in termini di banda), permette comunque di costruire dei sistemi intrinsecamente più robusti e adattativi ai cambiamenti nelle condizioni sottostanti della rete, senza bisogno di meccanismi esterni al protocollo che gestiscono gli errori.

### Segnalazione su IP

In Internet non è previsto alcun protocollo di segnalazione: è basato su un servizio besteffort, non orientato alla connessione dove non c'è bisogno di mantenere nessuno stato.

Un tentativo però è stato fatto (di creare stato nei nodi interni) ed è quello di fornire qualità del servizio alle applicazioni multimediali, andando a prenotare banda sui link con il **Resource Reservation Protocol – RSVP** ma poi non fu utilizzato nella pratica. Quando si è provato a fare qualità del servizio in Internet con RSVP, lo si è provato a fare anche nella modalità multicast, dato che in molti casi è necessario avere trasmissioni punto multipunto nelle trasmissioni multimediali.

### Multicast su IP

Si è provato a fare multicast IP, ma è fallito miseramente, ma oltre ad esperimenti di laboratorio su piccola scala (anche su IPv6), non c'è più nessuno che pensa di utilizzare il multicast a livello tre. Anche perché nel frattempo sono arrivate le Content Delivery network, quindi soluzioni multicast a livello più alto, applicativo, che hanno decretato un po la fine dell'idea di fare il multicast su IP.

Per il multicast era stata riservata un'intera porzione dello spazio di indirizzamento Ipv4. All'incirca 256 milioni di indirizzi che non sono pochi, però c'è stato subito problemi di natura politica, cioè come li dividiamo? Chi li gestisce? Chi ha diritto a prendersene uno?

Poi sono stati sviluppati tutta una serie di protocolli per farlo funzionare. Tipo IGMP che serviva proprio alla gestione del concetto di gruppo Multicast nei ricevitori. A tutto questo c'era la necessità di nuovi protocolli di routing che avessero la capacità di gestire multicast, quindi andavano modificati gli algoritmi precedenti che per la natura erano punto punto, in modo da rendere possibile realizzare strade punto multipunto.

L'idea che era stata proposta di internet multicast era la seguente: le sorgenti, i mittenti che volevano mandare in multicast dei pacchetti, semplicemente generavano dei pacchetti mettendo come indirizzo di destinazione l'indirizzo multicast, dopodiché avevano finito il loro compito. Quindi il ruolo delle sorgenti era estremamente semplificato dal fatto che c'era questa astrazione nel gruppo Multicast (IP Multicast), per cui le sorgenti tutto quello che facevano erano di generare un pacchetto con un indirizzo multicast. Viceversa, i ricevitori che interessati a ricevere pacchetti di un certo gruppo multicast, dovevano, dentro la loro rete locale, iscriversi ai gruppi multicast, quindi dovevano essere in una rete locale con un gateway in grado di supportare il servizio multicast, altrimenti non era possibile il tutto. Quindi i ricevitori, iscrivendosi dichiaravano il loro interesse a ricevere pacchetti di un certo multicast, e lo facevano tramite il protocollo IGMP che serviva proprio a gestire nelle reti locali l'iscrizione dei ricevitori ai gruppi multicast.

I router di bordo che avevano sotto di loro almeno un utente interessato a ricevere un gruppo Multicast, dovevano poi entrare a far parte di un percorso. Tutti i router di bordo dovevano poi coordinarsi tra di loro, in modo da creare un albero di distribuzione che, a partire dai mittenti, andasse a consegnare tutti i pacchetti di quel gruppo multicast a tutti i router che avevano almeno un utente interessato a ricevere quel pacchetto.

Quindi la cosa, avveniva in due fasi: una fase locale in cui si informavano i multicast gateway locali di essere interessati a far parte del gruppo multicat, usando IGMP e poi bisognava, su scala geografica, coordinare tutti i router per mettere in piedi l'istradamento punto multipunto su cui sono state fatte tante proposte di algoritmi di Istradamento Multicast.

IGMP

Quindi in una rete locale, viaggiavano due tipo di pacchetti che permettevano di supportare questa appartenenza ai gruppi multicast: una sono i messaggi IGMP Report, che venivano mandati dagli host quando volevano entrare a far parte di un gruppo Multicast, informando il loro router che sono interessati a quel gruppo multicast. E poi invece i messaggi IGMP Query che invece vengono mandati dai router a intervalli regolari per chiedere agli host di ribadire il loro interesse a far parte del gruppo (equivalente di soft state) se il non veniva sentita nessuna risposta in automatico il Router rimuoveva quell'host dalla lista degli host interessati al gruppo multicast.

## Routing Multicast

A livello di routing invece, la cosa era estremamente complessa. Mentre nel caso di comunicazioni punto punto è tutto concettualmente risolto in modo brillante dall'algoritmo di Dijkstra, di complessità polinomiale, quando si devono trovare delle strade punto multipunto, il tutto si complica, tanto da definire il problema un tipo NP.

## Spettro dei meccanismi di segnalazione

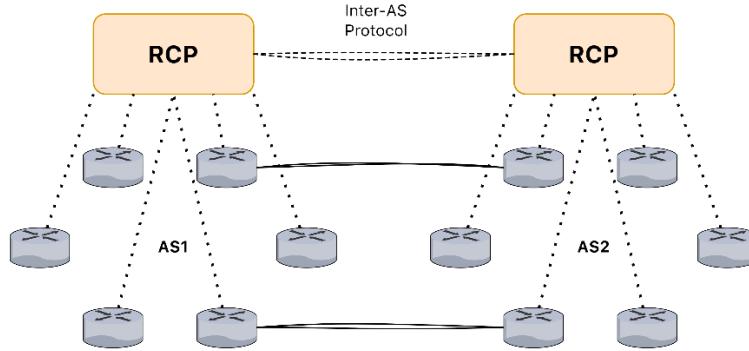


Non esiste soltanto il soft state e l'hard state, ma esistono anche delle versioni intermedie, che sono quelle che si ottengono quando si parte dall'approccio soft state e si aggiunge ad esempio, la rimozione esplicita dello stato. Quindi, non ci si affida al time out del ricevitore per rimuovere lo stato, ma si rimuove esplicitamente con il tear down esplicito (IGMPv2/v3) Oppure chiedo un riscontro, ack esplicito al messaggio iniziale di setup dello stato.

# Software Defined Networking

L'idea della software defined networking è quella di permettere agli amministratori di una rete di gestire le cose in modo flessibile ed efficiente.

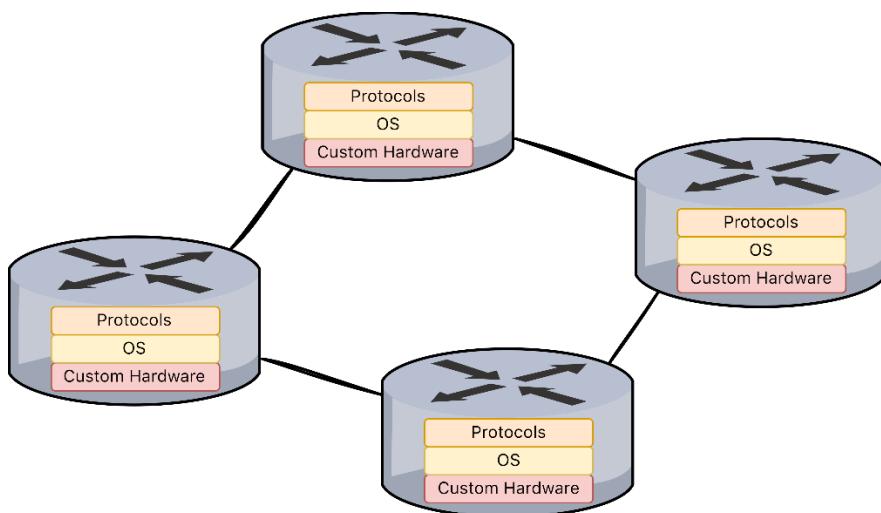
L'idea che è alla base del software defined networking è quella di avere un controllore di rete che controlla in modo logicamente centralizzato l'instradamento, all'interno di un dominio.



Questa idea era nata già nel 1970 con SNA di IBM, e l'idea era appunto di avere dei server di controllo (**Router Control Platform – RCP**) dove veniva calcolato in modo centralizzato l'instradamento per tutto un autonomous system. Quindi, invece di calcolare le strade in modo distribuito, lasciandolo fare ai router, c'era un unico punto dove veniva raccolta l'informazione della topologia della rete e veniva calcolato l'instradamento in modo arbitrario (non doveva necessariamente essere che il metodo dei cammini minimi).

Dopodiché, queste informazioni calcolate in modo logicamente centralizzato, venivano distribuite ed insegnate ai vari router del dominio.

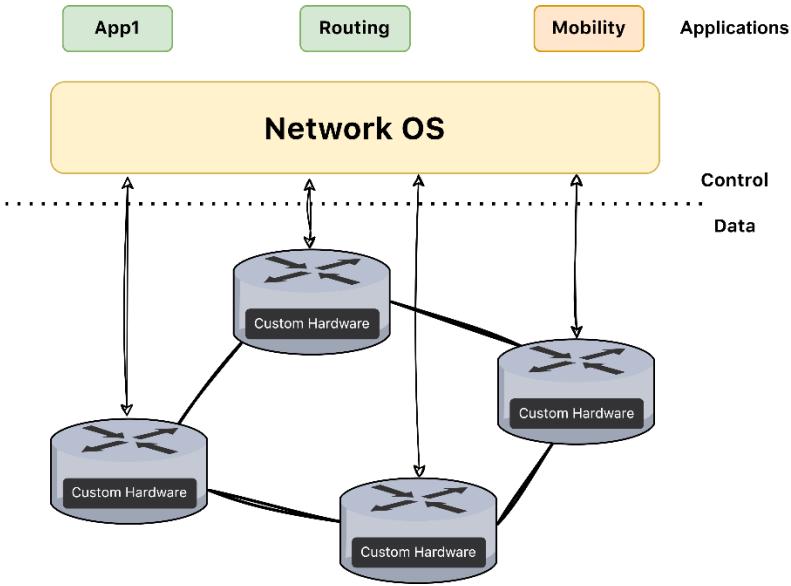
Particolare successo ha avuto l'architettura OpenFlow, proposta da un gruppo di Stanford. L'idea che hanno venduto con successo al resto del mondo è quella di creare un piano di controllo della rete il più possibile aperto e svincolato dall'hardware.



Nella visione tradizionale i router sono in sostanza oggetti complicati dove viene integrato una parte hardware che gestisce l'inoltro/commutazione dei pacchetti, un sistema operativo piuttosto

complicato (proprietario del costruttore) che li gestisce ed un'insieme di protocolli e di applicazioni che girano su di essi.

In SDN il concetto è quello di separare questi strati, separando le componenti, tirando fuori una sorta di sistema operativo di rete che collega tutto l'hardware sottostante. I router diventano quindi solo oggetti che fanno hardware, fanno smistamento di pacchetti ma non hanno più nessuna intelligenza sopra, non hanno più un sistema operativo e non hanno più dei protocolli che ci girano sopra. I router sono diventati soltanto dei commutatori.



Il piano di controllo che viene separato dai router, viene centralizzato logicamente inserendolo dentro un controllore di rete, che gestisce questa volta in modo software il controllo della rete, tramite un sistema operativo di rete. Nella visione più futuristica dell'SDN, la parte di controllo può anche essere ospitata in un servizio cloud esterno alla rete, in modo da renderlo anche robusto ai guasti e ridondato (essendo un punto vulnerabile della rete, single point of failure). Si paga un po in termini di latenza, ma si guadagna molto in robustezza, affidabilità.

Con SDN c'è quindi un ripensamento della divisione dei compiti nella rete con una netta separazione tra piano dati e piano di controllo. Il piano di controllo viene astratto dalla rete e logicamente centralizzato in modo software dentro il controllore. Dopodiché quello di cui si ha bisogno è semplicemente un protocollo che faccia parlare il controllore con i router (avere un'interfaccia aperta verso il piano dati). Quindi rendere il tutto il più possibile programmabile e personalizzabile e adattabile ai bisogni dei singoli amministratori di rete con degli standard open source totalmente aperti che permetta la personalizzazione completa.

Quindi l'SDN è una cosa pensata non tanto per gli utenti ma per amministratori di rete. E anche per ridurre i costi di gestione, ridurre i costi fisici dell'infrastruttura, in quanto i commutatori sono diventati molto più semplici di prima, quindi si ha una riduzione dei costi dell'hardware.

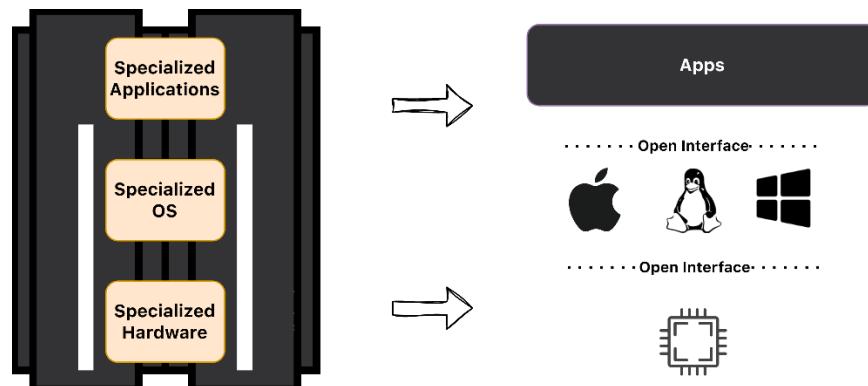
Oltre all'applicazione del Routing ci sono altre applicazioni sfruttabili (applicazioni sul piano di controllo), possibilità di gestire la mobilità degli utente, traffic engineering, avendo questa flessibilità nella scelta delle strade è possibile anche risolvere in modo programmabile il problema di come gestire i flussi dentro la rete e quindi si è arricchito anche il ventaglio delle applicazioni possibili.

Ma come si mettono in comunicazioni i vari livelli? È qui che entra in gioco Openflow. Quindi Openflow è un protocollo che permette la comunicazione tra gli switch ed il controllore.

## OPENFLOW

Openflow è una possibile realizzazione dell'idea di SDN, non è l'unica esistente, infatti c'è ne sono delle alternative, però indubbiamente è quella di cui si sente parlare più spesso. Non confondere SDN con Openflow: Openflow è una particolare implementazione dell'idea di SDN, quella che ha avuto più successo. E non è nient'altro che un formato di messaggi che fa parlare gli switch con il controllore.

Per la realizzazione di Openflow bisogna cominciare con l'astrazione dello switch. L'altro pezzo è il controllore SDN che è quello in cui viene logicamente centralizzata l'elaborazione dei flussi. Anche per la realizzazione software del controllore ci sono numerose soluzioni possibili che sono essenzialmente diverse tra di loro per l'uso del linguaggio di programmazione e per le prestazioni che ottengono. Ci sono controllori scritti interamente in C, ci sono quelli scritti in Python, in altri linguaggi di programmazione, però fanno tutti più o meno lo stesso mestiere, mettono a disposizione delle librerie delle api e si possono facilmente utilizzare, per mettere in piedi la rete SDN nel linguaggio che si preferisce.



Chi ha promosso l'idea di SDN, ha voluto fare questa analogia con l'evoluzione dei computer, facendo notare che è successa una cosa simile nel mondo dei sistemi di calcolo. All'inizio, prima che nascesse il personal computer, c'erano dei grossi sistemi di calcolo che si chiamavano mainframe, che erano degli oggetti monolitici di proprietà di pochissimi costruttori (tipo quelli IBM) dove era integrato assieme all'hardware, un sistema operativo e delle applicazioni tutte però proprietario. Alla nascita del personal computer, diciamo anni 80, si sono divisi gli strati, separati in dei processori semplici, generali e controllati da un sistema operativo di cui sono cominciate a nascere alternative (Windows, MacOS, Unix). E infine, a livello più alto, le applicazioni utenti. Aver separato le componenti, aver creato delle interfacce aperte tra questi strati ha avuto enormi conseguenze, sia in termini di costo sia in termini di innovazione e diversificazione degli attori in gioco.

Similmente nel mondo dei dispositivi di rete, il Router è un oggetto molto complesso, avente hardware specializzato, un control plan specializzato e delle applicazioni proprietarie. Il tutto monolicamente integrato all'interno. Con SDN invece si fa il mestiere simile di separazione delle funzionalità dei personal computer, quindi si separano a livello più basso, solo switch che fanno forwarding dei pacchetti e basta. Un'interfaccia aperta verso il Control Plan e un'interfaccia aperta verso le applicazioni del piano di controllo

Con Openflow si è stata presa una decisione abbastanza importante, cioè la granularità del controllo, quindi l'oggetto minimo che siamo in grado di controllare è un flusso. Openflow ha adottato questa idea di non fare granularità a livello di singoli pacchetti (perché appariva troppo fine, troppo complicato) ma appunto di aggregati.

Il flusso però lo si è visto nel senso più generale possibile, quindi basato su tutti gli header dal livello più basso fino al livello quattro. Quindi, quando in openflow bisogna prendere una decisione su come trattare un flusso di pacchetti, il tutto viene fatto guardando tutti gli header fino al quarto. Con questa astrazione del flusso, si è perso la possibilità di fare:

- Per packet processing, quindi di trattare in modo diverso i pacchetti che appartengono allo stesso flusso.
- Deep packet inspection, quindi non si guarda oltre l'intestazione di livello quattro.

## Switch in Openflow

L'astrazione di switch in Openflow consiste in tre componenti:

1. **Una tabella di flussi (flowable)**: definisce i flussi riconosciuti da quello switch, che cosa farne e qual è l'azione da intraprendere su quel flusso. Nel caso più semplice, inoltro di un pacchetto sulla porta di uscita, scartare un pacchetto appartenente ad un flusso. In particolare, avendo a disposizione tutti questi Header, è possibile installare delle tabelle di flussi che fanno operare uno switch come se fosse un router, un firewall, un nat o un load balancer
2. **Un canale di comunicazione sicuro verso il controllore**: gli switch devono dialogare con il controllore e farlo in modo sicuro e robusto. In particolare le tabelle di forwarding devono essere mandate in modo sicuro verso gli switch
3. **Openflow protocol**: L'aggiunta e la modifica delle tabelle di inoltre avviene appunto tramite protocollo Openflow.

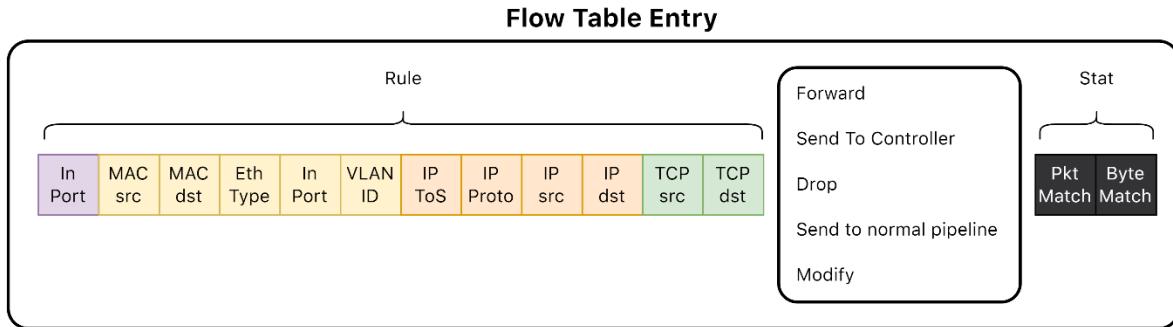
## Astrazione del piano dati

Quindi gli switch hanno delle regole per riconoscere i pacchetti come appartenenti ad un certo flusso, facendo matching su uno o più campi. Successivamente lo switch può eseguire diverse azioni su quel flusso:

- **Scarto** (default)
- **Inoltro su una porta** (anche più di una, quindi replicarlo)
- **Invio sullo stack protocollare standard**
- **Modifica** (tipo TTL, NAT)
- **Inoltro al controllore**

Queste azioni vengono prese secondo delle priorità (come nelle ACL). È possibile siano presenti più match entry e quindi bisogna poi scegliere quale di queste entry usare, quindi definire una priorità tra le entry.

