

PROLOGO

Componenti

- Scheda di rete: dispositivo collegato direttamente alla scheda madre della calcolatore. Ha il compito di trasmettere e ricevere i dati oltre che di codificarli e decodificarli nel formato che supporta il link trasmissivo. Ad ogni scheda di rete e' associato un indirizzo MAC univoco a 48 bit.
- Connettore di rete: interfaccia standard che permette di collegare la scheda di rete al link trasmissivo.
- Link trasmissivo: mezzo fisico nel quale viaggiano le informazioni sotto forma di segnale analogico. Possono essere fili di rame, cavi coassiali, fibre ottiche o l'ambiente nel caso delle onde radio.

Infrastruttura di rete

L'infrastruttura di rete e' la struttura dei collegamenti di una rete.

- Punto-punto: connessione diretta tra due host
- Connessioni multiple:
 - Completamente connessa: esiste un collegamento diretto da ad ogni host. Massimo grado di ridondanza, molta tolleranza ai guasti ma alti costi.
 - Minimamente connessa: esiste un cammino da ogni host ad ogni host che eventualmente passa per nodi intermedi. Costi minimi ma grande rischio di partizionamenti di rete visto l'assenza di ridondanza.
 - Partizione di rete: due isole di host non connesse.

Topologia di rete

Le infrastrutture di rete possono avere diversi schemi di connessione dei dispositivi:

- Ad anello: ogni host e' connesso direttamente ad ogni host adiacente. Esiste quindi un cammino da ad ogni host, ed i cammini girano nella rete in senso orario od antiorario. La struttura presenta un grado di ridondanza, se cade un collegamento la rete risulta comunque minimamente connessa.
- A stella: rete minimamente connessa, ma gli host non sono connessi direttamente, bensì si connettono ad un dispositivo centrale (uno switch o un hub) che propaga il segnale in tutte le direzioni. Questa topologia presenta un alto rischio di partizioni di rete ed il dispositivo centrale e' un single point of failure.
- A bus: dispositivo connessi ad un bus comune a tutti. Anche in questo caso non c'e' ridondanza inoltre si presenta il problema di gestire l'accesso al mezzo di comunicazione condiviso.
- Ad albero: struttura gerarchica in cui ogni nodo puo' comunicare direttamente con i suoi figli diretti.

Nelle reti Lan o Pan in genere si adotta uno di questi schemi, nelle reti Wan e Man la struttura e' più complicata, presenta una combinazione di queste topologie ed e' detta a maglia.

Mezzo di trasmissione

- Cavi o fili metallici: permettono di codificare l'informazione digitale mediante variazioni della corrente elettrica. Possono essere doppi in rame intrecciati o cavi coassiali. Prestazioni (1-2 Gbit/s), basso costo ma suscettibili ad interferenze.
- Fibra ottica: fili sottili in vetro nei quali passano onde luminose che vengono codificate in base a variazioni della frequenza. Presentano le prestazioni migliori (migliaia di Gbit/s), subiscono poca interferenza ma hanno alti costi di giunzione.
- Segnali radio: trasmissione mediante onde elettromagnetiche nello spazio. Hanno il vantaggio di permettere una comunicazione in mobilità, ma presentano problemi circa le interferenze e gli errori, il problema dell'attenuazione del segnale con la distanza e la minor capacità di trasmissione.

Canali di comunicazione

Un singolo mezzo fisico trasmissivo può essere visto come un insieme di canali logici virtuali.

- Canale punto-punto: un canale logico viene riservato per far comunicare direttamente due host.
- Canale ad accesso multiplo (broadcast): un unico canale viene utilizzato da più dispositivi in modo che ciò che trasmette uno arriva a tutti. Presenta la problematica delle collisioni dovute alla sovrapposizione di segnali che transitano nello stesso tempo sul canale. Questa problematica e' molto più semplice da gestire nei canali punto-punto.

Commutazione a circuito

Per far comunicare due host si riserva un circuito di canali logici per tutta la durata della comunicazione. Prima di iniziare la comunicazione bisogna quindi creare il circuito ed una volta che inizia tutte le risorse sono dedicate per la comunicazione end-to-end. Il vantaggio e' un minor ritardo visto che le risorse sono allocate per tutto il tempo, tuttavia presenta diverse problematiche come il fatto che bisogna riservare risorse di fatto anche se non si sta sfruttando il circuito riservato, quindi questo risulterà in maggiori costi di comunicazione

Commutazione di pacchetto

I dati vengono spezzati in pacchetti, immessi nei canali ad accesso pubblico ed affidati a dispositivi chiamati commutatori di pacchetto che li faranno arrivare a destinazione, seguendo gli indirizzi del mittente e del destinatario. In questo modo la rete risulta utilizzata al meglio, si trasmette solo quando si ha necessità di farlo e non si allocano risorse ma si puo' utilizzare la rete solo se e' disponibile. Questo si traduce in maggiori ritardi di

comuncazione da un lato ma anche in minori costi dall'altro, in quanto si possono allocare piu' utenti allo stesso canale condiviso.

Servizi

I servizi offerti dalle reti nella commutazione a pacchetto dipendono dalle proprietà dei protocolli utilizzati:

- **Connection oriented:** i dati vengono spediti e ricevuti in ordine, inoltre arrivano senza errori e vengono rispediti i pacchetti andati perduti.
- **Connection-less:** i dati vengono affidati alla rete che per definizione è un servizio non orientato alla connessione, perciò può perderli, farli arrivare sbagliati o non in ordine.

Protocolli e stack ISO/OSI RM

Un protocollo è un insieme di regole sintattiche e semantiche. Devono definire il formato dei messaggi scambiati e le regole comportamentali per decidere quando e come inviare i messaggi. Un protocollo ha il compito di risolvere una problematica di rete. Classi di protocolli si occupano di gestire determinate funzioni della comunicazione. Queste classi di protocolli sono raggruppate in livelli, ogni livello gestisce una fase della comunicazione. Un livello x fornisce servizi al livello $x+1$ e richiede servizi offertogli dal livello $x-1$. La struttura a livelli standard per le reti di calcolatori è chiamata modello ISO/OSI RM:

- Livello applicazione: è la sede dei protocolli che permettono di far comunicare applicazioni remote. A questo livello le applicazioni si scambiano dati tramite i protocolli di questo livello, e questi pacchetti di informazione sono detti **messaggi**.
- Livello presentazione: risolve eventuali eterogeneità del formato dei dati tra i nodi della rete.
- Livello sessione: mantiene e gestisce lo stato attuale del collegamento tra due applicazioni remote.
- Livello trasporto: fornisce alle applicazioni i servizi orientati alla connessione e non, controlla la congestione della rete. In questo livello i messaggi vengono spezzati in **segmenti**.
- Livello rete: si occupa dell'instradamento dei **datagrammi** nella rete e di scrivere gli indirizzi di mittente e destinatario.
- Livello data link: si occupa della corretta trasmissione dei pacchetti tra nodi adiacenti della rete. A questo livello i datagrammi sono divisi in **frame**.
- Livello fisico: si occupa di definire le tecniche di codifica e decodifica dei dati e la loro trasmissione sul canale trasmissivo.