Poi associato ad ogni flusso c'è anche una serie di contatori (per ogni entry) per effettuare delle misure: quindi per ogni flusso si tiene conto del numero di pacchetti e per maggiori informazioni anche il numero di byte che hanno matchato quel flusso e quindi si ha anche una misura del traffico che è stato gestito con quella entry.



Quindi le entry della flow table sono organizzate in questo modo (immagine), ogni riga ha tre macroparti:

- La parte della regola (Rule), su cui viene fatto il match
- La parte di azione sul flusso (Action), quindi forward, drop, etc
- La parte di statistiche, quindi il numero di pacchetti e il numero di byte che hanno matchato.

Cosa si guadagna con questa astrazione? La possibilità di caratterizzare e definire dei comportamenti di una generica middlebox tramite match + azione. Mestiere del router, il mestiere dello switch. È anche possibile fare multicast, che invece in IP è stato abolito, grazie alla replicazione di pacchetti su più porte d'uscita. Il firewall che prende tutto quello che vuole dei vari campi fino al livello quattro e potrebbe banalmente fare bloccare un flusso, o inoltrarlo. Il NAT, guarda gli indirizzi IP e le porte e fa nulla di più rispetto a prima.

### Azione sui pacchetti

Abbiamo visto che bisogna specificare un'azione da fare sui pacchetti, di default viene scartato. Il Forwarding (inoltro) può essere effettuato:

- **ANY:** in Openflow significa inoltro di il pacchetto su tutte le interfacce di uscita, esclusa quella di arrivo.
- **IMPORT:** inoltro sull'interfaccia di arrivo. Tipicamente non si rimanda mai indietro
- **CONTROLLER:** quando il pacchetto viene inoltrato al controllore, incapsulato dentro un messaggio Openflow.
- **TABLE:** Il pacchetto segue l'azione specificata dalla flow table
- **LOCAL:** Invio al networking stack locale

### Controller

Il controllore deve reagire a degli eventi che provengono dagli switch e in risposta a questi eventi, impartire dei comandi agli switch. Gli eventi possibili che possono arrivare dagli switch possono essere:

- **Topology Change event:** se avviene un cambiamento nella topologia con una rottura o aggiunta di uno switch, il controllore, che deve avere una visione il più possibile aggiornata della topologia, deve essere informato. Quindi ci sarà un messaggio che lo switch manda al controllore di Topology Change.
- **Traffic Statistic event:** Dato che lo switch mantiene anche le statistiche per ogni flusso, se interrogato dal controllore (query statistic), dovrà mandare delle misure di traffico. È una cosa importante anche per accorgersi di anomalie e attacchi in corso.
- **Packet arriving event:** può direttamente arrivare da uno switch, un pacchetto del piano dati che lo switch affoga al controllore.

Viceversa, i comandi che il controllore manda verso gli switch, in risposta agli eventi possono essere:

- **Install/Uninstall rules:** installa o disinstalla regola/e di inoltro
- **Query statistic:** fare interrogazioni di statistiche del traffico
- **Send Packets:** In risposta ad un evento di packet arriving, il controller può rispedire indietro il pacchetto allo switch, indicandogli cosa fare (mandarlo su una porta specifica, etc)

Openflow comunica con gli switch usando TCP, a volte cifrando il tutto.

### **Controllo di accesso dinamico**

È interessante riflettere sulle flessibilità che ci dà un approccio di questo genere, ad esempio, per fare un controllo di accesso dinamico degli utenti: se un nuovo utente apre un nuovo flusso, il primo pacchetto che arriva al primo switch, quello switch non saprà cosa farsene, lo spedisce al controllore (incapsulato in openflow). Il controllore che avrà delle policy automatiche per gestire questa situazione (quindi analizza il pacchetto, e può decidere di ammetterlo o meno), se lo ammette, potrebbe inoltrare agli switch nella rete le strade che permettono di smistare il traffico generato appartenente a quel flusso nuovo. Quindi essendo un flusso sconosciuto, se non si fa niente verrebbe tutto scartato. Una volta installate le regole su tutti gli switch, poi tutti i pacchetti faranno match su questa regola, quindi non passeranno più al controllore, ma faranno forward.

Se il controllore decide di bloccare quel flusso, lo rimanda indietro e indica di scartarlo (per questo è un'applicazione per il controllo di accesso dinamico: è possibile farlo flusso per flusso, utente per utente).

### **Mobilità trasparente**

Un'altra applicazione interessante è la gestione della mobilità. Se si ha un flusso in corso e l'utente si sposta, collegandosi in un altro punto della rete, di nuovo con l'idea che il primo pacchetto fa parte di un flusso sconosciuto, verrà mandato al controllore. Questa volta il controllore deve istruire gli switch per disinstallare le regole della strada che aveva impostato precedentemente (non servono più, l'utente si è spostato, i pacchetti faranno percorsi diversi) ed installare le regole nuove sugli switch opportuni.

La cosa è del tutto trasparente per gli host, ed il tutto viene gestito in modo automatico, sul piano SDN.

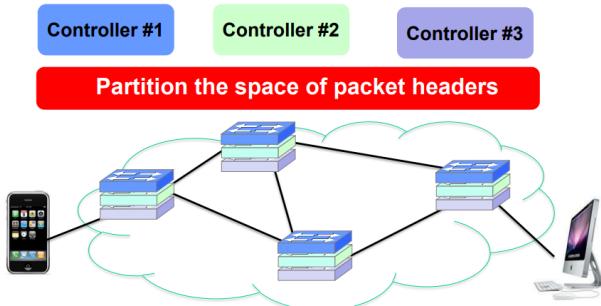
### **Bilanciamento del carico**

Bilanciamento del carico: l'obbiettivo è ad esempio di smistare il traffico proveniente da più flussi che arrivano ad uno switch, e ripartirli su più strade, in modo da bilanciare il carico verso i server che quei flussi vogliono raggiungere. Con SDN è sufficiente andare a installare delle regole di inoltro che, ad esempio, dividono il traffico sulla base dell'indirizzo IP sorgente. Quindi tutte le connessioni che hanno un certo range di indirizzi IP vengono inoltrati verso una porta, un altro range su un'altra porta.

Questo metodo è il più semplice ma non quello più efficiente. È anche possibile usare algoritmi di hash sui flussi: per ogni flusso si prendono i campi tipici di un flusso che sono indirizzo IP, sorgente, destinazione, porta sorgente, porta destinazione, e li do in pasto ad un hash function, ricavando un digest numerico che si mette in modulo numero dei server.

## Esempio: Virtualizzazione della rete

- ❖ si divide lo spazio degli indirizzi in tre partizioni
- ❖ ogni partizione gestita da un diverso controllore



### **Virtualizzazione della rete**

L'idea di virtualizzare riguarda il fatto che la rete SDN è stata spinta fino al livello di immaginare la presenza di più controllori, che controllano la stessa infrastruttura fisica.

L'unica cosa che bisogna fare con molta attenzione quando ci si mette in un contesto di questo tipo è che non avvengano conflitti e cioè che poi questi controllori non si diano fastidio tra di loro: bisogna partizionare lo spazio degli header in modo chiaro, in modo da chiarire chi si occupa di un range e chi di altri range.

Però, che cosa c'è di bello in tutto questo? Che una stessa infrastruttura fisica potrebbe essere gestita da più Operatori di rete virtuali. Quindi tutti usano alla fine le stesse infrastrutture però sono gestori di telefonia diversi. È una cosa che va molto di moda attualmente si chiamano virtual mobile network operator, cioè operatori di rete cellulari virtuali, che cioè non possiedono fisicamente gli strati dell'infrastruttura, però ne controllano una certa porzione del traffico che ci scorre dentro.

Allora se si divide lo spazio degli indirizzi in partizioni precise e distinte e se ogni partizione viene gestita da un diverso controllore, tutto è ok. Se invece le cose sono sovrapposte, può succedere di tutto.

### **Messaggi controller-switch**

- **Features:** Il controllore può interrogare lo switch in merito al suo stato, o in merito alle sue caratteristiche, quindi quanti link ha, che velocità, che interfacce ha, e lo switch gli risponde.
- **Configure:** il controllore, dopo aver tipicamente interrogato, setta dei parametri di configurazione dello switch.
- **Modify-state:** riguarda nello specifico l'aggiunta e la cancellazione di entry della flow table.
- **Packet out:** il controllore ordina di inviare un pacchetto specifico e di Mandarlo fuori verso una specifica porta dello switch.

### **Messaggi switch-controller**

- **Packet-in:** trasferisce il controllo di un pacchetto specifico verso il controllore.
- **Flow-Removes:** nel caso in cui venga rimossa una entry della flow table che potrebbe anche essere rimossa perché è scaduto un certo time-out, lo switch, informa il controllore, è.
- **Port-status:** cambia lo stato di un'interfaccia

Tutte queste cose non c'è ne dobbiamo occupare noi sono state già implementate in vari linguaggi di alto livello messi a disposizione del sistema operativo di rete.

## **SDN per datacenter**

Un contesto dove è piaciuto molto l'idea di SDN è il contesto dei data center in quanto mostruosamente complicati. In un data center ci sono centinaia di migliaia anche di RACK, dove ci sono centinaia di server che dovranno poi essere collegati tra di loro attraverso una gerarchia di switch. Quindi ci sarà un vincolo legato al fatto che bisogna utilizzare l'instradamento classico, oltre ad esserci una questione di costo: se si hanno 200.000 server che devo gestire con 10.000 switch da 5000\$ l'uno, il costo complesivo sarà di 50 milioni. Se invece si semplifica molto il mestiere dello switch, togliendogli tutti i pezzi di controllo (SDN Approach), costruisco degli switch più economici da 1.000\$, si risparmia 40 milioni di dollari. Quindi questo è il vantaggio in termini economici.

Poi a livello di controllo è chiaro che se si utilizzasse un approccio SDN, si potrebbe utilizzare tutto ciò che SDN offre, quindi bilanciare il traffico, creare nuovi servizi, applicazioni in modo molto più flessibile e che i vari amministratori si possono personalizzare come vogliono

## **Conseguenza per gli standard**

Ovviamente la cosa non è indolore, bisognerebbe effettuare una migrazione verso SDN, il che richiede anche un grosso sforzo di know-how quindi insegnare ai tecnici a usare tutto un diverso modo di fare le cose.

Il ruolo degli standard cambierebbe, quindi, mentre prima con gli RFC gli standard definiscono il comportamento (non solo definiscono un formato di pacchetti, ma anche un modo di fare le cose), il comportamento della rete sarà invece gestito direttamente dagli amministratori, permettendo di introdurre nuove features senza che queste siano definite in un RFC. Quindi la programmazione delle reti non avrà bisogno di essere standardizzata.

## **Sviluppi del networking con SDN**

Che cosa dobbiamo ringraziare di tutto questo? Ancora una volta, come spesso succede nelle informatica, le astrazioni, quindi le separazioni dei livelli, il fatto che ci risparmiano dai dettagli di livello più basso creano le astrazioni, salvano i programmati della complessità, rendono i sistemi più flessibili e ci portano in luoghi non ancora esplorati.

# Randomizzazione

La randomizzazione è una tecnica elegante usata per risolvere in modo efficiente una gran quantità di problemi nell'informatica. Esistono già un'intera famiglia di algoritmi che usando la randomizzazione, che trovano delle soluzioni piuttosto geniali a dei problemi usando la randomizzazione (Ethernet, Wi-Fi, TCP)

## Randomizzazione in CSMA/CD

In ethernet succede che una sorgente comincia a parlare, e dopo un istante di tempo alfa, il segnale raggiunge tutti gli altri host che, sentendo il segnale occupato, smettono di trasmettere.

Quindi ogni host che trasmette un pacchetto, dopo un tempo Alfa, è sicuro che tutti gli altri non gli parleranno sopra, quindi non genereranno una collision. Il problema è che dentro questo tempo Alfa può purtroppo succedere che qualcun altro cominci a parlare e allora inevitabilmente si crei collisione.

Alfa determina quello che viene chiamata la finestra di vulnerabilità del CSMA. È un tempo critico ma piccolo dato che è dovuto solo al tempo che impiega il segnale a propagarsi da un nodo all'altro ed essere sentito dalla scheda del ricevitore.

Si crea ora un modello con queste ipotesi:

- Si suppone che la lunghezza dei pacchetti è fissa.
- Si suppone che il tempo di trasmissione di un pacchetto è di 1 unità di tempo.
- Si suppone di conoscere il carico offerto ( $G$ ): cioè il numero di tentativi di trasmissione che tutte le stazioni generano nell'unità di tempo (è un rate)

Ora supponiamo che questo rate  $G$  di tentativi di trasmissioni generato dall'insieme di tutte le stazioni, generi un processo poisson di parametro  $G*t$ . Quindi, per come è definita la funzione di densità:

$$\mathcal{P}_\lambda(n) = \frac{\lambda^n}{n!} e^{-\lambda}$$

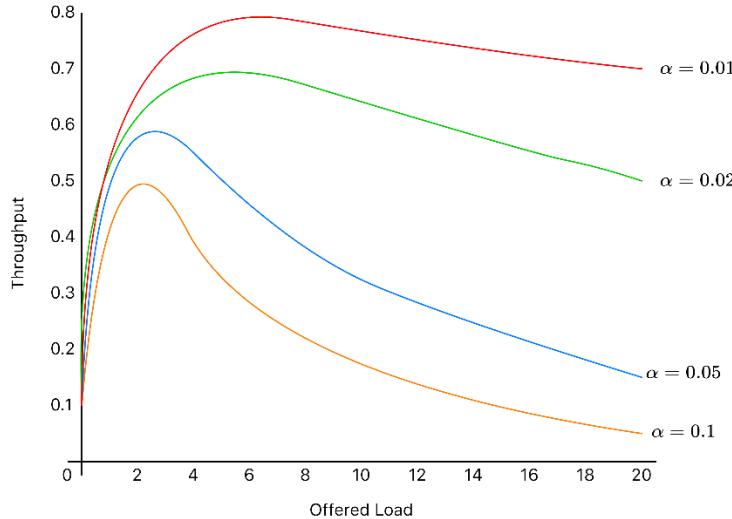
che esprime le probabilità per il numero di eventi ( $n$ ) che si verificano successivamente ed indipendentemente in un dato intervallo di tempo, sapendo che mediamente se ne verifica un numero lambda.

Preso una qualunque finestra di tempo di durata ( $t$ ) arbitrariamente grande, la probabilità di avere ( $k$ ) trasmissioni dentro questo intervallo lungo ( $t$ ):

$$\mathcal{P}_{Gt}(k) = \frac{Gt^k}{k!} e^{-Gt}$$

In questo modo è possibile dire quanti tentativi di trasmissione sono cominciate dentro una finestra  $t$ .

Questo grafico mostra come, dipendentemente come cambia, la finestra di vulnerabilità (alfa), in funzione di  $g$ , la distribuzione ha un andamento non monotono: all'inizio cresce, raggiunge un massimo e poi decresce. Alfa 0.1 significa che la finestra di vulnerabilità è il 10% della dimensione di un pacchetto.



Per studiare cosa succede sul canale si vede l'attività sul canale come una sequenza di cicli (C), dove ogni ciclo è fatto da:

- un periodo Idle (I): stato del canale dove nessuno trasmette
- un periodo Busy (B): stato del canale dove uno o più stazioni trasmettono

Quello chesi fa a questo punto è una cosiddetta analisi di ciclo, una tecnica standard nell'analisi dei protocolli che permette di calcolare il Throughput a lungo termine del protocollo, guardando semplicemente cosa succede nel ciclo tipico (e da qui derivare quello che succede sull'intero sistema) per ipotesi che nell'uso del processo di poisson, tutti i cicli sono statisticamente simili tra di loro. Quindi quello che succede in un ciclo è indipendente.

Un periodo busy può essere "buono" o "cattivo" in base a quello che accade durante quel periodo:

- Il periodo busy "buono" corrisponde alla trasmissione di un singolo pacchetto trasmesso con successo
- Il periodo busy "cattivo" è dato dalla sovrapposizione di più di un pacchetto e quindi sarà un periodo busy in cui c'è una collisione, quindi purtroppo è un periodo busy che non serve, non c'era traffico utile.

Il throughput (**S**) è calcolato quindi in questo modo:

$$\text{Throughput} = \frac{\mathcal{P}_{good}(pkt)}{E(C)} = \frac{\mathcal{P}_{good}(pkt)}{E(I) + E(B)}$$

Il throughput è quindi uguale alla probabilità che un ciclo contenga un pacchetto trasmesso con successo, diviso la durata media del ciclo E(C).

Dove la probabilità che un periodo busy sia di tipo buono, cioè occupato da un singolo pacchetto trasmesso con successo, è uguale alla probabilità che nessun altro pacchetto venga trasmesso nella finestra alfa di vulnerabilità:

$$\mathcal{P}_{good}(pkt) = \mathcal{P}(0 \text{ transmission in } \alpha) = e^{-Gt}$$

A questo punto servono gli altri pezzi, e quindi la durata media di un periodo idle E(I) e la durata media di un periodo busy E(B).

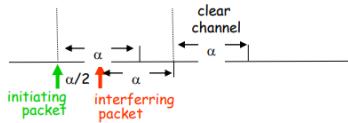
## Analisi di CSMA/CD (cont.)

I distribuito esponenzialmente, a media  $1/G$  ( $E[I] = 1/G$ )

Calcolo di  $E[B]$ :

Caso 1: no collisioni (NC):  $E[B|NC] = 1 + \alpha$  con  $P(NC) = e^{-\alpha G}$

Case 2: collisione (C):  $P(C) = 1 - e^{-\alpha G}$



Calcolo approssimato di  $E[B]$ :

considero caso più probabile: una sola altra trasmissione interferisce col pacchetto che comincia il periodo busy

Il pacchetto interferente arriva in media  $\alpha/2$  dopo l'inizio del periodo busy

Il pacchetto interferente trasmette per  $\alpha$  unità di tempo prima che venga rivelata la collisione (caso peggiore)

Occorre tempo addizionale  $\alpha$  prima che canale diventi idle

Perciò  $E[B|C] = 5\alpha/2$

4-11

la durata media di un periodo Idle viene anche abbastanza semplice, perché dal processo di poisson si riesce a dire immediatamente che la durata media del periodo Idle è l'inverso del parametro del processo di poisson, quindi:

$$E(I) = \frac{1}{G}$$

L'ultimo pezzetto che serve è la durata media del tempo di Busy  $E(B)$ . Dato che esistono due casi di periodi Busy, si utilizza il teorema delle Probabilità totali.