Ad ogni livello i protocolli aggiungono informazioni, andando ad incapsulare il messaggio dentro delle "buste". Queste informazioni verranno poi analizzate in parte dai nodi intermedi della rete ed infine dal destinatario. Ogni livello si occupa di analizzare e scrivere nelle buste i dati relativi al suo livello.

Integrazione di reti

La struttura a livelli permette di semplificare la visione della rete ai livelli più alti:

- Al livello fisico la rete è un segmento che collega dispositivi che condividono un mezzo di trasmissione con tecnologia in comune. Questo livello si occupa di codificare e decodificare i dati e di trasmetterli e riceverli.
- Al livello data link la rete diventa l'unione di più segmenti di rete, eventualmente anche con diverse tecnologie. Affronta il problema di determinare degli indirizzi per i dispositivi (MAC), di gestire l'accesso al canale condiviso, di evitare gli errori di trasmissione e gestire la comunicazione tra segmenti di rete con tecnologie differenti. A questo livello la rete diventa una rete locale.
- Al livello rete la rete si affronta il problema dell'integrazione di reti, organizzate secondo una visione gerarchica, quindi di fatto diventa una rete di reti. Si rende necessario l'utilizzo di altri indirizzi. Il protocollo IP definisce un nuovo schema di indirizzamento globale e gerarchico, che permette di identificare univocamente tutti i dispositivi di rete e allo stesso tempo la loro rete locale di appartenenza.
- Al livello trasporto esiste una rete di reti affidabile di tipo orientato alla connessione.
- Al livello applicazione si mette semplicemente a disposizione il servizio di comunicazione offerto dalla rete di reti.

Livello fisico

La scheda di rete codifica e decodifica i dati, converte i bit in segnali analogici e viceversa. La velocità di propagazione dei segnali è circa quella della luce, mentre la velocità di trasmissione dipende dal numero di bit/secondo che si riescono a trasmettere. Questo in particolare dipende dalle tecniche di codifica, ovvero da quanto a lungo un bit è rappresentato nel canale di trasmissione. Una velocità di codifica maggiore può causare un maggior numero di errori.

Livello MAC

Ogni calcolatore è dotato di una scheda di rete capace di codificare e decodificare i segnali. Ad ogni scheda di rete è associato un identificativo chiamato indirizzo MAC. Quando si inviano i frame sul link comunicativo si aggiunge l'indirizzo MAC del mittente e del destinatario. Infatti essendo il canale condiviso tutti riceveranno il messaggio, ma questo verrà passato ai livelli superiori solo da chi possiede l'indirizzo MAC corrispondente a quello del destinatario. A questo livello si vuole rendere il canale sicuro ed affidabile. Per fare ciò il frame viene arricchito con campi utili per identificare eventuali errori. Dopo la trasmissione il mittente fa partire un timer, se il pacchetto arriva al destinatario errato, esso non trasmetterà l'ACK di conferma. Di conseguenza se il mittente non lo riceve entro un certo lasso di tempo procederà a tentativi finché riuscirà a trasmettere correttamente il messaggio. I protocolli di questo livello dipendono generalmente dalla tecnologia utilizzata e sono:

- Ethernet: usato nelle reti cablate. Ha il vantaggio di poter ascoltare il canale in modo da trasmettere solo se non sono in atto altre trasmissioni. Quando un frame arriva a

destinazione, si manda immediatamente l'ACK di conferma così da ottimizzare l'utilizzo della rete. Inoltre può ascoltare il canale anche mentre sta trasmettendo, così può accorgersi subito di eventuali errori ed interrompere immediatamente la trasmissione. In questo caso vengono impostati dei timer casuali prima di riprovare la trasmissione.

- Wi-Fi: utilizzato per le reti wireless, permette di ascoltare il canale ma non se la trasmissione è in atto. Si basa sulla prevenzione delle collisioni dilazionando nel tempo i tentativi di accesso.
- Token ring: utilizzato per le reti a topologia ad anello. Trasmette solo chi possiede un frame speciale chiamato **Token** che viene fatto passare di volta in volta a tutti gli host della rete. In questo modo si annullano le collisioni, inoltre i messaggi vengono ricevuti da tutti in senso orario od antiorario, quindi se chi riceve un messaggio possiede il token, esso può interpretare questo segnale come ACK implicito. Il problema di questo protocollo è quando viene perso il token durante una trasmissione, infatti si deve gestire la rigenerazione del token in modo tale che non se ne crei uno per ogni host.

E' possibile comporre diversi segmenti di rete per far fronte alla degradazione del segnale che aumenta con la distanza. Esistono diversi dispositivi:

- Repeater: amplifica il segnale di arrivo e lo rigenera, tuttavia unisce segmenti al livello uno, quindi non può collegare segmenti con tecnologie diverse ed inoltre non gestisce l'accesso al canale quindi aumenta il dominio di collisione.
- Hub: repeater a più porte.
- Bridge: collega segmenti di rete anche con tecnologie diverse lavorando a livello 2, inoltre gestisce l'accesso al canale riducendo le collisioni.
- **Switch**: analogo al bridge ma multiporta, possiede tabelle in cui associa ad ogni indirizzo MAC una porta di uscita, così da evitare la trasmissione di segnali verso segmenti di rete inutilmente riducendo il rischio di collisioni.

La nuova struttura di Internet2 e dei Servizi Differenziati permette di garantire i requisiti di qualità del servizio di comunicazione che Internet non può supportare (come il ritardo minimo garantito). Tutto si basa essenzialmente su nuovi router, che sono in grado di spedire prima i pacchetti urgenti e poi tutti gli altri, se avanza tempo.

CAPITOLO I

Struttura di internet

Internet è una rete di reti. Gli end-system (o host) sono i dispositivi che si connettono e sulla quale girano le applicazioni. È strutturata secondo un sistema gerarchico che vede gli ISP regionali che offrono servizi agli end-system e i global ISP che offrono servizi agli ISP regionali. La rete di accesso, come una rete domestica o una rete universitaria è chiamata così perché è la rete che ci dà accesso ad internet, ovvero alla quale ci colleghiamo. I link di comunicazione sono i mezzi fisici che interconnettono tutti i dispositivi e possono essere di varia natura. L'unità di informazione base che viaggia su internet è il pacchetto, e i

dispositivi che si occupano dell' inoltro dei pacchetti su internet sono i commutatori di pacchetto, router o switch. I protocolli sono le regole da seguire sia di carattere sintattico che di carattere semantico. I protocolli si trovano nelle **RFC** (request for comment) che vengono creati dalla IETF (Internet Engineering Task Force), che sono scritti in testo ASCII e che originariamente erano necessari per chiedere appunto commenti circa la creazione di un nuovo protocollo. Il ruolo principale di internet e' quello di scambiare bit e permettere quindi alle applicazioni di comunicare tra loro.

Network edge

Internet e' basato sul paradigma Client-Server. Sia client che server sono end-system anche chiamati host. Spesso i regional ISP sono considerati edge. Quando ci connettiamo al router di una rete locale ancora non siamo connessi ad internet. Se poi il router di questa rete locale e' connesso tramite un regional ISP allora abbiamo accesso ad internet. Se inoltre il nostro regional ISP e' connesso ad un router di Tier-1 allora siamo nella core network. Possiamo connetterci ad internet tramite reti residenziali, istituzionali (come ad esempio l'universita') o con servizi wireless per cellulare. Con il termine bandwidth indichiamo la larghezza di banda nominale di una rete. Tuttavia tale larghezza di banda puo' non essere dedicata ad ogni singolo utente, ma al contrario potrebbe essere condivisa e di conseguenza l'effettiva velocità di trasmissione potrebbe essere inferiore. Questa e' una problematica delle reti edge, mentre le reti core sono tarate per permettere una comunicazione sempre efficiente e veloce.

Una rete di accesso residenziale e' generalmente costituita da un modem DSL, che spesso possiede anche funzionalità di access point wireless e di router/switch e che e' connesso ad una presa telefonica. Il modem permette di trasmettere i dati in arrivo lungo la rete telefonica, producendo variazione di corrente a determinate frequenze. Ad esempio la voce viaggia su frequenze da 0 a 4 KHz, da 4 a 50 KHz i segnali di upload e da 50 KHz in su i segnali in download. In questo modo e' possibile distinguere i segnali delle chiamate telefoniche da quelli destinati ad internet, pur avendo un unico mezzo trasmissivo. I dati in arrivo alla centrale operativa dell'ISP finiscono in un DSLAM che instrada i pacchetti verso la rete internet, mentre le chiamate vanno nella rete telefonica. Queste reti sono tipicamente asimmetriche e permettono di scaricare più velocemente di quanto permettano di uploadare su internet.

La rete infrastrutturale di un'azienda e' più complessa. Il router istituzionale analizza i dati in arrivo. Infatti bisogna sempre distinguere gli host degli impiegati e gli host nella DMZ (zona demilitarizzata). In questa zona vi si collocano i server aziendali che devono poter essere raggiunti dall'esterno, mentre gli host privati aziendali non devono essere raggiungibili dall'esterno, di conseguenza se il router vede arrivarsi dall'esterno pacchetti destinati ad host privati aziendali, li bloccherà per ragioni di sicurezza.

Uno dei parametri che ci permettono di misurare la bontà di una rete e' il ritardo di trasmissione. Il ritardo di trasmissione di un pacchetto e' il rapporto tra gli L bit del pacchetto e la velocità del link in cui viaggia R bit/s. I link di trasmissione possono essere guidati, se l'informazione viaggia vincolata nel mezzo fisico come ad esempio nel filo in rame, mentre quelli non guidati non sono vincolati e viaggiano nell'ambiente, come ad esempio le onde radio. Il vantaggio dei mezzi non guidati rispetto a quelli guidati e' che permettono una maggiore mobilità, tuttavia i segnali in questo modo sopravvivono meno e subiscono più interferenza, di conseguenza necessitano di essere trasmessi a maggior intensità'

consumando più' energia e, inoltre, viaggiando nell'ambiente arrivano dappertutto, potenzialmente anche a malintenzionati. I cavi ethernet sono costituiti da fili di rame intrecciati schermati per ridurre l'interferenza esterna. I cavi categoria 5 permettono di viaggiare a 100 Mbps, mentre i categoria 6 sono schermati meglio e raggiungono anche 10 Gbps. I cavi coassiali sono due conduttori di rame concentrici isolati tramite un isolante in mezzo, e la differenza di potenziale che si crea tra i due conduttori di rame genera sinusoidi che possono essere differenziate in base alla frequenza. Nella fibra ottica viaggiano onde luminose che permettono di trasmettere dati ad alta velocità'.

Network core

L'architettura core e' una mesh di router interconnessi. I router della rete core prendono i pacchetti che arrivano dalle reti di accesso e li inoltrano. I link che connettono i router core sono fibre ottiche ad alte prestazioni. Quando abbiamo un file da trasmettere lo dividiamo in pacchetti di dimensione L. I bit viaggiano e vengono accumulati nei router intermedi finché l'intero pacchetto non e' stato ricevuto (Store-and-Forward). Quindi per spedire un pacchetto il ritardo end to end e' $\frac{N*L}{R}$ dove N e' il numero di collegamenti. Generalmente i pacchetti vengono accumulati in una coda. Vi e' un buffer per ogni porta di ingresso ed ogni porta di uscita. Quando il pacchetto viene messo in una coda di uscita deve aspettare finché' il canale non e' libero. I pacchetti possono andare perduti se le code sono piene. Ogni router deve incarnare due aspetti, i routing protocol, ovvero come scrivere nelle tabelle di instradamento quale linea di uscita far prendere ad ogni pacchetto in arrivo, ed il forwarding, ovvero analizzare l'intestazione di ogni pacchetto in ingresso e destinarlo a una linea di uscita basandosi sulle tabelle di inoltro. I routing protocol devono essere coerenti e basati su regole comune tra i router. Un sistema autonomo e' un'isola di router che e' gestita tramite le medesime regole.

E' ipotizzabile implementare **circuit switching** su una rete internet. Per fare ciò' bisogna allocare risorse end-to-end tra chi comunica. Quando si allocano risorse non vi e' bisogno di piu' link di comunicazione perché' e' possibile dividere lo stesso link in piu' canali virtuali tramite suddivisione di frequenze (FDM) e suddivisione di tempo (TDM). Nel caso di TDM il tempo e' suddiviso in frame e ad ogni comunicazione viene riservato uno slot all'interno del frame. Il problema del circuit switching e' che bisogna per tutta la durata della comunicazione allocare risorse end-to-end, anche durante i momenti in cui non avvengono scambi di dati. Per una questione di probabilità' e' difficile che molte persone siano connesse contemporaneamente, per questo e' preferibile utilizzare packet switching che per la stessa quantità' di risorse disponibili permette di accomodare più' utenti assicurando le stesse prestazioni. Inoltre nel packet switching non c'e' il call setup, ovvero la creazione del circuito tra mittente e destinatario.

Ad esempio se abbiamo utenti attivi il 10%, che usano 100kbps su un link di 1 mbps ne possiamo accomodare al massimo 10 con circuit switching. Usano la distribuzione binomiale secondo la formula $\binom{n}{k} (p)^k (q)^{n-k}$ si ha lo 0.0004 di probabilità' che con 35 persone accomodate più di 10 stiano trasmettendo.

Il problema del packet switching e' il congestionamento, i ritardi di accodamento e il possibile packet loss.