Il caso in cui non avviene collisione, la durata media di un periodo busy è:

$$E[B|NC] = 1 + \alpha$$

Il caso in cui avviene una collisione e si calcola con la probabilità complementare della precedente, quindi:

$$\mathcal{P}_{bad}(pkt) = 1 - e^{-Gt}$$

Per capire quanto è lungo un periodo Busy in collisione si fa una approssimazione, perché calcolare la durata in modo esatto sarebbe complesso, un'analisi semplificata è basata su questa idea qua: quando avviene una collisione la collisione è avvenuta tra due stazioni. Tanto si ottengono dei risultati sufficientemente buoni, accurati considerando anche solo l'evento più frequente, cioè che avvenga una collisione solo tra due stazioni. Alla fine il tempo medio di Busy, nel caso di collisione tra due stazioni si può approssimare come:

$$E[B|C] = \frac{5\alpha}{2}$$

Quindi, usanto le Probabilità totali, il tempo medio di Busy, su entrambi i casi:

$$E(B) = E[B|NC]\mathcal{P}_{bad}(pkt) + E[B|C]\mathcal{P}_{good}(pkt)$$

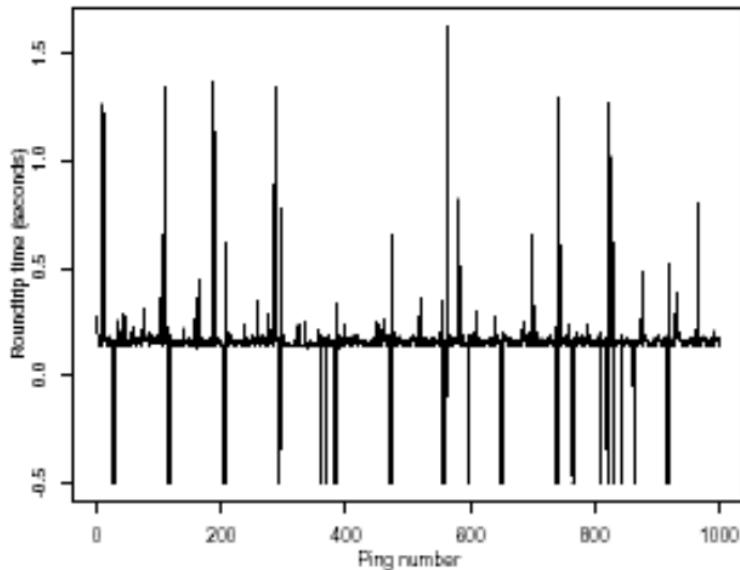
$$\cdot = e^{-\alpha G}(1 + \alpha) + (1 - e^{-\alpha G})\frac{5\alpha}{2}$$

A questo punto si mette tutto assieme, ed il throughput è:

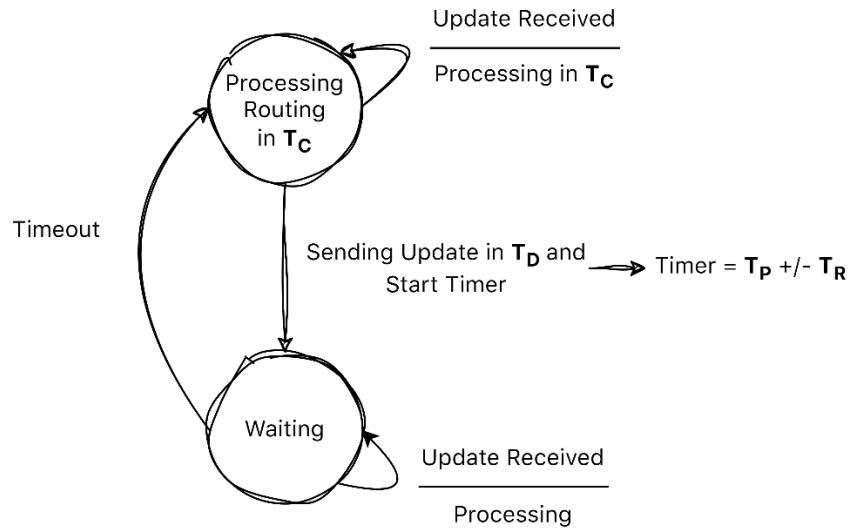
$$Throughput = \frac{\mathcal{P}_{good}(pkt)}{E[I] + E[B]} = \frac{e^{-\alpha G}}{\frac{1}{G} + e^{-\alpha G}(1 - \frac{3\alpha}{2}) + \frac{5\alpha}{2}}$$

Comunque tutto questo per vedere un esempio di uso della randomizzazione per ottenere in forma distribuita la soluzione del problema dell'accesso multiplo condiviso. Avere un'algoritmo distribuito per accedere a un canale condiviso è un'ottima cosa, e di fatto è il motivo che ha decretato il successo di ethernet e poi il successo anche del wifi.

## Randomizzazione sui router



Questo esempio sulla randomizzazione nelle reti, di fatto è più una curiosità storica. A metà degli anni 90, facendo delle misure in rete, dei ricercatori si sono accorti di un fenomeno, molto patologico, che succedeva nella rete che stavano misurando: lanciando in loop il comando ping, notarono che avvenivano delle misteriose perdite di pacchetti, di periodi in cui la rete era congestionata e perdeva pacchetti o comunque gli causava dei ritardi molto alti. I ricercatori si chiesero che cosa stesse succedendo, dopo un po riuscirono a svelare i motivi: i router della rete si sincronizzavano tra di loro, inoltrando tutti nello stesso momento traffico di segnalazione, cioè mandavano nello stesso momento i loro aggiornamenti ai router vicini, che mandava momentaneamente la rete in congestione (piano dati e piano di controllo insieme). I ricercatori si chiesero come fosse possibile uno scenario del genere, dato che ogni router lavorava per conto proprio.

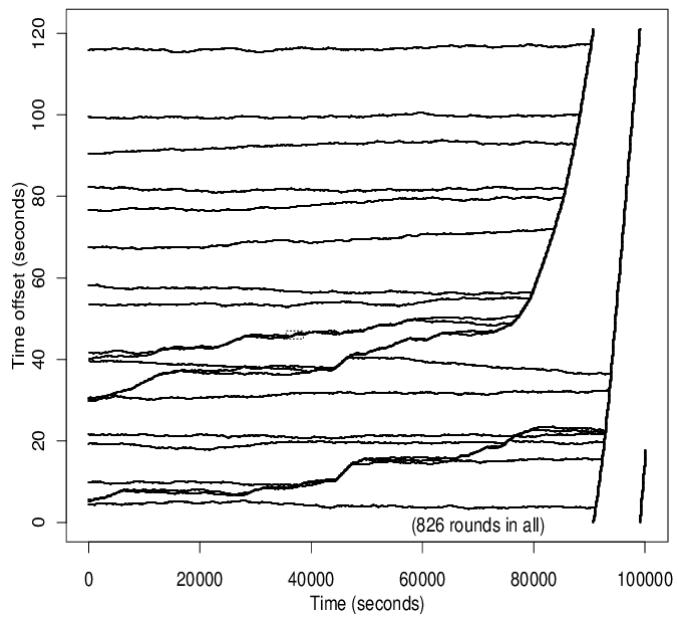


Il motivo è descritto da questo diagramma a due Stati che descrive come il protocollo dei router agiva per inoltrare o ricevere traffico di segnalazione.

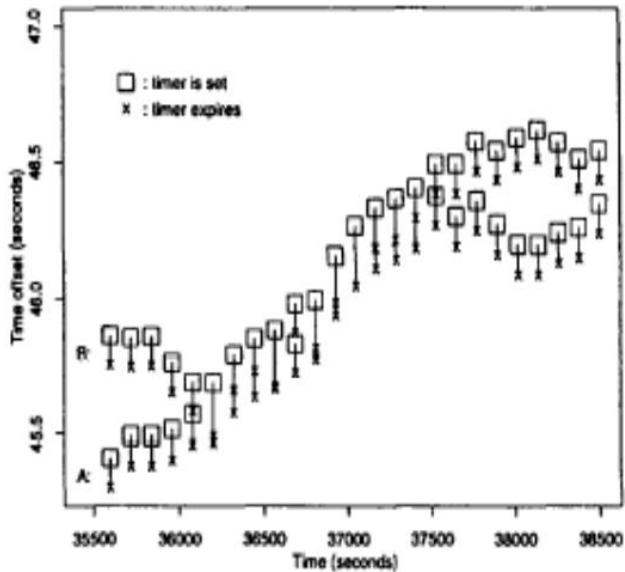
I due stati

- **Stato di attesa (Waiting):** i router non facevano attivamente niente, ricevevano il traffico di segnalazione da parte dei router vicini. Questo traffico verrà utilizzato nello stato di update. La transizione di stato avveniva ogni tot, tramite timeout che durava un tempo  $T_p +/- T_r$ , dove  $T_p$  è un tempo fisso, mentre  $T_r$  è il motivo di tutto questo casino, perché era quasi deterministico, non randomico abbastanza.
- **Stato di update (Processing):** i router ricalcolavano le loro strade, processando tutto il traffico di segnalazione ricevuto durante la fase di attesa dai propri vicini. Al termine del calcolo (che durava un tempo  $T_c$ ), inviavano messaggi di segnalazione ai router vicini. Nel caso in cui durante il calcolo arrivava un altro pacchetto di segnalazione da parte di un router vicino,  $T_c$  veniva fatto ripartire e rieffettuavano i calcoli. Dopo aver inoltrato i nuovi dati, andavano in attesa.

Questi ricercatori hanno dimostrato che la rete si poteva sincronizzare, cioè che i router potevano sincronizzarsi tra di loro e l'hanno fatto con l'aiuto di una simulazione di cui i risultati:



In sostanza avevano simulato 20 router che inizialmente partivano desincronizzati, cioè con tempo di calcolo e tempo di attesa completamente sfasati tra loro e. Partendo quindi in modo casuale con un offset tra 0 - 120 secondi iniziale, all'aumentare del tempo, i router tendevano a sincronizzarsi (dopo tanto tempo, circa 4-5 ore).



Andando ad investigare meglio che cosa succedeva, in questa figura c'è uno zoom che fa vedere meglio come si sincronizzavano due router. I due router si sincronizzano per il fatto che uno di questi due router, espandeva il suo tempo di calcolo perché riceveva un aggiornamento dal router vicino. Effettuando il ricalcolo però, si avvicinava piano piano a sincronizzarsi con l'altro router. A quel punto andranno avanti sincronizzati tra di loro.

Quindi c'è questa correlazione che si innesca tra i router durante la fase di calcolo che li porta possibilmente a sincronizzarsi tra di loro e via via sincronizzarsi sempre con più router, finché tutta la

rete si sincronizza e quindi tutti i router sono sincronizzati tra di loro, generando un traffico di segnalazione sincronizzato che causa quelle perdite periodiche nella rete.

Per evitare tutto ciò la soluzione che poi è stata trovata abbastanza banale, cioè è aumentare la randomicità del tempo di attesa, quindi aumentare quel tempo  $T_R$  che inizialmente era stato scelto troppo piccolo, più grande, in particolare l'articolo suggerisce:

$$T_r \in [.5T_p, 1.5T_p]$$

Morale della favola. c'è il problema delle sincronizzazioni quando si hanno degli oggetti che hanno un comportamento deterministico e i sistemi informatici sono tipicamente di questo tipo, anche se li si fa partire scorrelati nel tempo completamente sfasati tra loro.

## Randomizzazione nelle code

Un altro esempio di randomizzazione riguarda la gestione attiva delle code nei router.

Sulla porta di uscita di un rotuer possono finire un sacco di pacchetti, tutti destinati a uscire su uno stesso link dove può generarsi un problema di congestione. Quindi queste code che crescono portano ad un aumento di latenza e potenzialmente anche perdita di pacchetti, se questi buffer si riempiono.

Si può scegliere di non fare niente e gestire le code di uscita con la politica di default che è quella drop tail. Quindi usare usare uno scheduler FIFO, cioè il primo pacchetto che arriva, esce e quando il buffer si satura, si scarta la coda del buffer (Drop tail). Questo è quello che succede se si usa la metodologia di default, cioè non applica quella che si chiama una **Active Queue Management - AQM**.

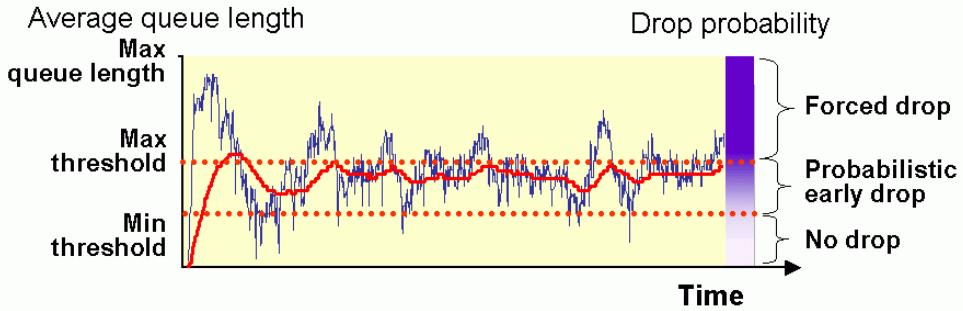
La gestione della coda drop tail può essere una cosa indesiderata, il primo effetto indesiderato è che se si lascia che la coda si allunghi, si intasi, questo causerà un aumento di latenza dei pacchetti e questo fa molto male a certe applicazioni sensibili ai ritardi, tipo applicazioni Voip possono diventare inutilizzabili.

Un'altra cosa indesiderata del drop tail è che, quando una coda drop tail si riempie, comincia a buttare via pacchetti di connessioni che attualmente stanno attraversando quel link. Se si immagina che ci siano connessioni TCP, alla perdita di pacchetti, tutte le connessioni contemporaneamente dimezzano la loro finestra di trasmissione. Questa non è una bella cosa perché possono portare a sottoutilizzate il link, sprecando risorse trasmissive. Inoltre queste perdite che avvengono tutte concentrate in un certo punto, possono portare anche al fenomeno in cui, connessioni che avevano una finestra alta, sopravvivono meglio di quelle che avevano finestra bassa il quale vanno in time out e quindi subiscono un impatto più forte.

## Scarto casuale in anticipo

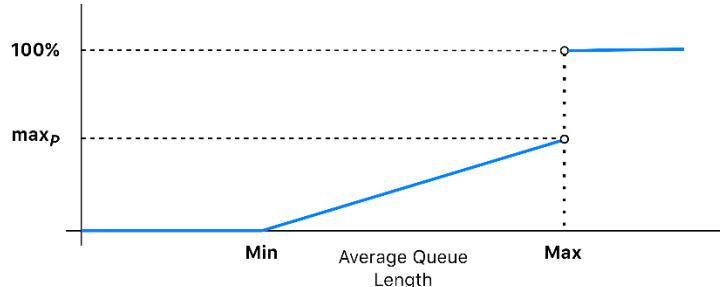
La combinazione di tutti questi motivi ha portato qualcuno a concepire questa diversa modalità di gestione della coda che si chiama Active Queue Management. L'idea è: invece di aspettare che la coda si riempia e quindi prima di perdere pacchetti, si comincia artificiosamente a perderli prima. Vuol dire che anche se la coda non si è riempita, ma è in procinto di farlo, si cominciano a droppare pacchetti e lo si fa in modo casuale (ed è qui che entra in gioco la randomizzazione). I pacchetti che arrivano, li scarto con una probabilità  $P$ . Per rendere la cosa anche più adattiva allo stato della coda, è stato proposto di perdere in anticipo i pacchetti, sempre in base ad una probabilità  $P$ , ma che dipenda anche dallo stato di occupazione della coda, quindi più la coda si riempie, più aumenta la probabilità di buttare via un nuovo pacchetto che arriva.

Questo ha portato ad un vero e proprio meccanismo di gestione attiva delle code che si chiama RED (Random early detection). L'idea è appunto quella di fare una early detection della congestione del buffer, quindi accorgermi che il buffer sta andando in congestione e e risolverlo scartando in modo probabilistico i pacchetti in arrivo.



Gli inventori di RED, in realtà, hanno applicato qualcosa di leggermente più sofisticato, cioè droppare i pacchetti non sulla base dell'occupazione istantanea della coda, ma considerando una lunghezza media della coda, con meno oscillazioni, una sorta di media mobile esponenziale per inseguire in modo più smooth l'andamento medio di occupazione del buffer e meno soggetta al rumore della lunghezza istantanea della coda, che oscilla troppo velocemente nel tempo e che porterebbe a un comportamento troppo schizofrenico della coda, troppo instabile.

In questo modo hanno pensato che il buffer potesse comportarsi meglio, in particolare con i burst di traffico.



Dopodiché quello che si sono inventati è un cosiddetto profilo di perdita, cioè, si vanno a perdere i pacchetti secondo un profilo di probabilità di perdita che dipende dal valore della media mobile. Quando la curva si trova tra una soglia minima e una soglia massima, si ha quindi una qualità di perdita che cresce linearmente. Oltre la soglia massima perdo tutto.

L'obiettivo appunto è di operare nel mezzo, quindi trovare un punto di stabilità del sistema tra la soglia minima e la soglia massima che sono due parametri che posso scegliere, e se riesco davvero a stabilizzare la coda dentro queste due soglie, poi riesco a garantire una latenza, desiderata quindi che non sia troppo alta e quindi non faccio male alle connessioni, ad esempio sensibili al ritardo.

### **RED vantaggi e svantaggi**

I vantaggi di RED sono:

- fornire una transizione morbida da nessun pacchetto perso a tutti i persi che il comportamento invece di drop tail, che è considerato diciamo indesiderabile perché

appunto una transizione troppo forte tra non perdo mai e perdo tutto. E quindi fornire un avvertimento alle di una imminente congestione della rete. Eh?

- Fornire in modo Fear, quindi dare lo stesso tasso di perdita a tutti i flussi per il modo stesso in cui si introducono le perdite (con una certa probabilità indipendente tra tutti i pacchetti), quindi si trattano in modo equo tutti, perché tutti hanno la stessa probabilità di perdere un pacchetto mentre con drop tail c'era il problema della starvation dei flussi che trasmettono poco, hanno a finestra bassa e che quindi possono andare in in time out e venire praticamente uccisi dai flussi invece con finestra più alta che recuperano le perdite più facilmente.
- Evito la sincronizzazione degli eventi di perdita percepiti dalle sorgenti.

Svantaggi di RED:

- Numero di parametro: l'idea di per sé è interessante, il problema è che ha un grande numero di parametri difficili da configurare. Ci sono 4-5 parametri che non si sa bene come scegliere: le soglie, la probabilità, il profilo di perdita.

Tutte le volte che avete delle soluzioni con tanti parametri, queste sono destinate in inevitabilmente a fare una brutta fine, è una storia che si ripete ciclicamente nelle reti.

Ci sono anche degli autori che hanno messo in dubbio i vantaggi di red rispetto a drop tail, dicendo che poi tutto sommato non sono così significativi, non giustificano l'aumento della complessità, quindi il fatto di dover configurare tutti i parametri, etc. Per cui il meccanismo è stato di fatto implementato nei router ma è pochissimo utilizzato se non quasi mai, dagli amministratori di rete

## Randomizzazione in BitTorrent

Un altro esempio di uso della randomizzazione nelle reti è BitTorrent.

Basato sul paradigma peer to peer, inizialmente utilizzato solo per la distribuzione di grandi file. In sostanza gli utenti usano la loro banda di upload per inoltrare pezzi del file ad altri utenti, mettendo a disposizione del sistema la banda aggregata di upload di tutti gli utenti, che può essere molto alta.