Per consentire una comunicazione globale bisogna interconnettere le reti di accesso tra di loro. Una soluzione ingenua e' connettere ogni access ISP ad ogni access ISP, ma in questo modo la rete non scala in quanto avremmo bisogno di N^2 connessioni. Per questo nella

pratica si crea un'architettura gerarchica creando una rete di global ISP che connettono tra di loro i vari ISP regionali. Vi sono diversi gruppi di ISP globali che sono collegati tra di loro mediante **peering link** e **IXP**, ovvero punti di scambio ad altissime prestazioni. Gli ISP clienti si connettono ai fornitori mediante i Point of Presence (**POP**), ovvero un gruppo di router tramite i quali gli ISP clienti si connettono al fornitore. Un meccanismo parallelo e' costituito dalle reti dei fornitori di contenuti, in cui un'azienda come Google crea la sua rete che connette solamente i suoi server e che puo' connettersi agli ISP regionali senza dover passare per un ISP di tier-1. Gli elementi che producono ritardi dovuti ad un router sono:

- Nodal processing: dovuto al controllo di errori e scelta del link di uscita;
- Ritardo di accodamento: dovuto al tempo che un pacchetto deve aspettare in coda prima di poter essere trasmesso;
- Ritardo di trasmissione: dipende dalla larghezza di banda e dalla dimensione del pacchetto;
- Ritardo di propagazione: dipende dalla distanza fisica tra il router e il nodo successivo.

Il fattore **La/R**, in cui L sono i bit si ogni pacchetto, a i pacchetti in arrivo ogni secondo ed R la velocità di trasmissione, chiamato **intensità di traffico** rappresenta il rapporto di bit in arrivo e bit in uscita. Se questo rapporto e' minore di 1 vuol dire che quello che arriva e' minore di quello che parte, quindi siamo ancora al di sotto del livello di congestione. Tuttavia quando questo rapporto tende ad uno il ritardo di accodamento inizia a tendere ad infinito. Per misurare questi fattori si usa il programma **Traceroute**. Traceroute e' un programma applicativo che sfrutta il protocollo di livello rete **ICMP**. ICMP scambia messaggi a livello rete utilizzando il protocollo IP, e riguarda sia router che host. I messaggi scambiati hanno come contenuto informazioni sulla rete, ad esempio:

- Rete di destinazione non raggiungibile (possibile interruzione di rete?)
- Rete di destinazione sconosciuta (indirizzo di rete male specificato?)
- Host destinazione non raggiungibile (host spento o scollegato?)
- Host destinazione sconosciuto (indirizzo di host male specificato?)
- Protocollo richiesto non disponibile (servizi non previsti)
- Ricerca di un cammino alternativo per la destinazione (se esiste)

Il programma traceroute, basato su ICMP, manda una serie di pacchetti per la rete verso una destinazione. Ognuno di questi pacchetti ha un **TTL** (time to live) che indica il numero massimo di hop che quel pacchetto puo' compiere. Il primo pacchetto viene mandato con un TTL pari ad 1, quindi il primo router lo decrementera' a 0 ed invierà il suo pacchetto eco di risposta che contiene il suo indirizzo IP e il RTT, ovvero il tempo di andata e ritorno.

Dopodiche' verra' mandato un secondo pacchetto con TTL = 2 e cosi via ad aumentare finché si spera di giungere alla destinazione. Router successivi possono avere RTT minore perche' nel mentre le condizioni della rete possono cambiare, o perche' puo' essere cambiato anche il percorso. Quando un pacchetto ICMP non torna entro un tempo limite (generalmente 3 secondi, ma è modificabile dall'utente), il comando traceroute sostituisce il tempo mancante con un asterisco. Generalmente vengono inviati 3 pacchetti di fila non uno solo. Un'altra applicazione e' **PING**. Ping serve per verificare l'esistenza di una connessione tra due host. Viene mandato un pacchetto di tipo echo request e si fa partire un timer.

Quando arriva al destinatario questo manda un messaggio echo reply al mittente che serve per calcolare il RTT, e in caso di insuccesso viene indicato che il timer per la richiesta inviata è scaduto senza ottenere risposta. Al termine dei tentativi, viene mostrato un elenco di statistiche sul numero di richieste andate a buon fine e i tempi medi stimati di andata e ritorno dei pacchetti

Il **throughput** e' la capacita' di consegna dei dati in bit al secondo da mittente a destinatario. Il throughput istantaneo e' la velocità' con cui si stanno ricevendo i dati, il throughput medio e' la velocità' di ricezione dei dati in un certo lasso di tempo. Il throughput end-to-end dipende dal link piu' lento nel percorso mittente destinatario, e questo link e' detto bottleneck link. Generalmente il collo di bottiglia si ha nei link della rete edge. Si definisce invece **goodput** la quantità di dati utili nell'unità di tempo del throughput trasmesso scartando la extrainformazione o informazione di overhead associata ai protocolli durante la trasmissione.

Reti sotto attacco

Un malware e' un codice malevolo che può' immettersi in un dispositivo mediante un **virus**, ad esempio eseguendo un programma che include codice malevolo o aprendo una mail dannosa. I virus sono malware che richiedono una qualche forma di interazione con l'utente per infettare il dispositivo. I **worm** sono malware che possono entrare in un dispositivo senza alcuna interazione esplicita con l'utente, ad esempio utilizzando un programma vulnerabile. I sistemi infettati possono essere controllati da utenti malintenzionati ed entrare in una **botnet**. Una botnet può' essere usata per effettuare attacchi Dos ad aziende, bombardando i server di richieste contemporaneamente e impedendo alle richieste legittime di andare a buon fine. Un'altra tecnica malevola e' il **packet sniffing**, che consente ad utenti malintenzionati di leggere il traffico in circolo in una determinata rete e ricevere messaggi o informazioni riservate non destinate a loro. Infine e' possibile mascherare il proprio IP (**IP spoofing**) per fingersi qualcun altro all'interno della rete per compiere azioni illecite.

CAPITOLO II

Principi delle applicazioni di rete

I programmi applicativi di rete devono essere fatti per essere eseguiti su diversi dispositivi attraverso Internet. L'architettura client-server si basa sul concetto di client e di server. Un server e' un host sempre connesso, sempre raggiungibile, che non muta il suo indirizzo IP e che offre determinati servizi. Un client e' un fruitore di servizi, puo' mutare il suo indirizzo IP e può connettersi solo quando gli e' necessario. I client non comunicano direttamente tra di loro. Nell'architettura P2P una macchina può svolgere sia il ruolo di client che di server. Quando un peer ha bisogno di un file effettua una richiesta e il primo peer che può risolvere tale richiesta fungerà da server per inviare quel file. Quindi i peer comunicano direttamente tra loro, e ogni richiedente diventa anche un potenziale fornitore, di conseguenza l'architettura P2P e' fortemente scalabile. Il problema di questo paradigma e' che non essendoci un dispositivo fisso a cui fare richiesta, tutti i peer possono cambiare il loro indirizzo IP e la gestione della rete diventa complicata quando bisogna determinare a chi richiedere un determinato file.

Le applicazioni comunicano tramite **processi** che si scambiano messaggi. Se due processi appartengono allo stesso host, allora questi possono comunicare mediante l'inter-process communication definito dal sistema operativo o potrebbero utilizzare lo stack ISO/OSI per

inviare e ricevere il messaggio sulla stessa macchina. Se la comunicazione avviene tra host diversi si deve stabilire una connessione tra **socket**. I client sono gli host che iniziano i processi di comunicazione mentre i server attendono che gli arrivino richieste. Sia chi trasmette che chi riceve utilizza un socket in modo analogo ad una porta: chi trasmette spinge i dati verso il socket mentre chi riceve preleva i dati dal socket. Nell'architettura ISO/OSI i primi quattro livelli vengono gestiti dal sistema operativo, mentre l'utente può creare le applicazioni utilizzando le API del livello applicazione. I processi per scambiare messaggi devono avere identificatori. Un identificatore deve includere sia l'indirizzo IP della macchina sia il numero di porta su cui quel processo è in esecuzione. I protocolli del livello applicazione devono definire il tipo dei messaggi scambiati, la sintassi di questi messaggi e la loro semantica e le regole comportamentali per decidere quando e come inviare o rispondere a questi messaggi. I protocolli possono essere open, nel senso che chiunque può vedere la definizione del protocollo, di conseguenza studiarlo e sfruttarlo al meglio, o proprietari per questioni economiche. I servizi che le app necessitano a livello trasporto sono:

- Data Integrity, certe applicazioni necessitano che i dati siano trasmessi con successo e senza errori. Alcune app sono tolleranti ad un certo margine di errore.
- Timing, alcune applicazioni necessitano bassi ritardi e bassa variabilità del ritardo per tarare nel modo giusto la connessione.
- Throughput: alcune app possono richiedere che un minimo throughput sia garantito per consentire una comunicazione efficiente, altre app 'elastiche' possono funzionare anche se in alcuni momenti il throughput subisce dei cali significativi.
- Security: alcune app necessitano che la comunicazione sia sicura.

I protocolli del livello trasporto che offrono servizi alle applicazioni sono TCP e UDP. TCP gestisce la comunicazione in modo che tutti i dati siano recepiti senza errori, regola il congestionamento della rete e non spedisce i dati più velocemente di quanto il destinatario possa riceverli. Non dà assicurazioni circa il timing, il throughput e la sicurezza. La connessione TCP è connection oriented, ovvero viene stabilita una connessione prima che avvenga la trasmissione. UDP non fornisce alcun servizio, è un protocollo più leggero di TCP e viene utilizzato da quelle app che non hanno bisogno di molte garanzie per quanto riguarda la comunicazione. In poche parole serve per bypassare il livello trasporto. Per rendere TCP sicuro bisogna aggiungere il protocollo SSL, che agendo a livello applicazione serve a rendere cifrate le connessioni TCP, senza agire sulle connessioni in sé'.

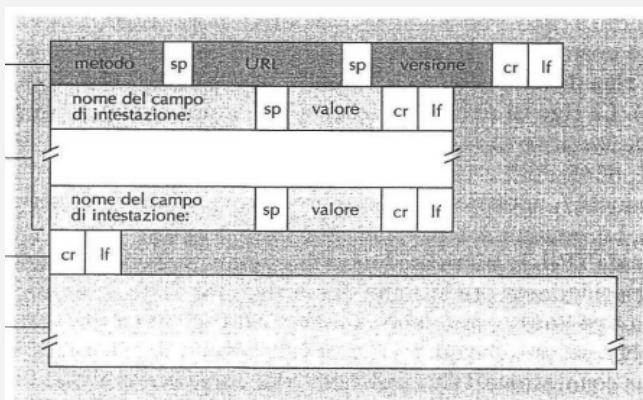
Web e HTTP port 80

L'obiettivo del servizio Web è condividere documenti. I documenti web sono scritti nel linguaggio **HTML**. Quando un browser scarica una pagina Web, esso mostra tutto il suo contenuto testuale ed il suo formato. Eventuali altri oggetti che sono inclusi (linkati) nel documento HTML vengono rilevati dal browser quando esso ne fa il parsing del documento, e successivamente verranno scaricati e disposti nel riquadro apposito secondo quanto definito dal contenuto del documento HTML. Ogni oggetto possiede un indirizzo univoco chiamato **URL**. Un URL è composto dal hostname e dal percorso in cui questo oggetto si trova all'interno del sistema. HTTP è il protocollo che definisce lo standard di come effettuare le richieste e inviare le risposte nel web. HTTP si basa su TCP. Il client inizializza una connessione TCP sulla porta 80 verso il server. Se il server accetta la connessione può iniziare lo scambio dei messaggi. I messaggi scambiati sono messaggi HTTP che il client

invia per mezzo di un browser ad un Web Server. Quando il client ha ricevuto la pagina web la connessione TCP, nella versione HTTP 1.1 cade, e se nel documento vi era un link ad un altro oggetto bisogna ristabilire un'altra connessione TCP. Questo viene fatto per evitare di lasciare allocate inutilmente risorse da parte del Web Server. Tuttavia vista l'evoluzione delle pagine web, chiudere ogni volta la connessione può essere sconveniente, per questo nella release 1.2 del protocollo HTTP e' il client a decidere quando buttare giù la connessione. HTTP e' senza stato, quindi i server non mantengono informazioni delle richieste effettuate dai client. Definiamo Round Trip Time (RTT) il tempo impiegato da un piccolo pacchetto per andare da mittente a destinatario e viceversa. Nel caso di connessione persistente, ogni volta bisogna aprire la connessione (1 RTT) e poi richiedere il file (1 RTT + file transfer). Quindi per ogni oggetto il tempo e' di 2 RTT + file transfer. Nelle connessioni persistenti, la connessione TCP viene lasciata aperta quindi per ogni oggetto il tempo e' 1 RTT + file transfer. I messaggi HTTP possono essere di richiesta o di risposta. Un messaggio HTTP di richiesta include sempre una **riga di richiesta**, che e' suddivisa in un campo metodo, un campo pathname e un campo versione HTTP. Poi si trovano le header lines. Ogni riga e' nel formato campo: valore. alla fine di ogni riga si aggiungono i simboli \c\r (a capo e nuova linea). Per terminare si aggiunge una riga vuota con solo \c\r, poi c'e' il corpo del messaggio. I metodi possono essere GET, POST, HEAD, PUT, DELETE:

- GET serve per richiedere pagine web
- POST per caricare sul server informazioni come ad esempio quando si digita sul form di ricerca di un motore di ricerca. Le informazioni che scriviamo nel form sono immesse nel campo body del messaggio HTTP. Tuttavia questo e' possibile farlo anche con il metodo GET estendendo l'URL con le informazioni che immettiamo.
- Il metodo HEAD e' simile a GET ma non preleva gli oggetti richiesti.

Questi tre metodi sono inclusi in HTTP\1.0, HTTP\1.1 include anche i metodi PUT e DELETE. Non tutti i campi delle header lines sono obbligatori, tranne il campo Host: che indica a chi fare richiesta. Altri campi possono riguardare il formato in cui ricevere i messaggi, il browser utilizzato e se lasciare aperta la connessione o no.