Successivamente si è usato anche per applicazioni audio/video in real time, però questo caso (quello dello streaming) è fondamentalmente diverso da quello di file statici, perché impone dei requisiti di ritardo molto più stringenti. In particolare, è necessario che i chunk non arrivino sparsi, bisogna dare priorità ai chunk che stanno per essere riprodotti dal decodificatore. L'uso del peer to peer per lo streaming ha avuto un po di successo in passato, adesso la cosa è andata un po in declino con l'avvento di piattaforme di streaming più basate sulle Content delivery network, tipo Netflix, che hanno vinto rispetto all'approccio peer to peer.

Per distribuire un torrent, quindi si crea un quello che si chiama swarm (sciame di utenti), una rete di peer che in un momento stanno contribuendo alla distribuzione del file, indipendente da tutti gli altri, quindi ogni file viene distribuito su uno swarm diverso.

Il file viene diviso in tante piccole parti, ed ogni parte in sottoporti ancora più piccole, dette chunk. Per implementare tutto ciò, è necessario che tutti siano d'accordo su un modo unico di dividere il file in parti, in modo che tutti abbiano la stessa visione di come il file è stato diviso.

Ogni chunk, a livello logico, viene replicato nel sistema creando un albero di distribuzione. I vari alberi che corrispondono ai vari chunk, sono in un certo senso ortogonali tra loro, quindi ogni chunk definisce un albero di distribuzione che si sovrappongono tra di loro.

## Peer torrent

Peer in generale un'istanza di un client BitTorrent in esecuzione su un computer a cui altri client si connettono e trasferiscono/reperiscono dati.

I peer sono si dividono in due tipi: quelli che stanno scaricando il file, ma ne hanno solo alcuni pezzi, che vengono chiamati **leechers**, e i peer che hanno già scaricato tutto il file, e continuano a contribuire alla sua distribuzione restando nello swarm e vengono chiamati **seed**.

Ovviamente la presenza dei seed è molto utile in quanto, avendo il 100% del file, possono contribuire a distribuire un qualunque chunk. È quindi evidente che per ogni torrent deve esserci almeno un seed (quando non c'è nessun seed nello swarm, bisogna necessariamente partire da un server, quindi ci deve essere almeno un punto che possiede tutti i chunk). A un certo punto non è più necessario che ci sia un server con tutti i chunk, perché il torrent si autosostiene grazie solo agli utenti

## Funzionamento

Quando si apre un torrent, attraverso un client, è necessaria una fase di inizializzazione del nuovo peer, che implica che il nuovo peer scarichi un file dal cosiddetto **tracker**.

Il tracker è l'unico punto client-server del sistema, ed è un server che contiene la visione più recente del torrent, quindi di quali sono gli utenti dello swarm e i loro indirizzo IP, dimensione del file, di quanti pezzi è composto, la dimensione dei singoli pezzi, etc.. (ogni torrent ha un tracker). In un certo senso è anche il punto vulnerabilità del sistema: quando si cerca di distruggere un torrent swarm, soprattutto per la distribuzione di file coperti da copyright, quello che viene attaccato è il server che ospita il tracker.

Dopo aver contattato il tracker, un nuovo peer ottiene una lista casuale di alcuni altri peer dello swarm (primo ingrediente di casualità), che in quel momento stanno scaricando il file. Tipicamente i client BitTorrent iniziano a farsene dare dell'ordine di una cinquantina di questi indirizzi iniziali di altri peer. È una lista che poi varia nel tempo (non si va di oltre 80-100). Non si usa ottenere tutti i peer dello swarm per vari motivi, il primo perché tutti potrebbe essere un'esagerazione, se ci fossero 10k peer, otterrei 10k. Il secondo è che in questo modo randomico si ottiene gratis un modo di bilanciare bene il carico tra tutti i peer, se si scegliessero ad esempio solo alcuni peer si creerebbe congestione verso quei peer. Non è invece necessario che ci sia almeno un seed tra i peer da cui un downloader sta ottenendo chunk, l'importante è l'aggregato di tutti i peer a cui si è connessi, tra di loro, coprano tutti i chunk del file. Ovviamente non si controlla che i peer a cui si è collegati abbiano, aggregati, tutti i chunk del file,

## Bitmap file

Una volta ottenuta la lista iniziale di peer, il client sceglie quattro di questi (in modo randomico) all'interno della lista inizialmente, con lo scopo di cominciare a scambiare pezzi del file. Saranno quindi connessioni attive TCP, che viaggeranno sia in upload che in download. Ed ognuno di questi peer con cui il client ha aperto connessioni, inoltra una bitmap dei chunk che possiede.

In questo modo, i peer riescono a scambiarsi in modo molto rapido, con pochissima informazione, a volte in un solo pacchetto, tutta l'informazione dei chunk che un peer possiede. Questa bitmap è una sequenza di zeri e uni dove, viene marcato con uno, se un chunk è posseduto da quel peer, zero altrimenti. In questo modo, chi riceve una bitmap, potrà facilmente vedere se ci sono dei chunk che gli interessano, e richiederli.

## **Tit-for-tat mechanism**

Cosa succede se uno dei peer da cui ho scelto inizialmente di scambiare chunk, non contribuisce alla distribuzione? Cioè, io invio dati a lui, ma lui non li invia a me?

Questo viene gestito secondo un'altro meccanismo chiave di BitTorrent, il cosiddetto tit for tat. Questo meccanismo serve ad incoraggiare gli utenti a condividere la loro banda in upload e a scoraggiare invece i cosiddetti free-rider, cioè peer che si attaccano al sistema ma scaricano solo senza contribuire.

L'algoritmo è abbastanza semplice: il client misura (ogni 30 secondi) il rate in downstream per ognuno dei quattro peer con cui ha aperto le connessioni TCP. Allo scadere dei 30 secondi, il client scarta il peer che ha contribuito di meno, tra i quattro iniziali, e le estrae un altro dalla lista che il tracker gli aveva inviato, con la speranza di attaccarsi ad uno che magari ha più banda. In questo modo viene scoraggiato il free-riding.

Da notare che il nuovo peer che si estrae, è randomico, quindi può essere anch'esso un free-rider, quindi il client sarà "buono" con lui per i 30 secondi successivi all'estrazione, poi lo butterà via. Questa idea di essere inizialmente generosi verso un nuovo peer casuale per i primi 30 secondi, è anche utile per far entrare nello swarm, i nuovi peer che non ha ancora niente da scambiare infatti, all'inizio, il nuovo peer non riescirà ad apportare niente, però grazie al fatto che ogni 30 secondi altri peer lo faranno entrare nella loro cerchia dei quattro, avrà la chance di entrare nel sistema e di cominciare a scaricare i primi chunk.

In questo modo si riesce ad estrarre il più possibile chunk da un peer a cui si è connessi finchè quello non ha più niente da offrire, o viceversa. In generale quando si diventa inutili l'uno per l'altro, si viene scartati.

## **Rarest-first mechanism**

Un ingrediente molto utile che è stato implementato dentro il protocollo, è il meccanismo del rarest first. Questo meccanismo concede priorità ai chunk più rari.

L'algoritmo è il seguente: tra i peer con cui sto scambiando chunk, tra tutti i nuovi chunk a cui io potrei essere interessato, scelgo quello che in un certo momento è il meno replicato tra i peer, cioè si da priorità al chunk che si osserva essere il meno replicato. Questo viene fatto usando la bitmap: con un po di calcoli il client ordina dal più diffuso a quello meno diffuso i chunk di tutti i peer a cui ha aperto attualmente connessioni. Una volta individuato il più raro, chiederò prima quello, poi i restanti.

In questo modo si contribuisce in modo efficiente alla distribuzione totale del file: fare in modo che non ci siano dei chunk molto diffusi ed altri meno diffusi, ma che tutti tendono ad avere la stessa distribuzione nello swarm in modo di nuovo da bilanciare le risorse tra tutti e non creare Colli di bottiglia.

C'è un caso in cui vengono disattivati i meccanismi di Rarest-first e questa eccezione avviene appena un peer si connette allo swarm, quindi, all'inizio quando un peer si connette per la prima volta, non utilizza il rarest-first, andando a scegliere i chunk più rari, ma sceglierà i chunk in modo randomico, sempre utilizzando le bitmap. In questo modo, il nuovo peer ottiene rapidamente dei pezzi con cui inizierà a contribuire per la distribuzione del file e di conseguenza a non essere scartato dall'algoritmo tit-for-tat, ottenendo subito qualcosa che possa barattare con gli altri. È per questo che all'inizio di un torrent si è lenti: non si hanno chunk, tutti ti butteranno fuori, finchè non riesci a contribuire anche tu.

## **Problema dell'ultimo pezzo**

Un'altra parte delicata del protocollo è la parte finale di un file, la cosiddetta fine del gioco, e cioè quando mandano pochissimi chunk alla fine del download. Quando mancano pochissimi chunk ad un file, potrebbe essere difficile trovarli, e che il client stenta a trovare nell'attuale lista a cui è connesso. Ed è per il motivo per cui può succedere che certe implementazioni di BitTorrent rallentino nella parte finale, magari andando al 99.9%, e poi bloccandosi perché hanno fatica a trovare gli ultimi chunk.

Un rimedio che è stato trovato a questo problema è che verso la fine del download, si utilizza un'altra tecnica che è quella di mandare a tutti i peer (non solo quelli connessi), delle richieste per l'ultimo (o ultimi) chunk che manca. Quindi il client manda in broadcast a tutti i miei peer chiedendo particolari chunk, a quel punto se un peer possiede alcuni dei chunk finali, li inoltrerà.

Non si è capito esattamente perché, ma ci sono delle situazioni dove ancora si incontra il problema dell'ultimo pezzo, il sistema è molto complicato da studiare e non si capisce se è dovuto a cattive implementazioni del client oppure se proprio un problema intrinseco del meccanismo. Alcuni hanno proposto il meccanismo del network coding, ma l'idea poi è stata un po scartata: lo stesso inventore di BitTorrent Coen, a un certo punto ha affermato che questa implemtazione fosse esagerata.

## **Dettagli**

La fase iniziale, quella di estrarre i peer dal tracker, permette di evitare disconnessioni della rete, e di avere un grafo non connesso. L'obbiettivo è tenere lo swarm il più possibile connesso, non si vuole che uno swarm si divida in sotto swarm, non più collegati tra di loro. Essendo che io un client ottiene la lista di peer è assai improbabile che la swarm si partizioni in dei sottosuoli indipendenti.

Un altro dettaglio del protocollo è il ruolo dei seed. I seed sono dei peer generosi che dopo aver scaricato il file 100% non se ne vanno, ma restano nello swarm. Per i seed, è necessario inventarsi un meccanismo nuovo di aggiornamento della lista dei peer. Questo perché i seed non scaricano più niente, inoltrano solo, quindi non misurano più nessuna banda dai loro peer. È necessario inventare un altro meccanismo (diverso da quello dei leechers che usano il tit-for-tat) per evitare che un peer è agganciato sempre allo stesso seed, un peer sanguisuga.

Per ovviare a questo, viene adottato un classico meccanismo basato sul tempo, quindi i seed hanno una lista di peer a cui stanno inviando chunk, e la aggiornano automaticamente dopo un tot di secondi, scartando la connessione più vecchia, il peer a cui hanno dato di più e ne fanno entrare un nuovo peer random.

## **Randomizzazione e bilanciamento del carico**

Un caso del tutto diverso di uso della randomizzazione è il problema di bilanciamento del carico. Il problema del binalciamento del carico si osserva con il fenomeno che si chiama la potenza delle due scelte.

Il fenomeno è possibile spiegarlo attraverso un'analogia: Una persona che si reca al supermercato, ed una volta finito di fare il giro deve scegliere in che cassa andare. Si suppone ora che ce ne siano N di casse, quindi N servitori diversi. Si suppone inoltre che la persona, a differenza di quello che succede in un supermercato, non riesca a vedere le casse (quello che accade nella realtà è che la persona andrebbe in quella più scarica perché intuitivamente la cosa migliore da fare per diminuire la latenza, quindi per impiegare meno tempo per uscire è andare nella coda più vuota).

Ora se si proietta questo problema in un sistema di load balancing in rete, dove non si ha la visione dello stato delle code di tutti i servitori che possono soddisfare la richiesta. Questo perché, essendo

tanti, se tutti dovessero inviare il loro stato per capire chi è il più scarico, si creerebbe un traffico di segnalazione assurso (si immagina uno scenario di un datacenter da 10k server...). Però intuitivamente, questa soluzione è la migliore, quella che se tutti usassero bilancerebbe nel modo ottimale il carico tra tutti i server, e viene chiamata **Join Shortest Queue**.

Quindi, quando arriva una nuova richiesta, in che modo viene distribuita su uno dei server, in modo da minimizzare la latenza media cioè, il tempo medio speso da un cliente per uscire dal sistema ed essere servito? Si potrebbe scegliere uno dei server a caso (random), usando randomizzazione, oppure una via di mezzo, ovvero, quella di non scegliere a caso una sola coda, ma vengono selezionate N code a caso e tra le N estratte a caso, si inoltra la richiesta nella coda più corta.

Quindi quale tra le politiche usare? Join Shortest Queue, random, oppure quest'ultima? Inoltre, cosa succede se aumenta N da due a tre? O da tre a quattro? Da notare che se N = 1, si ha la politica Random, mentre con N = Max, si ha la Join Shortest Queue.

La cosa molto sorprendente che è stata scoperta è che già solo con N uguale a due, quindi passare semplicemente da random semplice con N uguale a uno ad una con N uguale a due ha un'enorme beneficio, la latenza media del sistema si abbatta in modo logaritmico e cioè c'è un'enorme miglioramento dal passare da N = 1 ad N = 2. Nel passare da N = 2, ad N = 3,4,5.. si ha un miglioramento leggero, aumenta solo di un fattore costante.

In particolare:

- Join Shortest Queue:  $O(n)$
- Random N = 1:  $O(\log(n))$
- Random N = 2:  $O(\log(\log(n)))$

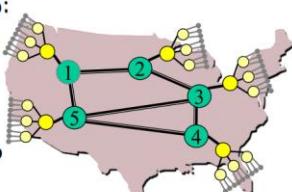
Per riassumere, random N= 1 è facile da implementare, prestazioni scarse. La Join Shortest Queue ha ottime prestazioni ma è difficile da implementare perché richiede conoscenza globale dello stato. Random con N = 2 è facile da implementare perché richiede di esplorare solo due code scelte a caso, e funziona enormemente meglio di random N = 1, abbattendo la latenza in modo molto grande e avvicinandosi quasi alla soluzione ottima

## Instradamento randomizzato a due hop

**Obiettivo:** scegliere percorsi da router di ingresso a router di uscita in modo:

**robusto** a cambiamenti nel traffico:

- i link non vadano in sovraccarico, anche se i tassi sorg/dest cambiano



**con minimo overhead**

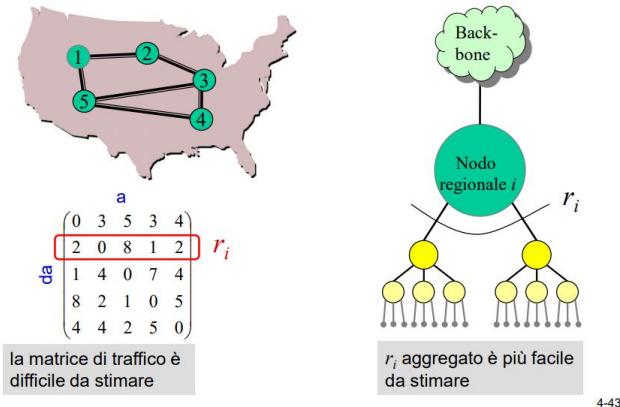
- di aggiornamenti di instradamento al variare del traffico

4-42

Un problema di bilanciamento del carico, si nota in uno scenario di instradamento con il routing, quindi di instradamento. L'obiettivo è smistare il traffico da smaltire dentro la rete cercando di bilanciarlo sui link interni della rete. Quindi l'obiettivo è scegliere dei percorsi da un router di ingresso ad un router di uscita, in modo da tenere un routing che bilanci bene il carico e che sia possibilmente robusto a cambiamenti nel traffico, quindi, anche quando avvengono dei picchi di traffico la rete sia in grado di assorbirlo. Un obiettivo di un sistema di questo tipo è che sia semplice,

che non risulti troppo complesso nel calcolo di nuove strade. Il problema passa anche sotto il nome di ingegneria del traffico, quindi di come smistare in modo ottimale del traffico dentro una rete.

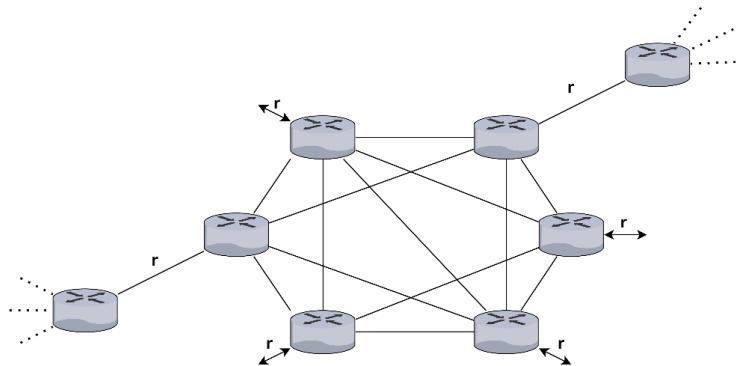
## Matrici di traffico



## Matrici di traffico

Quello che in teoria si potrebbe fare, che però richiede un sacco di informazione è avere quella che si chiama la matrice di traffico, cioè l'informazione completa di tutta la quantità di traffico che entra da un certo nodo e esce verso un altro nodo. È appunto possibile effettuare il bilanciamento più perfetto possibile a patto però di conoscere l'intera matrice di traffico. Il problema nella reti è che spesso la matrice di traffico è ignota: è difficile anche da stimare. Quello che invece andiamo a supporre, è che la matrice di traffico completa non lo sappiamo, però quello che è possibile fare, è trovare un limite superiore a tutto il traffico aggregato che entra dentro un nodo e viceversa tutto il traffico che esce. Questa cosa è molto più semplice da misurare è anche facile da trovare perché è sufficiente considerare la capacità totale del link o dei link che entrano e escono da uno di questi router, perché quello che può arrivare o uscire da un certo nodo non potrà mai superare la capacità massima del link in ingresso e un'uscita dal router.

Supponendo quindi di conoscere il traffico totale che entra o esce da un router di bordo, una tecnica molto elegante che spalma bene il traffico ed è robusta a picchi di traffico ed è molto semplice da realizzare che passa sotto il nome di Valiant Load-Balancing. È una tecnica di instradamento randomizzato a 2 hop. Per capire questa tecnica conviene ragionare su una rete a maglia completa.



Quindi si suppone il caso in cui una rete è una rete a maglia completa, dove tutti i nodi sono connessi con tutti gli altri (anche se questo non è vero nella realtà). Si parte da una situazione

idealizzata di questo tipo, dove tutti sono connessi con tutti, perché se risolviamo questo problema poi lo possiamo facilmente mappare su un caso dove si ha una rete fisica non a maglia completa ma con una certa topologia particolare.

Il meccanismo è il seguente: ogni pacchetto che entra nella rete effettua una strada a 2 hop, in cui il primo nodo è scelto a caso, quindi il primo hop è randomico, scegliendo uniformemente tra gli altri  $n - 1$  nodi. Dopo che il pacchetto ha effettuato il primo hop random, il secondo invece, è quello che deterministicamente porta al nodo di uscita di destinazione vero. Durante il primo Hop random, potrebbe anche succedere di mandarlo, direttamente al nodo di uscita giusto, in quel caso non effettua più due hop.