Una tipica risposta HTTP inizia con una linea di stato che inizia con la versione di HTTP usata, un numero che indica lo stato della richiesta. Le header lines includono informazioni circa il documento richiesto come la data dell'ultima modifica, importante perché se il client ha in cache una copia del file con data successiva all'ultima modifica, allora non e' necessario il

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(data data data data ...)
```

trasferimento dello stesso file. Poi il server che ha soddisfatto la richiesta ed altre informazioni sul formato dei dati e sulla connessione. I codici di stato della prima linea possono essere:

- 200 OK, ovvero richiesta soddisfatta;
- 301 Moved Permanently, ovvero il file non si trova più in quel server e la nuova locazione viene indicata nel campo Location;
- 400 Bad Request, il server non ha compreso il messaggio;
- 404 Not Found, il documento richiesto non è presente su quel server;
- 505 HTTP Version Not Supported.

Diversi Web Server utilizzano i **Cookies** per tenere traccia di chi siano i client che fanno richiesta e quale stato della comunicazione è stato raggiunto. Non sono altro che una riga di header nel messaggio HTTP. Un cookie è un valore intero numerico. La prima volta che un client contatta una piattaforma, il server crea un ID del client con un numero e lo inserisce nel suo database. L'ID viene inserito anche nella risposta HTTP nel campo set-cookie. A questo punto il browser associa quel server a quel cookie all'interno di una cache, e quando contatta nuovamente quel server va settato il valore cookie. I cookie servono per mantenere lo stato anche se la connessione TCP cade nel frattempo o cambiamo indirizzo IP.

Il **proxy server** è una macchina che si frappone tra chi fa le richieste e il server a cui vengono effettuate. Se una pagina richiesta si trova all'interno della cache del proxy server questo restituisce la pagina senza contattare direttamente il server di origine, in caso contrario il proxy web richiede la pagina al server e se la salva in cache e poi la inoltra al client. I proxy server vengono installati all'interno di reti locali dove generalmente la velocità di connessione è molto elevata. Il web caching è utilizzato dagli ISP che conoscono le richieste che gli utenti effettuano su internet e di conseguenza possono salvare le pagine che richiediamo per ridurre il traffico di dati su internet e aumentare la responsività della comunicazione. Se un proxy server o il nostro browser hanno nella cache una versione obsoleta della pagina web da noi richiesta, vorremmo trovare il modo di ricevere la versione aggiornata. Si aggiunge nelle header lines del metodo GET il campo If-modified-since<date>. Se il file è stato modificato dopo la data specificata allora il server produce un messaggio 304 Not Modified, altrimenti arriva la pagina aggiornata che va sovrascritta a quella in cache.

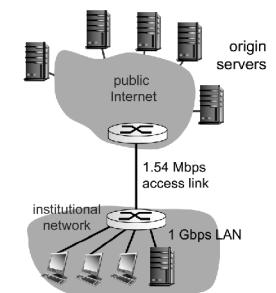
Caching example:

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 15% *problem!*
- access link utilization = 99%
- total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + usecs



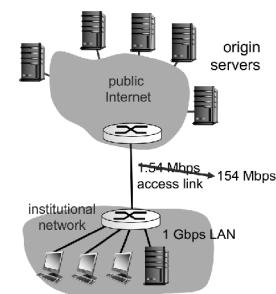
Caching example: fatter access link

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers: 15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: 1.54 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = 99% → 9.9%
- total delay = Internet delay + access delay + LAN delay
= 2 sec + minutes + usecs



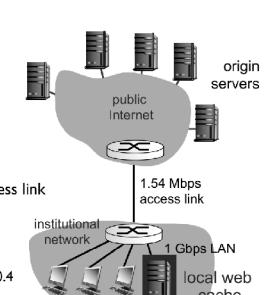
Cost: increased access link speed (not cheap!)

Application Layer 2-40

Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
 - 40% requests satisfied at cache, 60% requests satisfied at origin
- access link utilization:
 - 60% of requests use access link
- data rate to browsers over access link
= 0.6 * 1.50 Mbps = .9 Mbps
 - utilization = 0.9 / 1.54 = .58
- total delay
 - = 0.6 * (delay from origin servers) + 0.4 * (delay when satisfied at cache)
 - = 0.6 (2.01) + 0.4 (~msecs) = ~ 1.2 secs
 - less than with 154 Mbps link (and cheaper too!)



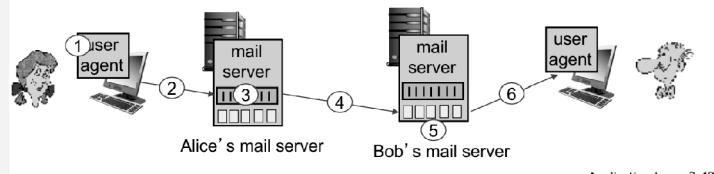
Application Layer 2-42

E-mail port 25

La posta elettronica si basa sul protocollo SMTP. Un **User Agent** è lo strumento utilizzato per comporre o leggere le e-mail. I messaggi vengono inviati tramite il protocollo SMTP dal lato client di un server mail al mail server di destinazione. Chi ha un account e-mail ha una quota di disco in un server SMTP del dominio di posta elettronica (Mailbox). I messaggi che non possono essere inoltrati al mail server di destinazione vengono immagazzinati in una coda. SMTP usa una connessione TCP persistente. I metodi di cui SMTP fa uso sono HELO, che indica l'host da cui proviene la mail, MAIL FROM che indica l'indirizzo mail del mittente, RCPT TO che indica a chi inviare la mail, DATA che serve per indicare il messaggio che si vuole inviare ed infine QUIT per chiudere la connessione TCP. La differenza tra HTTP e SMTP è che il primo è un protocollo PULL perché tende a tirare a sé gli oggetti che vengono prelevati dai server, mentre SMTP è un protocollo PUSH in quanto si inviano i messaggi verso i mail server. Lo standard per i messaggi di posta elettronica contenuto nel RFC 822 include 3 righe di header che sono From, To e Subject e che sono differenti da quelli di SMTP e in seguito il body del messaggio separato da una riga vuota. I protocolli usati per prelevare la posta dal proprio mail server sono **POP3**, **IMAP** e **HTTP**. Nel protocollo POP3 (porta 110) vi è una fase di autenticazione tra client e server in cui vengono inviati username e password. Se le autenticazioni vanno a buon fine il server risponde con OK altrimenti ERR. Poi è possibile scaricare i propri messaggi con i comandi list che genera una lista dei nuovi messaggi, retr che ritira un messaggio dele che lo elimina e quit per terminare la sessione. Il POP3 ha due limitazioni, l'autenticazione avviene in chiaro e quando ritiriamo un messaggio nella modalità scarica e cancella, la copia sul server viene eliminata, quindi non si può scaricare la posta da un altro client. IMAP presenta numerosi vantaggi rispetto a POP3 come la possibilità di accedere da remoto alle proprie mail che si trovano tutte sul server, la possibilità di mettere le mail in diverse cartelle e il fatto che IMAP mantenga lo stato tra una sessione e l'altra, come ad esempio i nomi delle cartelle e l'associazione tra messaggi e cartelle. Infine si può usare un browser come client mail e quindi sfruttare il protocollo HTTP per inviare o ricevere messaggi. Tuttavia i server mail che comunicano utilizzano ancora SMTP per recapitare i messaggi.

Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message "to" bob@someschool.edu
- 2) Alice's UA sends message to her mail server; message placed in message queue
- 3) client side of SMTP opens TCP connection with Bob's mail server
- 4) SMTP client sends Alice's message over the TCP connection
- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his user agent to read message



Application Layer 2-48

DNS port 53

Il DNS è implementato tramite una gerarchia di Nameservers che possono essere contattati tramite un protocollo per risolvere un nome di dominio in un indirizzo IP. Il servizio DNS è considerato una funzionalità stessa di internet e i server vengono distribuiti nella parte edge

di internet, perché applicare una politica distribuita permette di gestire a livello locale il lavoro che i server DNS devono svolgere. Altre funzionalità del servizio DNS sono la possibilità di fare aliasing, ovvero ad uno stesso indirizzo IP possono essere associati più nomi e anche la possibilità di associare più indirizzi IP ad uno stesso nome, in modo da distribuire le richieste che vengono effettuate ad un unico web server (ad esempio www.google.com). Il motivo per cui i DNS non vengono gestiti in un unico server DNS è che il sistema non scalerebbe. Il meccanismo con cui funziona DNS è ad albero: la richiesta parte dal basso e se il server DNS non sa risolvere tale richiesta la manda verso l'alto fino ad arrivare ai Root DNS server che fa ridiscendere le richieste nel lato giusto dell'albero. Nel livello più basso dell'albero ci sono gli authoritative DNS come amazon o yahoo. Al livello più alto ci sono i domini come .com, .org o .it (Top level domain) ed infine la radice. A volte ci possono essere DNS ancora più locali rispetto agli authoritative DNS. Ogni ISP possiede il suo DNS server e quando si effettuano richieste DNS si contatta prima questo.

Generalmente questo server ha una cache in cui contiene tutti i mapping nome-indirizzo IP di richieste precedentemente effettuate. Se non conosce l'associazione dovrà propagare la richiesta verso l'alto. La richiesta DNS può essere di tipo **iterativo**, in questo caso si contatta il proprio DNS locale. Se esso sa la risposta la darà al client, altrimenti contatterà il server DNS di livello superiore che gli dirà il DNS da contattare per ottenere la risposta.

Quindi ad ogni iterazione il local DNS otterrà il nome del server successivo da contattare.

Tutto il lavoro in questo caso viene fatto dal local server DNS. Un'alternativa è implementare la richiesta DNS in modo **ricorsivo**. Ad ogni passo il server interpellato diventa il client che fa la richiesta al server successivo in maniera ricorsiva. In questo modo il local DNS avrà meno carico di lavoro a discapito del root DNS. Il protocollo DNS implementa il protocollo UDP, tuttavia se non vede ritornare la risposta entro un certo limite la rimanda lui stesso. Una cache non può essere valida indefinitamente perché le associazioni potrebbe cambiare. Tutte le entry a livello cache hanno un time to live. Inoltre possono essere invalidate le tabelle di cache se bisogna aggiornare un'informazione. Un DNS record chiamato Resource Records (RR) è nel formato (name, value, type, ttl). ttl rappresenta tempo di vita del record. A seconda del valore di type gli altri campi assumono diversi significati:

- type=A, name è l'hostname e value è l'indirizzo IP;
- type=NS, name è il dominio, value è l'hostname dell'autoritative name server per questo dominio. Per esempio, (foo . com , dns . foo . com , NS) è un record di tipo NS;
- type=CNAME, è l'alias per qualche nome canonico, value è il nome canonico.
- type=MX, value è il mail server associato al dominio scritto in name.

Sia richieste che risposte DNS hanno lo stesso formato. I primi 16 bit sono dedicati all'identificazione in cui viene indicato il numero della domanda che effettuiamo, così da poter fare il mapping tra domanda e risposta. Altri 16 bit vengono dedicati ai flags in cui si specifica che si

identification	flags
# questions	# answer RRs
# authority RRs	# additional RRs
questions (variable # of questions)	
answers (variable # of RRs)	
authority (variable # of RRs)	
additional info (variable # of RRs)	

tratta di una query o di una risposta, se la query e' ricorsiva o meno, se la ricorsione e' disponibile o se la risposta arriva da un DNS authoritative. Nelle quattro righe successive troviamo i numeri che indicano le occorrenze delle quattro righe che seguono l'intestazione. Dopo l'intestazione troviamo nella prima riga la query che include il nome che si sta cercando e il tipo di richiesta (A, MX, ecc). Il campo risposta include gli RR in risposta alla query, il campo authoritative contiene i record di altri server authoritative, infine il campo addizionale che include informazioni aggiuntive. Per registrare un nuovo dominio bisogna fornire ad un DNS registrar il nome e gli indirizzi IP degli authoritative name server primario e secondario. Il registrar inserisce due RRs nel TLD server, un record type NS e un RR type A per associare il dominio all'autoritative DNS e quest'ultimo al suo indirizzo IP. A questo punto e' possibile creare nell'autoritative DNS server del nostro dominio tutti gli RR di tipo A per associare ogni nome ad un indirizzo IP, fare alias o mail server. Nell'ambito DNS un attacco DDos e' il più pericoloso. Attaccare un root server non e' molto efficace perché applicano filtraggio del traffico. Per questo si bombardano i TLD server. Altre tecniche sono intercettare le query e fare DNS poisoning, ovvero dare risposte sbagliate e compromettere la cache.

P2P

Vogliamo supportare un servizio attraverso la forma partecipativa di chi e' connesso in un determinato momento. Se un server deve mandare un file ad N client sulla rete, dovrà inviare N copie di quel file. Il tempo per distribuire le N copie e' il massimo tra il tempo che impiega il server per trasferire gli N file ed il tempo che impiega il client più lento a scaricare la sua copia. Il primo valore incrementa linearmente in N. Nel caso di architettura P2P, un server deve immettere almeno una copia del file, dopodichè potenzialmente non e' più necessario. Appena un client ha la copia potrebbe diventare server. Tutti i client nel complesso devono scaricare NF bits, dove F e' la dimensione di una copia del file. Tuttavia il max upload rate e' dato non solo dal server ma anche dalla somma degli upload dei client che hanno acquisito una copia. Quindi con un approccio P2P il tempo per distribuire le N copie e' il massimo tra il tempo che il server impiega per inviare una copia, il tempo che il client più lento impiega a scaricarla e il tempo che

impiega la rete a trasmettere le N copie al massimo upload rate, che aumenta linearmente in N ma scala meglio visto che potenzialmente aumentano i server che possono fornire il file ad altri. Ad esempio l'architettura BitTorrent funziona che i file vengono divisi in chunks da 256Kb e i peer si scambiano questi chunks. Vi sono poi dei nodi speciali chiamati **trackers** che registrano i nodi attivi in rete che posseggono la copia di un determinato file. In questa

$$\begin{aligned} & \text{time to distribute } F \\ & \text{to } N \text{ clients using} \\ & \text{client-server approach} \end{aligned} \quad D_{c-s} \geq \max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

$$\begin{aligned} & \text{time to distribute } F \\ & \text{to } N \text{ clients using} \\ & \text{P2P approach} \end{aligned} \quad D_{P2P} \geq \max\{F/u_s, F/d_{min}, NF/(u_s + \sum u_i)\}$$

increases linearly in N ...
... but so does this, as each peer brings service capacity

architettura il collo di bottiglia e' la velocità di download del client. Quando un client ottiene un chunks potenzialmente lui puo' diventare fornitore registrandosi nella rete, ovvero dicendo ai trackers il proprio indirizzo e la lista dei chunks che possiede. Un problema di questa rete sono i nodi (**selfishly**) che richiedono servizi ma non forniscono servizi nella rete (scaricano ma non fanno upload). Per questo esistono meccanismi che permettono di inviare chunks solo se chi gli richiede ha fornito servizio all'interno della rete.

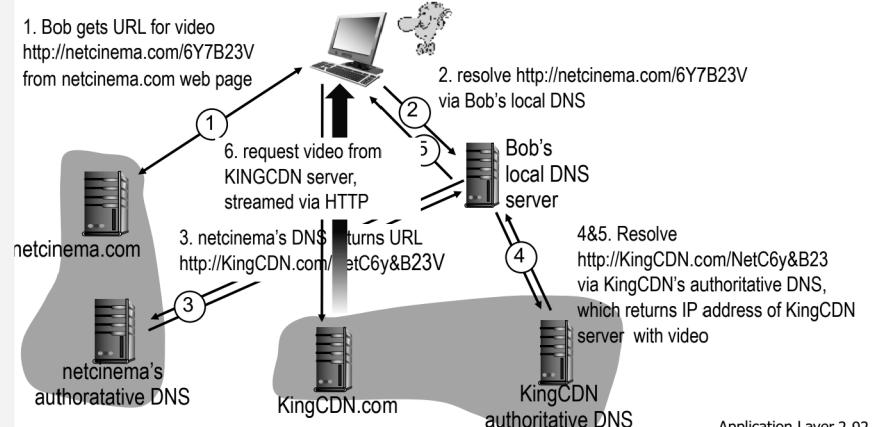
Video streaming and content distribution network

Dalla crescita a livello di internet dei contenuti video streaming si e' arrivati alla conclusione che la scalabilità dei sistemi riguardava sia i protocolli che l'architettura di rete. Vista l'eterogeneità dei client bisogna anche ottimizzare il modo in cui vengono codificati i file per impiegare meno rete possibile e fornire a tutti i contenuti che richiedono. La codifica delle informazioni delle immagini video deve eliminare la ridondanza dell'immagine e di quelle successive, ovvero eliminare l'informazione che può essere codificata implicitamente, così da diminuire il numero di bit da inviare. Tra un'immagine e la successiva si inviano solo i bit che sono cambiati. Si dice CBR (constant bit rate) se il video e' codificato sempre con lo stesso bit rate, mentre VBR (variable bit rate) se il bit rate cambia in base alle osservazioni precedenti. Una delle tecnologie streaming più utilizzate oggi e' il DASH (Dynamic, Adaptive Streaming over HTTP) che si basa sul protocollo HTTP. I server dividono i video in chunks. Ogni chunks e' memorizzato e codificato a diversi rate. Poi c'e' un file chiamato **Manifest File** che fornisce una URL diversa per ognuno dei diversi chunks, quindi per ogni frame abbiamo una URL per ogni formato. Il client periodicamente misura la quantità di bit che vengono effettivamente trasmessi dal server al client, così da analizzare il funzionamento della rete e intervenire di conseguenza, adattandosi ai chunks di qualità migliore possibile. Quindi il client in maniera intelligente determina quando richiedere i chunks, a quale encoding rate. Per realizzare questo servizio utilizzare un unico mega server e' impossibile perché sarebbe un unico punto di fallimento, i client potrebbero trovarsi troppo distanti dal server e questo sovraccaricherebbe lo sovraccaricherebbe di lavoro. In poche parole non scala. Un'alternativa e' generare copie multiple dei video distribuite geograficamente su determinati siti (**CDN**). Esistono due filosofie di posizionamento delle CDN, la prima detta Enter Deep consiste nel mettere un gran numero di CDN server nelle reti di accesso, così da avvicinarsi il più possibile agli utenti finali garantendo un alto

CDN content access: a closer look

Bob (client) requests video <http://netcinema.com/6Y7B23V>

- video stored in CDN at <http://KingCDN.com/NetC6y&B23V>



Application Layer 2-92

throughput e una bassa latenza. La seconda detta Bring Home consiste nel posizionare grandi cluster in meno punti e fuori le reti di accesso. Questi cluster vengono posizionati vicino i POP di Tier-1. In questo modo ci sono minori costi di manutenzione. Si sfrutta la authoritative DNS del fornitore di contenuti per mappare la richiesta verso il CDN da cui scaricheremo il nostro video, e l'authoritative DNS del CDN ci dirà da quale specifico server della CDN scaricarlo.

Socket programming

Nel caso di UDP non si stabilisce una connessione.