Per quale motivo bisogna fare così? Il motivo lo si capisce andando a stimare il caso peggiore della capacità che mi serve su ciascuno dei link logici sulla rete.

E come detto, non si ha la conoscenza dettagliata della matrice di traffico, ma soltanto una stima o un limite su tutto quello che entra o esce da ciascun nodo. Quello che si può facilmente fare è calcolare questa capacità richiesta su ogni link che nel caso peggiore è  $R$ .

Utilizzando il vailant load-balancing, la banda che si ha bisogno su ciascun link, nel caso peggiore è:

$$\frac{2R}{N - 1}$$

Quando  $N$  cresce, cioè il numero di router è necessario allocare poca banda sui link.

Il motivo della formula è il seguente: su ciascun link bisogna valutare cosa succede nel traffico massimo può transitare sopra. Se si divide questo traffico in due contributi:

- il traffico che sta facendo il primo hop
- il traffico che sta facendo il secondo hop.

allora ciascuno di questi termini vale al più  $R/N - 1$

motivo per cui ciascun hop dà un contributo  $R/N - 1$  è dovuto al fatto che si spalma il carico su tutti i nodi. Quindi, ad esempio, il traffico che sta facendo il primo hop random tra due nodi, non potrà mai essere più grosso di  $R$  su  $N - 1$ , perché di tutto quello che arriva in un nodo, indipendentemente da dove è diretto viene spalmato su  $N - 1$  altri nodi.

Similmente posso fare un ragionamento per il secondo hop che porta di nuovo lo stesso contributo, quindi, prendendo di nuovo un nodo a caso, qual è il traffico che va verso questo nodo e che sta facendo il secondo hop? Di nuovo, questo traffico non può essere più grande di  $R/N - 1$ , perché è stato precedentemente spalmato forzatamente su  $N - 1$  nodi, quindi da un particolare nodo, quello che arriva dentro di me non può essere più grande di  $R/N - 1$ . Questa stima, peraltro, è anche pessimistica perché assume che ci siano sempre da fare entrambi gli oppi, il primo e il secondo. Come giustamente mi ha detto qualcuno di voi, posso andare meglio di così, perché magari sono fortunato e faccio il primo, operando mi porta direttamente alla destinazione. Vabbè, in quel caso andrò ancora meglio, quindi Eh non. Come si può facilmente capire che comunque questo Bound due  $R$  su  $N - 1$  è davvero un Bound di worst case, valido in. Tutti i casi.

E valeva poi la legge dei grandi numeri che mi garantisce che su numeri così grassi su numeri così grossi, davvero il bilanciamento funziona. Comunque l'osservazione giusta, eh però, quello che forse non era forse chiaro è che sto ragionando pacchetto per pacchetto e ho una quantità di pacchetti enorme.

Con un'ulteriore uso della randomizzazione, in modo molto semplice ed elegante si riesce a spalmare bene il traffico dentro una rete anche quando non si conosce la matrice di traffico completa, ma ho solo appunto una stima di aggregati che entrano ed escono dai outer di bordo. In questo modo si ottiene una cosa semplice, robusta, in grado di assorbire facilmente i picchi di traffico e di facile implementazione, quindi con un basso overhead.

Perchè fare due hop quando ne potrei fare uno solo?

E qual è il ruolo della randomizzazione in questo scenario?

Perché non è detto che la distanza tra la sorgente e la destinazione è migliore, cioè nel caso in cui io dovessi effettuare un trattamento diretto non è detto che la distanza sia migliore rispetto all'andamento attraverso due o.

Fare l'hop diretto singolo richiede di allocare più risorse nella rete, cioè di mettere un'infrastruttura, posare dei cavi ed avere una banda allocata molto più grande anche di un fattore N più grande, rispetto a quando faccio i due hop, quindi costa molto di più. Quando faccio i due hop si ottiene uno spallamento ottimale del carico che fa sì che si ha solo più una banda richiesta su ciascun link logici di R/N e quindi costa molto meno.

Quindi il ruolo della randomizzazione qui è quello di ottenere un semplice meccanismo. Bilanciare il carico nella rete assorbendo picchi di traffico impredicibili.

Il traffico massimo che devo smaltire dentro la rete non sarà mai più grande di N volte R, per limiti fisici della capacità che ho in ingresso e in uscita questi nodi, quindi un Bound superiore al traffico massimo che devo smaltire globale aggregato è N volte R. Allora se io faccio la randomizzazione a due hop spalmo questo traffico bene su tutta la mia rete, sui miei link, anche se non conosco nel dettaglio dove vogliono andare

Posso anche farlo con una granularità un po meno ridotta, quindi ragionare su flussi delle varie connessioni che si aprono dentro la mia rete. Io scelgo casualmente il primo hop, però tutti i pacchetti di quel flusso attraversano poi sempre quella strada lì. Allora di nuovo lo spalamento funziona perché se ho tantissimi flussi 100.000 flussi e di nuovo vale la legge dei grandi numeri per cui spalmo bene comunque il mio carico sulla rete e in più garantisco che tutti i pacchetti di uno dello stesso flusso questa connessione vadano in ordine e quindi seguono tutti la stessa strada di nuovo su sistemi molto grossi. Posso anche fare così, quindi ragionare non per pacchetto ma per flusso.

# Indirezione

L'argomento dell'indirezione lo vediamo di nuovo come un principio generale che ha trovato applicazioni in una serie di esempi nel mondo delle reti.

Però, un po come abbiamo visto nel caso della randomizzazione il concetto è molto generale e di ampio uso non solo nel mondo delle reti, ma in generale in tutti i sistemi informatici.

Parliamo di indirezione tutte le volte che abbiamo una entità A che avrebbe voglia di accedere ad una altra entità B (accedere è visto come un concetto astratto, può significare connettersi a B, accedere a B, etc) però non lo fa direttamente, ma in modo indiretto tramite un'altra entità X. Quindi, in un certo senso, A delega a X il compito di accedere fisicamente a B.

In informatica si possono anche avere indirezioni multiple, cioè A che vuole accedere a B può passare attraverso X, che chiede ad Y che chiede a Z, che finalmente arriva a B. Quindi ci possono anche essere poi livelli multipli di indirezione.

## Indirezione nel Multicast

Il primo esempio è il Multicast.

L'implementazione del multicast che era stata concepita, utilizzava appunto l'astrazione del gruppo Multicast: le sorgenti che vogliono mandare un pacchetto in multicast hanno il semplice mestiere di prendere il pacchetto e mettere come indirizzo di destinazione un indirizzo di tipo multicast. Sono poi i ricevitori, che dentro le loro reti locali si iscrivono al gruppo Multicast, cioè informano il loro gateway attraverso IGMP dicendo di essere interessato a far parte di questo gruppo multicast

Quello che otteniamo in questo scenario è a tutti gli effetti un uso dell'indirezione, nel senso che, le sorgenti non conoscono quali sono i loro ricevitori. Si è riusciti a disaccoppiare le sorgenti dai ricevitori, sollevando le sorgenti dal compito di conoscere gli indirizzi di tutti i loro ricevitori. Il compito delle sorgenti quindi è normalmente semplificato, cioè non sono loro che devono gestire il problema di tener traccia dei ricevitori interessati a ricevere il loro traffico.

Quali sono i vantaggi che si ottengono da una soluzione di questo tipo, quindi basata su l'astrazione del gruppo Multicast e quindi i vantaggi di fare in questo modo piuttosto che in una soluzione che non fa uso degli indirizzi multicast, ma usa gli indirizzi IP normali?

Il nostro scopo è quello di risparmiare nella trasmissione dei pacchetti mandando il minor numero di repliche in giro per la rete. Inoltre, il grosso vantaggio che si ottiene dalla indirezione è quello di dividere un problema in tanti sottoproblemi in cui ciascuna componente si occupa del suo specifico compito: i ricevitori si occupano di iscriversi al gruppo, entrare e rimuoversi tramite un certo protocollo. Poi c'è la componente di istradamento dove si occupa di tutto il protocollo di routing multicast, ed il compito delle sorgenti è solo invece quello di banalmente inviare i loro pacchetti verso l'indirizzo di destinazione multicast.

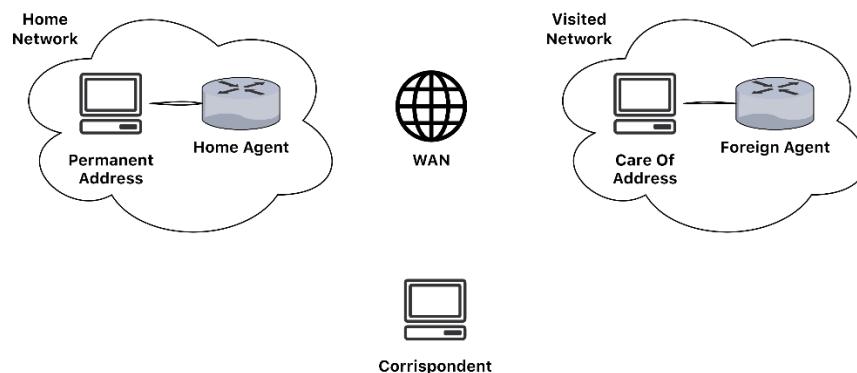
Quindi un problema inizialmente molto complesso, si è diviso in tanti sottoproblemi in cui ognuno risolve una parte del compito secondo le proprie strategie, rendendo tutto trasparente per le sorgenti. È un effettivo vantaggio rispetto a quello in cui le sorgenti dovessero occuparsi di tutto e quindi conoscere quali sono i ricevitori interessati a loro e gestirli, tenerne traccia se sono ancora vivi o sono staccati, gestire quante copie mandare e a chi, cioè sarebbe un problema di complessità enorme.

Un'altra cosa che può anche essere interessante nell'utilizzo di questa tecnica, riguarda la privacy e la sicurezza, cioè il fatto che le sorgenti non conoscono chi sono i ricevitori.

## Mobilità e indirezione

Secondo esempio è l'uso dell'indirezione per risolvere il problema della mobilità nel fatto di avere degli utenti che si spostano, vanno in giro per il mondo e vogliamo raggiungerli ugualmente.

In generale nelle reti abbiamo visto già più volte le complicazioni che nascono dal fatto che gli utenti si muovono (tipo nelle reti cellulari, o SIP). Ora cerchiamo di capire l'uso che si fa dell'indirezione per risolvere questo problema, anche se si utilizza l'approccio ddel Routing indiretto oppure del routing diretto



Per capire come funziona il routing diretto ed indiretto, un po di terminologia:

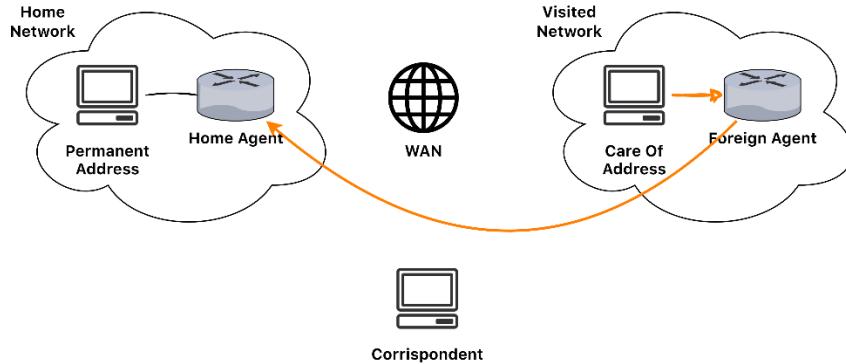
- **Il correspondent:** colui che vuole parlare al nodo mobile, quindi è sorgente di una comunicazione.
- **Home Network:** rete di casa “permanente” del nodo mobile
- **Permanent address:** un indirizzo IP permanente, un qualche identificativo univoco che il nodo mobile ha in modo statico.
- **Home agent:** una sorta di proxy, cioè una entità che dentro la home network svolge funzioni di mobilità per conto dell’utente mobile,

Quando il nodo mobile si sposta, entrerà in una visited network, dove ottiene un nuovo indirizzo, un indirizzo temporaneo che gli viene assegnato solo per il tempo in cui è dentro la rete visitata, che prende il nome di **Care of address**. Da notare che il nodo mobile, quindi, ha due indirizzi, il permanent ed il Care of address.

All'interno della visited network, è presente **Foreign Agent** che è di nuovo una entità terza che svolge funzioni di mobilità per conto del nodo mobile. Il ruolo di questa entità può in realtà essere svolta direttamente dal nodo mobile stesso, quindi concettualmente, è possibile separare il nodo mobile e il Foreign agent, considerando il Foreign agent come un processo in esecuzione sul dispositivo del nodo mobile stesso

### Registrazione

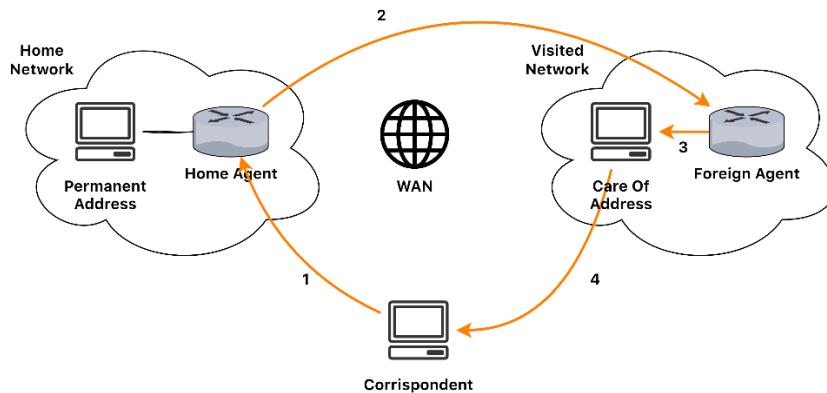
Quando il nodo si sposta, deve in qualche modo rendersi conto del fatto che si è spostato, e deve registrarsi presso il suo Foreign agent, indicando il fatto di essersi spostato e che si trova dentro la visited di network.



Quindi il nodo mobile si attacca ad una nuovo punto della rete, effettua la procedura di registrazione in cui essenzialmente si informa l'home agent di essersi aggangiato dentro una visited network (è il Foreign Agent ad inviare poi il care of address che è stato assegnato al nodo mobile). Quindi ci sono dei messaggi di segnalazione che vengono scambiati. Questa parte è necessaria, altrimenti evidentemente se manca la fase di registrazione, non c'è speranza di rintracciare il nodo mobile.

### Mobilità con routing indiretto

Ponendo di aver fatto la procedura di registrazione, il corrispondente può utilizzare due strategie per aprire una sessione verso il modo che si è spostato. La prima è la soluzione indiretta.



Nel routing indiretto, il corrispondente continua a comunicare con il nodo mobile, ignorando completamente il problema della mobilità continuando semplicemente a far finta che il nodo mobile sia sempre a casa sua. Quindi nel routing indiretto, il corrispondente manda dei pacchetti usando l'home address come destinazione. Questi pacchetti vengono intercettati dall'home agent che sa in che visited network il nodo mobile si trova in quel momento, quindi li intercetta e li dirotta verso la visited network. Quando questi pacchetti dirottati arrivano nella visita del network, poi vengono consegnati al nodo mobile.

L'instradamento è indiretto perché prima va dal corrispondente verso la home Network e poi i pacchetti da lì vengono mandati alla visita in network, quindi in modo indiretto.

Da notare che nella strada al contrario non accade questo: il traffico di ritorno non è vincolato a fare di nuovo la stessa strada complicata indiretta di partenza, ma il traffico può essere mandato direttamente dal nodo mobile verso il corrispondente che, quando ha cominciato a parlare ha rivelato il suo indirizzo, ha rivelato la sua posizione, la sua raggiungibilità tramite un certo indirizzo di sorgente.

## Vantaggi e svantaggi

Un problema di cui soffre la strategia indiretta è il cosiddetto problema di routing a triangolo, cioè i pacchetti effettuano un routing poco ottimale: effettuano una strada lunga (a volte distanze veramente notevoli) quando non c'è ne sarebbe bisogno perché i pacchetti vanno sempre verso la home network e poi dalla home network verso la visited, potenzialmente allungando il percorso di molto rispetto a quello che sarebbe strettamente necessario.

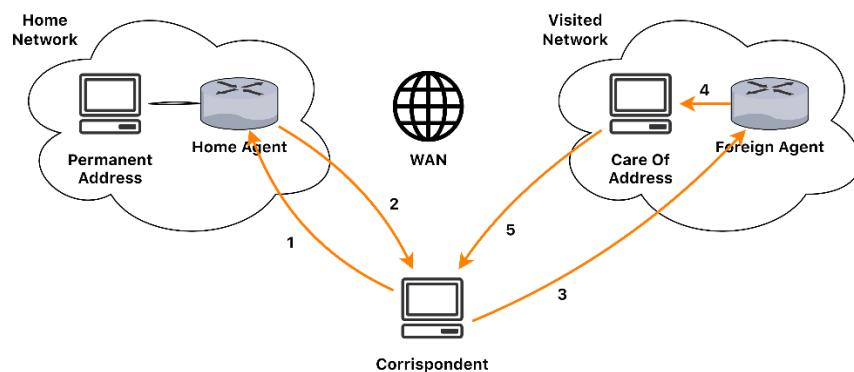
D'altra parte il routing indiretto ha invece un'enorme vantaggio che è quello che il tutto è trasparente per il corrispondent, cioè il corrispondente il mittente e ignora completamente il problema della mobilità. Non ha bisogno di preoccuparsi in minimo modo di andare a rintracciare l'utente mobile.

L'altro grosso vantaggio del routing indiretto è quando l'utente mobile si sposta durante una sessione, cambiando visited network. Con l'approccio indiretto succede che le connessioni in corso possono essere mantenute, cioè non c'è bisogno di abbattere la sessione e cominciare di nuove. Il motivo è proprio basato sul fatto che le cose vengono fatte in modo indiretto, quindi, appena l'utente mobile si accorge di aver cambiato rete, effettua una nuova procedura di registrazione, tutto questo all'insaputa del mittente.

## Problema dell'egress filtering

Nel routing indiretto quindi le connessioni possono essere mantenute perché in particolare il nodo mobile può rispondere direttamente al nodo mittente usando il Permanent address dalla visited network. Il fatto che risponda come IP sorgente con il suo permanent address, la cosa può essere ostacolata dall'egress filtering.

## Mobilità con Routing Diretto



con il routing diretto, prima di cominciare l'effettivo trasferimento dati, il corrispondent si fa comunicare dall'home agent l'indirizzo (care of address) in cui si trova il nodo mobile. Quando il corrispondent ha ottenuto l'indirizzo, comincia una sessione effettiva verso il nodo mobile che in questo caso avviene direttamente, quindi non più facendo una strada a triangolo ma aprendo la connessione diretta più corta dal mittente verso il destinatario.

In questo modo si ottengono vantaggi e svantaggi opposti rispetto al metodo indiretto

il vantaggio è che la cosa non soffre del problema del routing a triangolo. D'altra parte questa volta la soluzione per il corrispondent ha una complessità molto maggiore, cioè il corrispondent deve essere informato dell'attuale indirizzo del destinatario prendendolo dall'home agent, quindi la soluzione non è più trasparente per il mittente.

E anche se il nodo mobile si sposta durante la sessione c'è un problema con in routing diretto. Quindi se il nodo mobile cambia visited network durante la sessione, la connessione che aveva aperto il correspondent con l'utente mobile non funzionerà più, perché il care of address ottenuto non è più valido. Bisogna quindi chiuderla e farsi dare il nuovo care of address.

### **IP Mobile**

Routing diretto ed indiretto sono raccontate ad alto livello nell'RFC 3220 e successivi, che si occupano di IP mobile con tutti i dettagli della fase di registrazione di scoperta, di Incapsulamento.

Questa tecnica ormai è diventata obsoleta perché questo standard è stato concepito quando si aveva in mente di far tutto con l'indirizzamento IP. Al giorno d'oggi la mobilità non viene più gestita tramite lo standard dell'IP mobile, perché ci sono altri protocolli, altre altre architetture che lo gestiscono (SIP, reti cellulari)

Nelle reti cellulari ad esempio si usa l'analogo del routing indiretto e il routing anche segue un approccio indiretto, cioè prima il traffico viene mandato verso la rete di casa verso il GMSC.

## **Infrastruttura di internet con indirezione**

Alcuni ricercatori hanno proposto la costruzione di una generica infrastruttura di indirezione in Internet, ovvero utilizzare il meccanismo di indirezione come un servizio del tutto generico che può essere supportato in senso astratto dalla rete ed essere applicato in vari contesti.

I motivi per cui questo può essere utile è legato al fatto che l'infrastruttura di Internet tradizionale, per come è stata concepita inizialmente, ha i suoi limiti ed in particolare ha il suo limite principale nel fatto di essere stata costruita attorno al paradigma di comunicazione punto-punto (quindi da un indirizzo IP a un altro indirizzo IP). Quindi il meccanismo di base in Internet È quello che Permette di inviare un pacchetto da un host A ad un host B, assumendo che il mittente ricevitore siano in posizioni fisse ben note e che siano raggiungibili tramite un certo indirizzo IP.

Questo schema base purtroppo fallisce, o per meglio dire crea delle difficoltà quando si hanno delle applicazioni che richiedono altre primitive di comunicazione che non sono quelle punto-punto, come ad esempio il multicast (punto-multipunto). Anche la mobilità è un caso che non rientra nello schema classico, perché il destinatario mobile non è necessariamente in una posizione fissa e nota, quindi è un paradigma anycast (da un punto a un qualche altro punto variabile nel tempo). Anche applicazioni che utilizzano altre primitive che non siano punto-punto, come ad esempio servizi che distribuiscono lo stesso contenuto su più server distinti, ai client andrebbe bene uno qualsiasi di quei server, per avere quel contenuto (anche qui Anycast). Tra l'altro il paradigma anycast sta diventando quello di maggiore interesse nelle reti.

Utilizzando un'infrastruttura che si basa sul meccanismo di direzione, questo permette di superare le limitazioni dello schema classico punto-punto, fornendo questi nuovi servizi tipo il multicast, mobilità ed anycast. Sulla base di queste considerazioni c'è chi ha appunto proposto questo tipo di architettura in internet che è interessante vedere di cosa è necessario implementare, se si volesse davvero costruire un'infrastruttura che fornisca un generico servizio di indirezione.

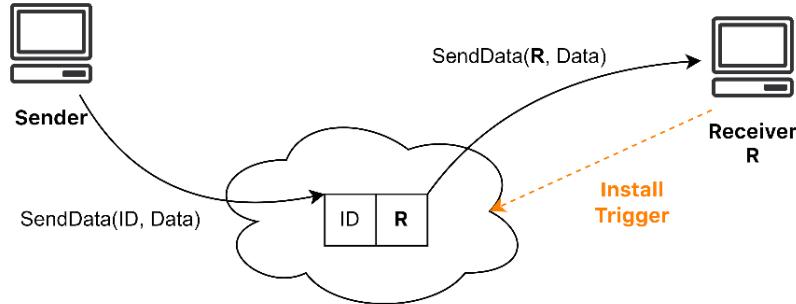
### **Architettura proposta**

Una delle possibili implementazioni, è stata discussa in un [articolo del 2004](#).

La proposta è appunto un cambiamento fondamentale nel paradigma di comunicazione che invece di essere punto-punto, basato su indirizzi dei nodi, ruota attraverso l'identificativo dei contenuti. Quindi i pacchetti non vengono più trasmessi nella rete sulla base dell'indirizzo di destinazione, ma

sulla base di un identificativo richiesto o del contenuto che viaggia nel pacchetto stesso in risposta ad una certa richiesta.

Se si effettua un cambiamento di questo tipo:



I ricevitori che sono interessati ad un certo contenuto con un certo identificativo ID, installano nella rete un cosiddetto trigger che indica la loro manifestazione di interesse a ricevere il contenuto ID nel farlo, specificano anche il loro indirizzo da dove possono essere raggiunti. I trigger sono mantenuti in una rete overlay.

I nodi che possono fornire quel contenuto, mandano dei pacchetti verso il nodo dove può essere stato installato un trigger per quel contenuto, quindi i trasmettitori non mandano direttamente i pacchetti ai ricevitori. Una volta che i pacchetti marcati con ID arrivano al trigger per quel contenuto ID, da qui avviene la connessione verso i ricevitori. Quindi il trigger estrae i dati nei pacchetti che vengono poi inoltrati verso i ricevitori. Da notare che il tutto viene fatto con indirezione, in questo caso a due hop.

La comunicazione avviene solo quando c'è almeno qualcuno interessato a quel contenuto: quindi chi invia i dati, si acorge che esiste un trigger, e quindi almeno un ricevitore che necessita di quei dati. I dati non vengono inoltrati comunque anche se non c'è il trigger per quella risorsa, in questo caso la comunicazione non avviene perché non c'è nessun interesse verso quel contenuto, quindi neanche lo si trasmette.

Ovviamente si ha il beneficio dato dall'indirezione: il mittente non conosce a chi manda il contenuto, quindi privacy e sicurezza sui ricevitori, oltre al fatto che non deve conoscere tutti i destinatari e gestirli.

Le primitive di questo servizio sono tre:

- Inserimento di un trigger (solo da parte dei ricevitori)
- Rimozione del trigger (opzionale) in quanto è possibile gestire i trigger in modalità soft state
- Invio di un pacchetto (solo da parte dei sender)

I trigger sono installati con filosofia soft state, quindi devono essere periodicamente rinfrescati dai ricevitori, altrimenti dopo un po verranno rimossi.

Dopo aver messo su il sistema secondo queste ipotesi, che cosa è possibile fare? È appunto possibile risolvere diversi problemi che con lo schema diretto punto-punto sarebbero difficili da risolvere.

### Struttura applicativa

È necessario un meccanismo che permette di installare questi trigger, rimuoverli, trovare il punto dove metterli, ed il tutto fatto a livello applicativo. Quindi avere a livello logico, a livello 7, una sorta di

infrastruttura di Overlay che sono i nodi che gestiscono il servizio gestiscono i trigger a livello applicativo.

Una cosa importante che è diversa dallo schema classico è che i trigger, sono a tutti gli effetti delle informazioni di routing che i ricevitori iniettano nella rete per farsi arrivare i dati che gli interessano. In particolare sono i ricevitori che (a livello 7) installano i trigger, quindi sono loro che sono responsabili per settare e mantenere le tabelle di routing che è una filosofia molto diversa da quella dell'architettura classica, dove invece le tabelle di routing vengono costruite facendo girare dei protocolli di Routing e non sono certo gli utenti a gestire il routing nella rete, ma sono gli amministratori di rete.

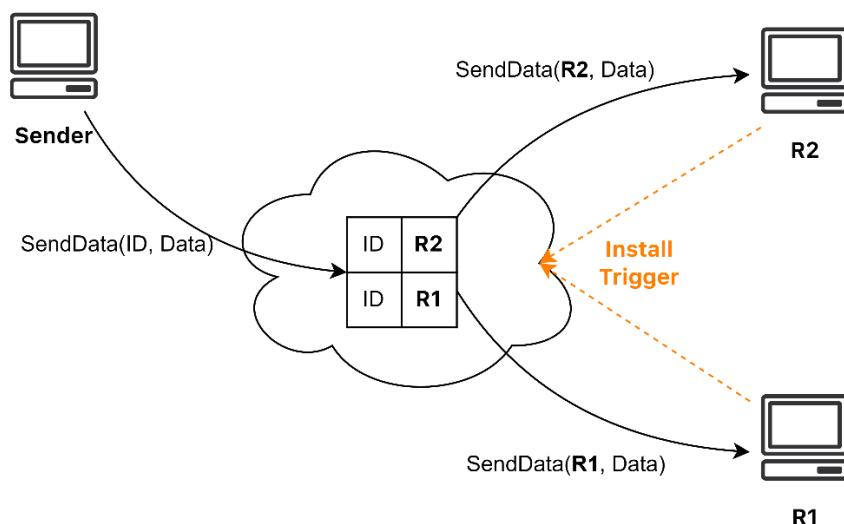
### Gestione della mobilità

Con I3, è possibile osservare la risoluzione al problema della gestione della mobilità

Il meccanismo è il seguente: i ricevitori sono responsabili di aggiornare nella rete l'informazione che contiene la loro posizione più recente (cosa che già avviene nelle reti cellulari, nel routing indiretto, etc) con la fase di registrazione. Quindi se i ricevitori si accorgono di essersi spostati, sono responsabili di aggiornare nella rete l'informazione del fatto che si sono spostati, andando concettualmente ad installare un trigger (rimuovendo quello precedente, oppure soft state) indicando il nuovo indirizzo.

Con i benefici della privacy sugli utenti, sicurezza ed il fatto che il sender non deve mantenere stato sui receivers.

### Gestione del Multicast

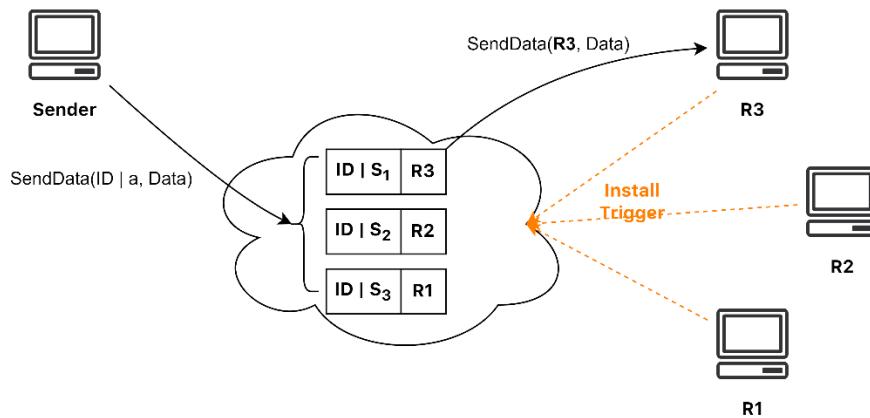


È anche possibile gestire il Multicast.

Di fatto, quello che otteniamo con uno schema di questo tipo è di rendere unicast e multicast unificati, come un unico forma di comunicazione. Il motivo è che tutti i ricevitori interessati ad un certo contenuto installano i loro trigger: che c'è ne sia uno solo, due o di più è essenzialmente la stessa cosa. Il Sender continua a mandare una volta sola in un'unica copia dei dati di quel contenuto. Quando poi i dati raggiungeranno il trigger, verranno duplicati e mandati in multicast a tutti i ricevitori che sono interessati a quel contenuto. Da notare che non è Multicast su IP ma è multicast a livello applicativo.

## Gestione dell'Anycast

L'altra primitiva di comunicazione che è facile implementare con IP Mobile è Anycast.



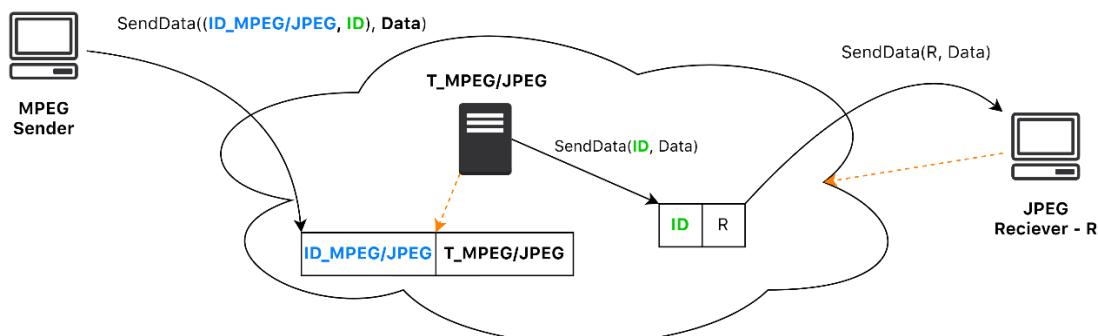
Di nuovo viene ottenuta in modo simile (si tratta sempre di far ruotare tutto attorno all'identificativo del contenuto), qui in un certo senso si invertono i ruoli di mittente e destinatario: i receiver nello scenario anycast, sono i nodi che possono offrire una certa risorsa, quindi, sono dei server equivalenti che rispondono alle stesse richieste, il sender invece è colui che manda una richiesta per un certo contenuto.

I receiver installano dei trigger dove specificano se stessi, quindi i loro indirizzi diversi uno dall'altro, l'identificativo del contenuto che possono offrire ed eventualmente anche dei cosiddetti qualificatori anycast, cioè aggiungono al fatto di poter offrire il contenuto ID anche delle informazioni che poi possono essere utili per andare a scegliere uno di questi server piuttosto che un'altro.

Un Sender che manda una richiesta per un contenuto, lo fa sempre con l'utilizzo di un identificativo ID. A questo punto la richiesta verrà inoltrata ad uno solo dei server registrati presso il trigger. Sul dove mandare la richiesta, la scelta è pilotata dal campo ulteriore di qualificatore anycast che si trova nel trigger installato dai ricevitori, in senso astratto è possibile inserire tante cose che possono essere utili nella scelta del receiver specifico, ad esempio la distanza di quel server (posizione geografica o RTT), livello di carico delle CPU. Comunque qualcosa che aiuti a scegliere il server più conveniente.

## Servizi componibili

Nulla toglie di utilizzare l'indirezione a più livelli. Ad esempio, un meccanismo che fa uso di due livelli di indirezione invece che uno, è quello di offrire dei servizi componibili.



Si suppone di avere uno scenario di questo tipo: ci sono utenti che inoltrano contenuti con un certo ID, e ricevitori che lo richiedono in un formato diverso. Quindi ad esempio, un traffico multimediale con un certo ID, in formato MPEG, ma i ricevitori desiderano averlo in JPEG. Quindi c'è il problema della transcodifica.

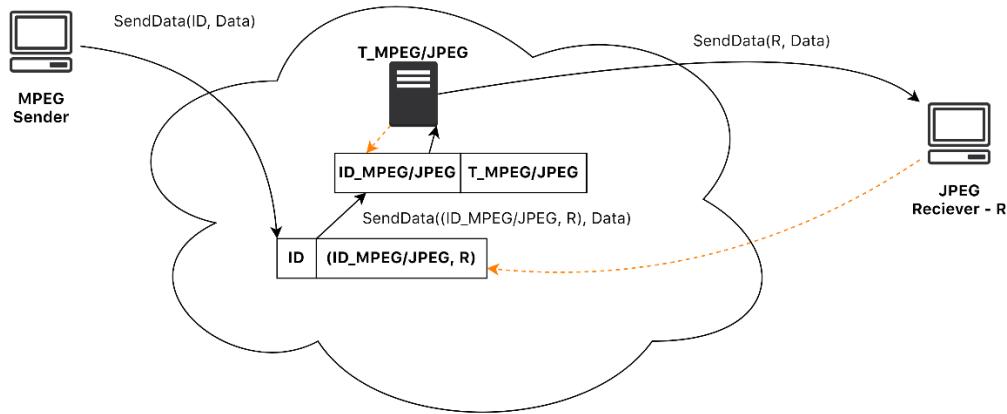
È possibile ostruire un sistema con indirezione a due livelli che permetta tutto questo. Eh. L'idea è che invece di avere un singolo ID, si ha uno stack di ID. Quindi una sequenza di ID in catena.

Nel primo scenario, il ricevitore installa il classico trigger, dove specifica ID e il suo indirizzo. Il sender, immette i dati indicando uno stack di ID: al top dello stack è indicato l'ID per la transcodifica, sotto questo ID c'è l'ID reale dei dati che è quello che ha installato anche il receiver.

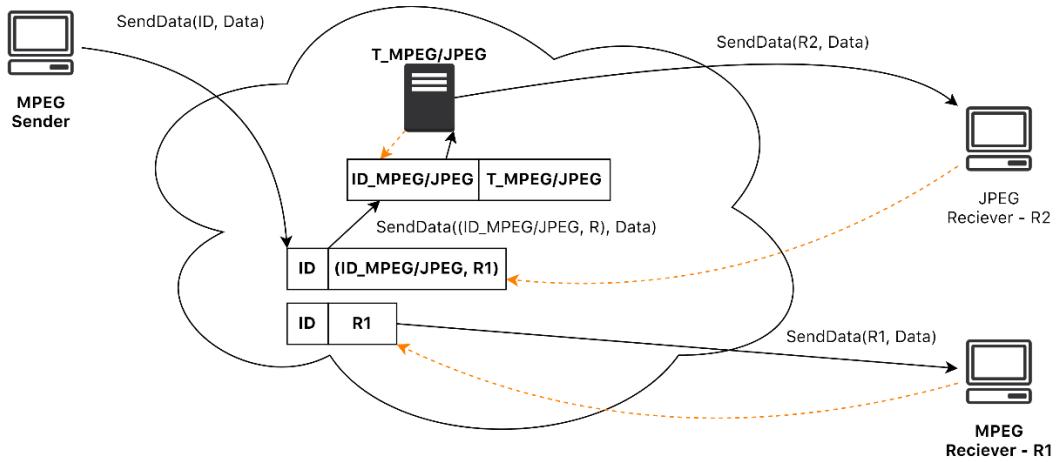
Il primo HOP che effettua il pacchetto inoltrato dal sender, arriva ad un transcodificatore che transcodifica i dati in base all'ID (quello sul top). Il secondo HOP è al trigger classico.

La cosa è gestita lato mittenti, quindi sono i mittenti che decidono di fornire il loro contenuto in un formato diverso usando questo schema

Un caso più interessante è il caso in cui invece il servizio è specificato non dai mittenti, ma dai ricevitori e quindi il tutto viene gestito lato ricevitori. Questo è interessante in quanto permette anche di mescolare ricevitori eterogenei, quindi alcuni vogliono il contenuto in formato JPEG, altri MPEG, altri PNG, etc.



In questo caso, i ricevitori installano dei trigger dove questa volta lo stack è gestito da loro. Quindi il ricevitore installa un trigger per un certo contenuto ID, inserendo lo stack di ID da cui dovrà passare. Il sender invia i dati con lo stesso ID. Quando i dati arrivano al trigger installato dal ricevitore, questo passerà prima tutti gli stack di ID per poi essere mandato al ricevitore tramite indirizzo.



Questo meccanismo è più efficiente, perché è possibile mescolare ricevitori eterogenei e supportare quindi anche ricevitori che invece sono ben contenti di ricevere il formato nativo della sorgente.

La morale di tutto questo è che con il meccanismo di indirezione si risolvono in modo elegante una serie di problemi. In particolare il multicast, mobilità, anycast, servizi componibili.

I vantaggi:

- trasparenza per i mittenti che non devono conoscere i loro ricevitori
- privacy
- sicurezza
- Non c'è nessuna complessità nei mittenti che si limitano a mandare i dati senza preoccuparsi della popolazione e della eventualmente eterogeneità dei ricevitori.

Svantaggi:

- Prestazioni: in termini di prestazioni potremmo star facendo delle cose poco efficienti. Tutte le volte che si utilizza un meccanismo indiretto rispetto a delle cose dirette, si perde in efficienza (tipo hop a triangolo)
- Punti di vulnerabilità: il posto in cui si immagazzinano i triggers è un punto debole del sistema che può essere attaccato, spesso compromesso, può essere un single point of failure. L'installazione alla rimozione dei trigger e espone a problemi di sicurezza. Se crolla la rete di Overlay che fornisce il servizio di trigger poi si riesce più a far funzionare nulla.

## Hash Table Distribute - DHT

Come si costruisce la rete di overlay che offre il servizio di indirezione? Cioè, dove fisicamente vengono installati i trigger, e come si fa in pratica a trovare i nodi che gestiscono i trigger, installarne di nuovi o rimuoverli?

Per capire come funziona è necessario introdurre un oggetto fondamentale nei sistemi distribuiti del networking moderno che sono le DHT distributed hashtable. Le distribute Hash Table offrono un meccanismo di insert/retrieve che è esattamente quello di cui si ha bisogno per mettere in piedi l'infrastruttura di direzione, in particolare l'installazione e la rimozione dei trigger.

Visto che le distributed hash table sono in realtà più generali di questo diciamo più in generale di cosa si tratta.

Questa non è solo utilizzata nella gestione della rete overlay per i trigger della infrastruttura di intrezzazione, è solo una loro possibile applicazione. Le distributed hash table hanno un sacco di altri utilizzi. Tanto per dirne uno, (che però non c'entra niente con l'indirezione) che è fondamentale e riguarda la gestione dei Tracker in BitTorrent: come si gestiscono questi tracker? Se li si mette tutti in uno stessopunto, la cosa è vulnerabile; se qualcuno chiude quel nodo, tutti i file non sono più scaricabili con bittorrent e questo uccide completamente la distribuzione con BitTorrent. Se però avessi un modo di distribuire i tracker tra tanti nodi che offrono la volonità di ospitare i tracker, si ottiene un sistema molto più robusto e inattaccabile, cioè a questo punto non riesco più a tirar giù il sistema, perché dovrei spegnere tutti i nodi che ospitano i tracker e la cosa può essere infattibile. Quindi se si costruisce una DHT, ad esempio, ospitare i tracker di BitTorrent è un'enorme vantaggio rispetto a uno schema centralizzato dove uno o pochi server sono responsabili di tutti i Tracker.

Una DHT è semplicemente una hashtable divisa tra tanti nodi, quindi spezzettata in tanti sotto tabelle, dove ogni pezzo di tabella viene distribuito ad un nodo diverso. Il vantaggio di fare questo è la gestione della struttura dati in modo distribuito e decentralizzato che ha un sacco di vantaggi tra cui essere più robusto, tollerante ai guasti, etc.

Il tutto è di nuovo gestito a livello applicativo, di fatto si utilizzano dei sistemi terminali, per costruire una rete overlay logica dove dentro i sistemi terminali si vanno ad inserire pezzi della hashtable complessiva.

Le primitive fondamentali in un sistema di questo tipo sono sostanzialmente due:

- Insert: avere quindi un'interfaccia base che permette di inserire un nuovo valore: quindi una chiave col suo relativo valore.
- Lookup: cioè data una chiave, trovare il nodo che gestisce quella chiave e quindi il valore

Entrambe le primitive hanno in comune l'operazione di base cioè quella di, data una chiave in ingresso, rintracciare il nodo che gestisce la chiave.

I nodi si prenderanno ciascuno un pezzo della tabella, quindi non si replica la stessa tabella intera su tanti nodi. I nodi fisici saranno poi connessi in qualche modo tra loro attraverso una rete Overlay, che tipicamente non è a maglia completa, perché la cosa può essere poco scalabile, complessa anche da gestire per il motivo, ad esempio, che i nodi vanno e vengono. Quindi la rete è dinamica, ci sono dei nodi che entrano, dei nodi che escono e bisogna anche gestire questi problemi.

Quindi ogni nodo conosce una piccola parte di tutti gli altri, un sottoinsieme gestibile di tutti gli altri. Per questo motivo che le DHT bisogna organizzarle in modo efficiente, costruendo la rete Overlay in maniera furba.

## Obiettivi di progetto

Quindi, nel momento in cui si progetta una DHT, bisogna avere come obiettivi di progetto quello di ottenere un mappaggio il più possibile efficiente tra chiavi e nodi fisici. Quindi la rete overlay che si costruisce deve avere queste caratteristiche:

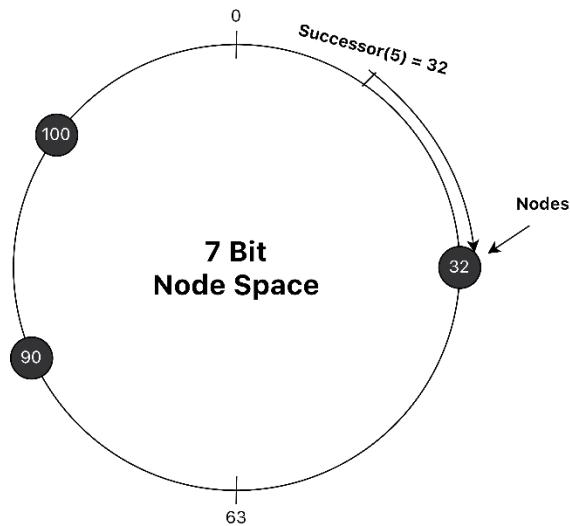
- **Avere un diametro piccolo:** Il diametro di una rete è il massimo numero di hop che devo attraversare per andare da una parte all'altra. Visto che le richieste passano di nodo in nodo finché non arrivano nel nodo che contiene la responsabile di quella chiave, si vuole che con pochi hop, porti nel peggiore dei casi

- **Fanout ridotto:** evitare di avere una conoscenza totale della rete, quindi non conoscere tutti quanti, avere un numero di nodi conoscenti ridotto (chiamato fanout)

Questi due criteri di progetto sono un po in contraddizione, perché si vuole contemporaneamente far sì che ogni nodo conosca un numero limitato di altri nodi, ed arrivare in pochi hop ovunque nella rete overlay.

### Chord

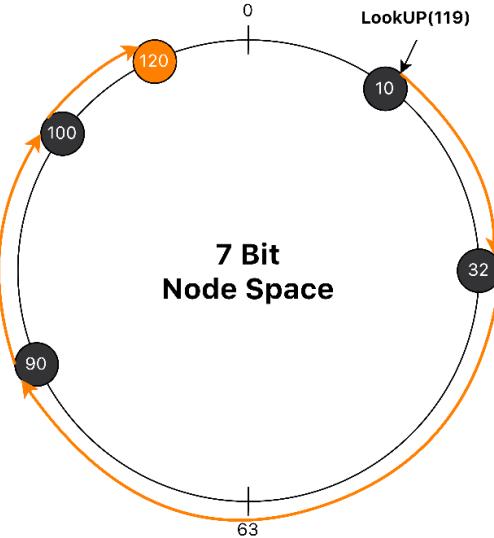
Chord permette quindi di costruire la rete di Overlay di una DHT in modo da ottenere contemporaneamente, un basso diametro e un basso fanout.



L'idea è la seguente, si organizzano le chiavi logicamente su un cerchio con uno spazio di m bit. Dopodiché ci sono dei nodi fisici che entrano nella DHT e che hanno la responsabilità di gestire un sottoinsieme di chiavi.

Da notare che tipicamente lo spazio delle chiavi è molto grande, a differenza del numero di nodi dentro la rete overlay che è più piccolo. Quindi si avranno molti meno nodi fisici che gestiscono le chiavi. Inoltre, ogni nodo gestisce tutte le chiavi fino al nodo fisico precedente.

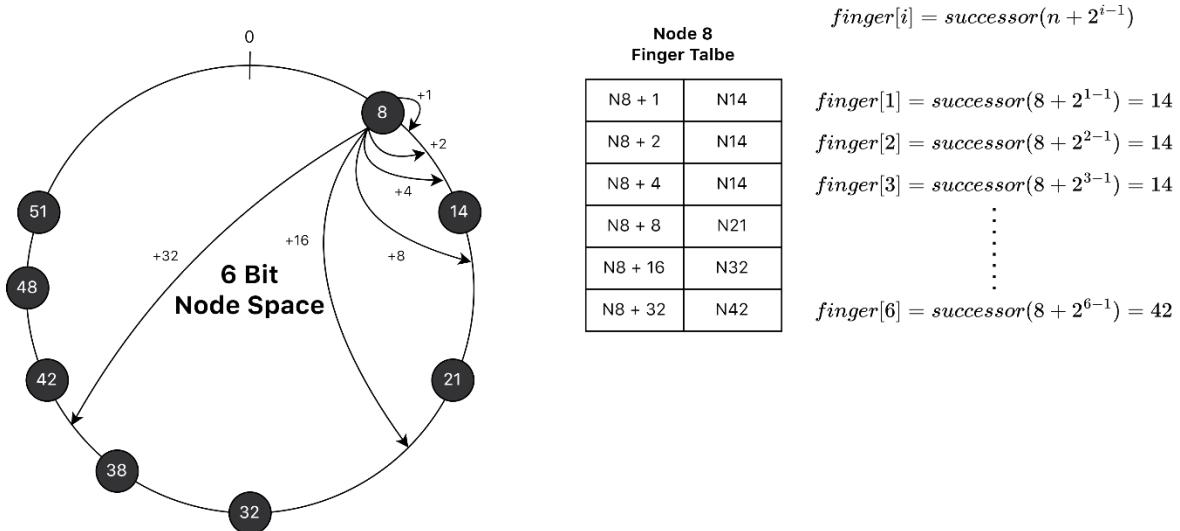
La funzione che logicamente, data una chiave, restituisce il nodo che gestisce quella chiave viene chiamata successor(K). L'algoritmo funziona in questo modo: ruota attorno all'anello cercando per ogni nodo fisico, finché non si incontra il nodo che possiede la chiave K.



Questa idea è una base del lookup, ma ha una complessità di instradamento nell'ordine di N, dove N è il numero di nodi fisici su cui è suddivisa la DHT. Complessità in termini di spazio è costante O(1), in quanto ogni nodo deve conoscere solo il successivo.

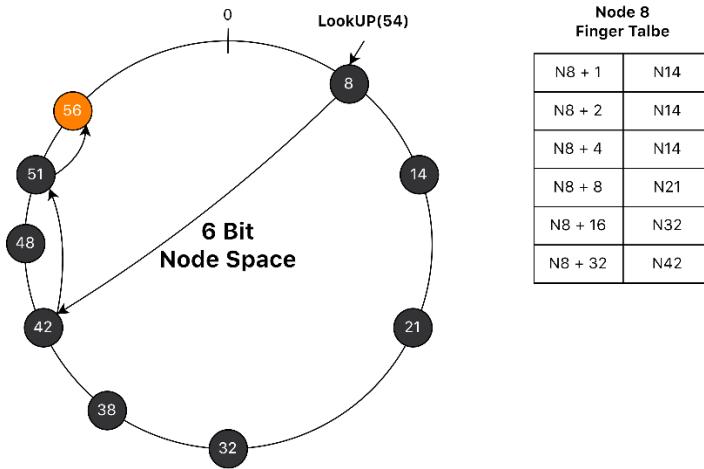
Chord si è inventato un meccanismo molto furbo per abbattere drasticamente il numero di hop che si effettuano durante un lookup, in particolare, trasformarlo da ordine n ad ordine log(n) e contemporaneamente mantenendo una complessità in termini di spazio costante.

L'accelerazione dei lookup si ottiene in questo modo: ogni nodo non conosce solo il prossimo nodo, ma conosce un'insieme di nodi spaziati lungo l'anello che cresce in potenze di due. Ogni nodo ha una conoscenza accurata dalla parte di anello seguente, più vicino a lui e poi gradualmente sempre più vaga. Questa vaghezza non è un male, anzi, consente con pochi hop di arrivare dappertutto sull'anello grazie a un meccanismo di approssimazioni successive.



La lista dei nodi a cui un nodo è collegato viene chiamata Finger Table. Quindi un nodo ha una serie di vicini, più precisamente 'm', dove 'm' è di fatto il numero di bit della chiave, quindi la finger table è composta di 'm' entries. La finger table accelera enormemente la l'instradamento permettendo di arrivare ovunque nell'anello nell'ordine di log(n)

Però, visto che tipicamente in queste posizioni non ho dei nodi fisici, quello che si memorizza nella Finger Table è il primo nodo che si incontra a partire dalla potenza di due ruotando lungo l'anello in senso orario (successor).



Questo approccio ha solo qualche piccolo problema, che è dovuto al fatto che i nodi non sono statici, ma entrano ed escono dal sistema, cosa purtroppo che succede, nei sistemi distribuiti peer to peer, i nodi entrano ed escono di continuo.

Quando il nodo esce dal sistema è un casino, in quanto bisogna andare ad aggiornare tutte le finger table dei nodi della rete in modo da aggiornare la nuova situazione. Similmente, quando entra un nuovo nodo, diventerà responsabile di un certo insieme di chiavi e quindi bisognerà andare a cambiare anche le informazioni degli altri nodi della rete.

Quindi c'è un overhead dovuto a quello che si chiama Churn, il fatto che nodi entrano ed escono dal sistema. Prendere le chiavi memorizzate in un certo punto, spostare la attribuirla ad un'altro modo e viceversa. Quando un nodo entra, bisogna assegnargli un certo numero di chiavi e aggiornare le finger table. Allora le cose non sono drammatiche, anche per il fatto che se io non aggiorno. Il meccanismo funziona, può essere lo stesso affidabile, però all'aumentare del Churn (più cresce il rate di uscita di ingresso di nodi) il sistema diventa instabile.

# Modelli di TCP

Questa sezione ha l'obiettivo di fare delle considerazioni analitiche su TCP e in generale sul controllo di congestione.

## Ripasso

TCP è il protocollo di trasporto che detta il rate con cui le sorgenti mandano traffico nella rete, quindi è una componente cruciale in Internet. In passato ci si è chiesti come si possano studiare le prestazioni di TCP. Quindi, capire come si comporta una rete che raggiunge certe condizioni, in un certo scenario, dove viene appunto utilizzato TCP

Per studiare le prestazioni di TCP ci sono vari approcci possibili, ognuno ha i propri vantaggi e svantaggi:

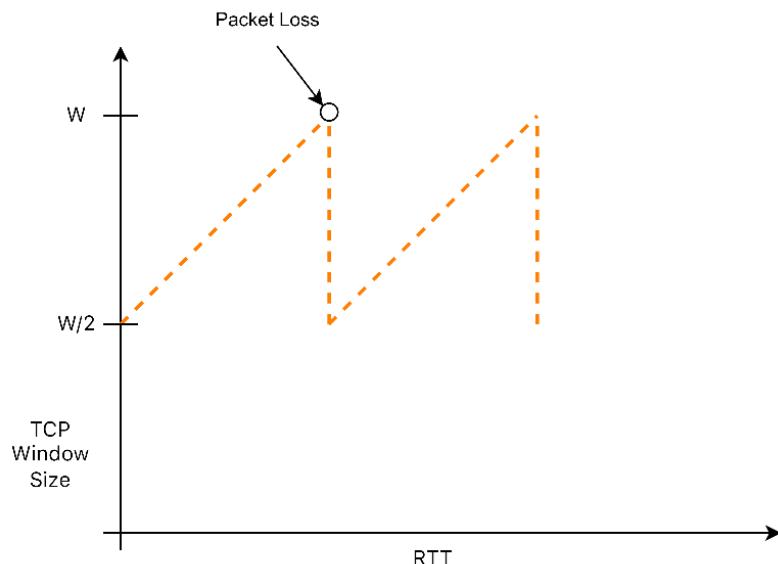
- Approccio simulativo/emulativo: ovvero si vanno a simulare nei dettagli una rete andando a simulare pacchetto per pacchetto che cosa fanno i flussi che attraversano la rete.  
Per approccio emulativo invece si intende che è possibile utilizzare questo approccio andando a utilizzare la vera implementazione di TCP nei sistemi operativi attuali, emulando il tutto via software in una ipotetica rete su cui far girare le connessioni TCP. Questo è possibile in quanto esistono vari strumenti software che permettono di emulare una rete, quindi di costruire in maniera virtuale una rete fittizia su cui poi i nodi mandano i loro pacchetti come fosse una rete vera.  
Questi software permetterò anche di definire una certa topologia, certe capacità dei link, certe latenze che subiscono i pacchetti, una certa probabilità di perdita dei pacchetti, etc.. È un approccio interessante perché fedele al vero comportamento di TCP, il problema però è che è costoso questo approccio in termini di tempo e di sforzi, perché bisogna simulare o emulare una rete in tutti i suoi dettagli. E quando si hanno delle reti molto grosse oppure dei link di capacità molto alta è computazionalmente molto costoso fare tutto ciò e la cosa scala poco.
- Modelli approssimati analitici: se non ci si vuole imbarcarsi in simulazioni o emulazioni perché troppo costose, è possibile ricorrere a dei modelli approssimati analitici. C'è ne sono essenzialmente di due tipi:
  - Modelli deterministici: qui si fanno delle ipotesi molto semplificative del vero comportamento di TCP che però hanno il vantaggio di essere molto semplici, molto veloci da risolvere che però, appunto, ignorano molte specifiche del TCP. Ignorando tutti i dettagli che è molto complicato ma bisogna farne dicendo delle ipotesi semplificative.  
Inoltre, questi modelli assumono tipicamente che nella rete si aggiunga un cosiddetto steady-state, quindi una situazione di equilibrio, una condizione stazionaria, soggetta ad un certo traffico TCP in ingresso che raggiunge una condizione di equilibrio. Quindi non è possibile usare questi modelli per studiare il transitorio di una rete, ma solo il suo comportamento di equilibrio.
  - Modelli fluidi stocastici: modelli sempre approssimati di TCP che però utilizzando tecniche fluido stocastiche, quindi approssimando il traffico che originariamente sarebbe pacchetto come un fluido ipotetico, riescono a modellare anche il comportamento transitorio della rete includendo anche effetti stocastici che i modelli deterministici invece non consideravano. Questi modelli fluidi stocastici sono di nuovo della stessa natura di quelli deterministici, quindi delle versioni semplificate, approssimate di TCP però scalabili, quindi al vantaggio di

poter studiare anche reti di grandi dimensioni e di alta capacità e perché scalano bene con dimensioni della rete crescente.

Ci focalizziamo su modelli approssimati, lasciando stare quelli deterministici.

Un semplicissimo modello di TCP, il primo che è mai stato inventato e alla fine degli anni 90, ed ha avuto subito molto successo perché è estremamente semplice, ma cattura molto bene in modo approssimato una formula del throughput di TCP che è diventata famosissima: la cosiddetta **Square Root formula TCP**

Quindi la possibilità di trovare una formula che ci dia il Throughput di una connessione TCP conoscendo l'RTT e la probabilità di perdita media dei pacchetti. L'analisi è stata effettivamente fatta e curiosamente molto dopo che è stato standardizzato TCP. Quindi solo negli anni 80, verso la fine del secolo scorso, i ricercatori hanno trovato la relazione che lega il Throughput con l'RTT e la probabilità di perdita. La cosa è un po' curiosa, perché prima si è standardizzato un protocollo e poi diversi anni dopo si è capito che prestazioni ottiene.



Il modello più semplice che permette di capire la relazione che c'è tra il Throughput con l'RTT e la probabilità di perdita, è basato su un modello deterministico molto semplice, un modello di evoluzione della finestra nella fase di congestione avoidance che è riportata su un grafico dove in asse X c'è il tempo misurato in RTT, e sull'asse Y l'evoluzione della finestra, che, in assenza di perdite, cresce in modo lineare, e al riscontro di una perdita, viene dimezzata la finestra. Quindi le perdite avvengono solo quando la finestra raggiunge un suo valore massimo  $W$  per ora misterioso, nel senso che non conosciamo questo valore di  $W$  però assumendo di conoscerlo. E assumendo che la perdita avvenga quando si raggiunge questo valore massimo, poi la finestra evolve con questo andamento a dente di sega. Quindi cresce in modo lineare, poi va giù bruscamente da  $W$  a  $W/2$ , riprende a salire, eccetera. Questa ovviamente è una visione semplificata nella realtà, le perdite non sono così regolari, quindi un comportamento vero di TCP, è sempre a dente di sega, ma è un dente di sega più irregolare, dove le perdite avvengono più casualmente nel tempo.

Bisogna anche qui, fare un'analisi per ciclo, cioè considerare un ciclo tipico, quindi uno solo di questi denti, perché questo è rappresentativo poi di tutto quello che succede anche al di fuori di questo particolare ciclo, perché tutti i cicli sono uguali.

Per trovare il Throughput di una connessione TCP, bisogna in sostanza:

- Trovare il numero pacchetti che sono trasmessi dentro un ciclo
- Trovare la durata di un ciclo

Dopo basta prendere il numero di pacchetti trasmessi dentro un ciclo e dividere per la durata del ciclo trovando così il Throughput. Bisogna adesso capire quanti pacchetti si trasmettono dentro uno di questi denti di sega. Il numero di pacchetti che trasmetto in ogni RTT è all'inizio  $W/2$ , poi  $W/2 + 1$ , fino a  $W$ , questo può essere calcolato dalla serie:

$$\begin{aligned} \frac{W}{2} + \left( \frac{W}{2} + 1 \right) + \dots + W &= \sum_{n=0}^{W/2} \left( \frac{W}{2} + n \right) \\ &= \left( \frac{W}{2} + 1 \right) \frac{W}{2} + \sum_{n=0}^{W/2} n \\ &= \left( \frac{W}{2} + 1 \right) \frac{W}{2} + \frac{W/2(W/2+1)}{2} \\ &= \frac{3}{8} W^2 + \frac{3}{4} W \\ &\approx \frac{3}{8} W^2 \end{aligned}$$

Di questi due termini, teniamo quello con contributo più alto.

La durata di un ciclo inceve è banalmente  $W/2$ .

Un'ulteriore cosa che serve, è legare tutto ciò alla probabilità di perdita dei pacchetti. In questo modellino assumiamo che venga perso un pacchetto quando la finestra raggiunge  $W$ . Si fa questa ipotesi semplificativa, per cui io perdo un singolo pacchetto, quando raggiungo il massimo ( $W$ ) poi si torna giù e da lì in poi non si perde più niente. Questo significa che ho un pacchetto perso per ciclo.

Che significa un pacchetto perso per ciclo. Quindi la probabilità di perdita in questo modellino è:

$$\frac{1}{\frac{3}{8} W^2} = \frac{8}{3} W^2$$

È anche possibile leggere questa formula nell'altro senso, cioè legare  $W$  alla probabilità di perdita e viene che  $W$  è uguale alla radice quadrata di  $8/3$  per  $p$  loss.

Allora, avendo questi pezzi, mettendo tutto insieme, si trova la funzione che indica il throughput medio:

$$B = avg\_throughput = \frac{\frac{3}{8} W^2}{\frac{W}{2}} = \frac{3}{4} W \frac{packets}{RTT}$$

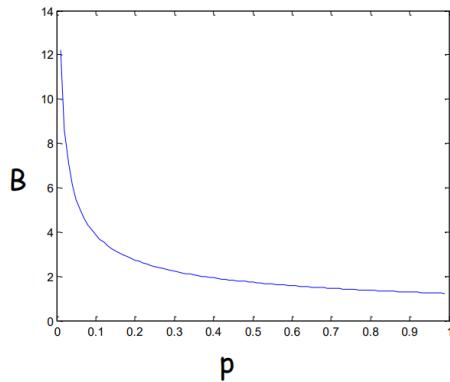
$$B = avg\_throughput = \frac{1.22}{\sqrt{P_{loss}}} \frac{packets}{RTT}$$

Ci si può convincere che emergono queste due dipendenze molto interessanti:

- Il Throughput è inversamente proporzionale al RTT
- Il Throughput è inversamente proporzionale alla radice quadrata di  $P_{loss}$

Il fatto di essere inversamente proporzionale alla radice quadrata di  $P_{loss}$  ha portato questa formula ad essere ricordata poi con il nome di Square Root formula, che, appunto, ricorda il fatto che il Throughput dipende dalla radice quadrata di  $P_{loss}$  con proporzionalità inversa.

Il modello è molto semplificato perché ignora la fase di Slow start, i time-out ed il fatto che la finestra possa avere un limite superiore e quindi saturare ad un certo valore, però la cosa fondamentale è che cattura comunque abbastanza bene il comportamento di TCP.



Possiamo plottare ora B in funzione di P. E no, pare che viene questo andamento qua. Questo ci suggerisce che quando la probabilità di perdita è bassa, ottengo dei Throughput elevati, mentre quando la probabilità di perdita aumenta, intuitivamente il Throughput va sempre peggio.

## PFTK

Questo modello baso si può estendere? La risposta è sì e questo è stata fatta in un lavoro molto famoso **J. Padhye, V. Firoiu, D. Towsley, J. Kurose, "Modeling TCP Throughput: a Simple Model and its Empirical Validation", Sigcomm 1998.**

I ricercatori hanno esteso questo approccio complicandolo un po, portando ad una formula più completa, dove gli autori hanno essenzialmente incluso l'effetto dei time out e anche l'effetto della finestra massima:

f

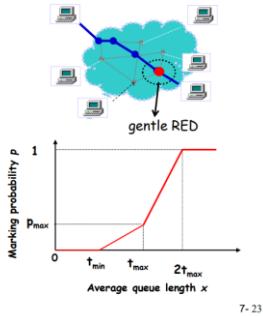
Ora bisogna fare un passo successivo, che è il seguente: finora abbiamo preso in considerazione un singolo flusso di TCP, assumendo di conoscere, di misurare o di avere comunque nota la risposta della rete che si traduce in pratica nella probabilità di perdita e nel RTT (li abbiamo assunti noti)

Adesso il problema interessante è il seguente: vogliamo studiare in modo analitico, la risposta della rete soggetta al traffico generato da tanti flussi TCP. In termini tecnici si dice che passiamo da un'analisi ad anello aperto ad un'analisi ad anello chiuso. Vuol dire che prima era ad anello aperto perché prendevamo la risposta della rete nota, e la mettevamo dentro il modello di che ci forniva il risultato. Adesso dobbiamo, diciamo chiudere l'anello e cercare di studiare sistema complessivo, cioè, il traffico di tante connessioni TCP che interagiscono con la rete e la rete risponde a questo carico con certa probabilità di perdita ed RTT.

## Ripasso di gestione della coda RED

- prob.  $p$  di scarto/mark dei pacchetti dipende da lunghezza media della coda  $x$   
 $\rightarrow p = p(x)$

- più in generale: active queue management (AQM)

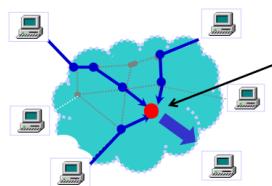


7- 23

Per effettuare questa analisi, assumiamo che nella rete ci sia un router RED. In questo router c'è un profilo chiamato Gentle Red. Questo per evitare una transizione troppo brusca tra  $T_{\max}$  ad 1 ed evitando una discontinuità attorno al valore di  $T_{\max}$ . Questo perché avere queste discontinuità si è capito che è una cosa un po troppo brusca e che può dare dei problemi di stabilità al controllo. Quindi è meglio fare un profilo più dolce di questo tipo che si chiama gente.

Il fatto di assumere che la rete sia gestita da red, ci aiuta ad analizzare la rete perché data una occupazione della coda, abbiamo una probabilità di perdita dei pacchetti e tra l'altro una probabilità di perdita dei pacchetti che ci piace perché, indipendentemente pacchetto per pacchetto e perdiamo con una certa probabilità  $p$ , questo fatto di perdere i pacchetti indipendentemente pacchetto per pacchetto ci aiuta nella costruzione del modello. Se i pacchetti magari introducessero perdite correlate a burst, sarebbe decisamente più difficile.

## Comportamento a singolo collo di bottiglia (bottleneck)



- router di bottleneck:
- un link di uscita pienamente utilizzato
- tutti flussi TCP che lo attraversano subiscono stessa prob. di perdita
- D: i flussi ottengono lo stesso throughput?

$$\sum_i B_i(p, RTT_i) = C$$

$C$  - link bandwidth

$B_i$  - throughput of flow  $i$

7- 24

L'analisi ad anello chiuso si fa, nel caso più semplice, assumendo che ci sia un unico link congestionato (un unico collo di bottiglia) dove possono avvenire perdite di pacchetti. Il resto della rete invece è privo di perdite e anche privo di particolari ritardi. (Poi ovviamente si può estendere al caso di  $N$  Colli di bottiglia ma non cambia molto)

Quindi ci sarà una certa probabilità di perdita introdotta da questo router red e visto che il Router non introduce la stessa priorità di perdita su tutti i pacchetti, intuitivamente tutte le connessioni di TCP saranno soggette tutta la stessa probabilità di perdita. Un'insieme di connessioni, tutti attraversano lo stesso collo di bottiglia che introduce per tutti i flussi. La stessa probabilità di perdita domanda, i flussi avranno di conseguenza tutti lo stesso Throughput? Avendo che la squared root formula dimostra dal RTT, può benissimo succedere che queste varie connessioni che attraversano il

collo di bottiglia abbiano RTT diversi. Di conseguenza, anche se hanno la stessa probabilità di perdita, perché attraversano tutto lo stesso red congestionato, hanno RTT molto diversi e quindi in generale il Throughput è diverso tra loro.

Un'equazione fondamentale che si può scrivere, quella che dice che la somma dei tutti le connessioni è uguale alla capacità del link, la capacità fisica di trasmissione del link.

Allora che perché è utile questa equazione? È utile perché poi la possiamo usare per risolvere il sistema trovare tutte le incognite che mancano. Quindi questa equazione, assieme ad altre che adesso vedremo, vengono poi messe tutte assieme in un unico sistema di equazioni che si risolve e permette di trovare sia il Throughput delle connessioni, sia la probabilità di perdita introdotta da RED e sia il RTT dei vari flussi.

## Modello e soluzione

### modello

$$\begin{aligned} p &= p(x) \quad (\text{AQM}) \\ RTT_i &= A_i + x / C \quad (\text{round trip time}) \\ \sum_i B(p, RTT_i) &= C, \quad \text{per } i = 1, \dots, N \end{aligned}$$

### soluzione a "punto fisso" (fixed point problem) per $x$

### $x$ usata nel calcolo di $RTT_i$ e $p$

### unica soluzione purchè $B$ monotona e continua rispetto a $x$

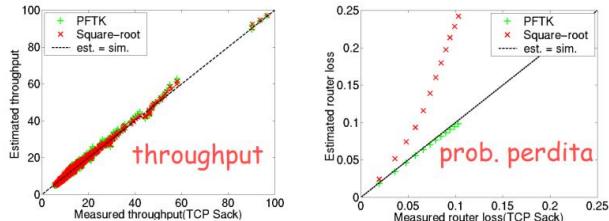
7- 26

Il modello è così formato:

- La funzione di perdita in funzione di  $X$  che dipende solo dal profilo red particolare che si utilizza.
- Il RTT della correzione  $i$ -esima. Questo è interessante: se facciamo l'ipotesi che la rete sia tutta scarica, tranne nel collo di bottiglia che invece introduce un punto di congestione, quindi anche un ritardo di accodamento significativo dei pacchetti. Il RTT lo possiamo esprimere in questo modo come una componente fissa  $A_i$ , dovuto solo a tempo di propagazione, che può essere diverso, flusso per flusso e poi la parte invece di latenza dovuta alla accodamento dei pacchetti, il router congestionato. Questa latenza si può calcolare in modo semplice con questa formula approssimata che è  $X/C$  che è appunto quanto impiega un pacchetto che arriva dentro questa coda mediamente riempita ad uscire fuori
- L'equazione che lega tutti i parametri, cioè che la somma di tutti i Throughput.

## Modello vs simulazione: singolo collo di bottiglia, flussi TCP infiniti

- capacità del link 4 Mbps e parametri RED assegnati
- 10-120 flussi TCP
- ritardi di prop. a due vie  $A_i = 20+2i$  ms,  $i = 1, \dots, N$



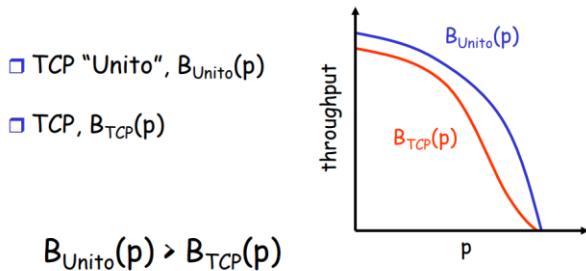
7- 27

Ci sono due domande a cui si può rispondere ora. La prima domanda è: ha senso inventarsi una versione di TCP più aggressiva, quindi che, data una certa probabilità di perdita, ottiene un Throughput più alto? Questa è la prima domanda. L'altra domanda interessante a cui si può rispondere col modello a collo di bottiglia e se è utile implementare FEC. Quindi la domanda è la seguente: se ho un collo di bottiglia congestionato e quindi introduce perdite, ha senso usare FEC?

Inventiamoci un TCP più aggressivo che chiamiamo TCP unito, che è una versione migliorata di quella standard il che significa che per una certa probabilità di perdita, la versione migliorata ottiene un Throughput più alto del TCP Standard.

Sono tanti i ricercatori in passato che hanno proposto dei metodi per aumentare il Throughput di una connessione TCP rispetto a quella standard, alla fine è un giochino relativamente semplice, no? Basta, ad esempio cambiare il modo con cui si aggiorna la finestra e realizzare un TCP con un comportamento più aggressivo.

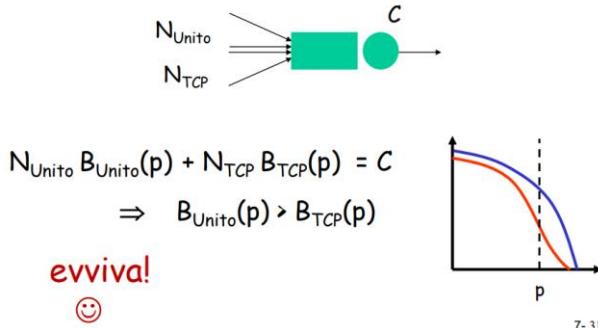
### Una nuova versione (migliorata) di TCP: TCP Unito



7- 30

La cosa non ha molto senso: supponiamo di avere due funzioni nella priorità di perdita, quella rossa che è la versione standard di TC, e quella blu che è quella modificata. Si nota che a parità di probabilità di perdita quindi il TCP rosso va sempre meglio, ha un Throughput migliore.

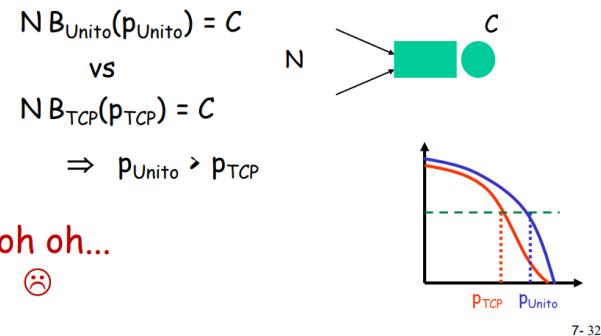
## Condivisione del collo di bottiglia con versione standard di TCP



Ora, nello scenario a singolo collo di bottiglia, si mettono un certo numero di connessioni con TCP classico mescolate con un certo numero di connessioni TCP nuove (TCP Unito). È stato misurato il sistema per simulazione o per via analitica, in modo da dimostrare che alla fine le connessioni della nuova versione unito vanno meglio. Il problema di tutto ciò, è che qui si è comparato un sistema in cui ci sono due metodi TCP diversi. Se ci fosse un TCP migliore, tutti adotterebbero quello.

Infatti il problema sorge quando questo mescolamento non è presente, e tutti utilizzando la versione TCP nuova.

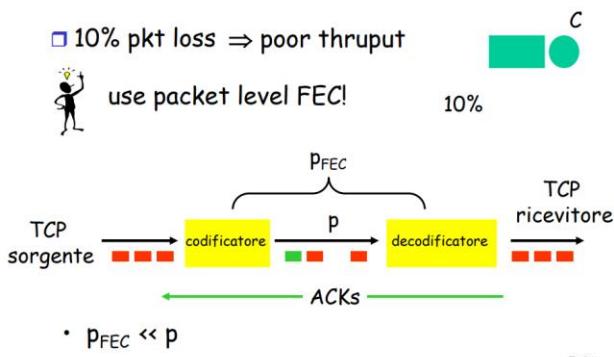
## Se si rimpiazza TCP con TCP Unito...



Ci si rende conto immediatamente e la cosa è ovvia, già solo scrivendo questa equazione qua che dice che la somma di tutti i flussi è uguale a  $C$ . Si può immediatamente trovare da questa equazione che il Thrughput  $B$ , di una generica connessione è uguale a  $C/N$ , ma è uguale a  $C/N$  per entrambi i casi. Questo vuol dire che il Thrughput è lo stesso in entrambi i casi. il troppo tempi nuovo.

Capiamo che le connessioni vecchie hanno lo stesso Thrughput di quello nuovo per il fatto che provocano nella rete delle probabilità di perdita diverse e in particolare il nuovo andrà a generare, essendo più aggressivo, crea più perdite nella rete, cioè genera una proprietà di perdita più grande di quella che generava il TCP originale.

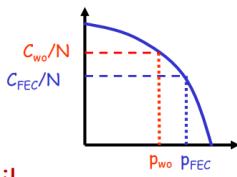
## Uso di FEC



Il motivo lo si spiega di nuovo, ragionando sul modello collo di bottiglia. Si suppone che si usi FEC che riduca la probabilità di perdita vera ad una priorità di perdita PFEC che ci si aspetta essere più bassa di P. Questo si suppone perché io recupero un po di pacchetti persi.

## Uso di FEC

- banda disponibile,  $C_{FEC} < C_{wo}$
- $B_{FEC}( ) = B_{wo}( ) = B_{TCP}( )$
- $N B_{wo}(p_{wo}) = C_{wo}$
- $N B_{FEC}(p_{FEC}) = C_{FEC}$
- $\Rightarrow p_{FEC} > p_{wo}$
- FEC riduce le prestazioni!



L'errore logico di tutto ciò sta nel fatto che si mandano più pacchetti.