```
from socket import*
ServerName e' una variabile che indica l'indirizzo IP del server da contattare.
ServerPort e' la porta del server.
clientSocket = socket(AF_INET (significa che usa IPV4), SOCK_DGRAM (significa UDP))
message e' il messaggio da inviare
clientSocket.sendto(message.encode(), (serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048), crea un buffer di 2048
bytes dove mettere ciò che arriva dal server.
print(modifiedMessage.decode())
clientSocket.close()
```

```
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverPort
serverSocket.bind((" ", serverPort)), server per assegnare al socket questo server address e
la porta serverPort.
```

while True:

```
    message, clientAddress = serverSocket.recvfrom(2048)
    print(clientAddress)
    modifiedMessage = message.decode().upper()
    print(modifiedMessage)
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

Nel caso di TCP bisogna prima fare il three way handshake.

```
ServerName
ServerPort
clientSocket = socket(AF_INET, SOCK_STREAM (significa TCP))
clientSocket.connect((serverName, serverPort)), server per stabilire la connessione TCP
message = input()
clientSocket.sendto(message.encode(), (serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048),
print(modifiedMessage.decode())
clientSocket.close()
```

```

serverSocket = socket(AF_INET, SOCK_STREAM)
serverPort
serverSocket.bind(("", serverPort))
serverSocket.listen(1), serve per stare in ascolto sul welcoming socket 1 connessione alla volta.
while True:
    tcpConnectedClientSocket, addr = serverSocket.accept()
    print(addr)
    modified = tcpConnectedClientSocket.recv(2048).decode(), si usa recv al posto di
    recv from
    modifiedMessage = message.upper()
    tcpConnectedClientSocket.send(modifiedMessage.encode()), si usa send al posto di
    sendto
    tcpConnectedClientSocket.close()

```

CAPITOLO III

Il livello trasporto implementa un servizio di comunicazione logico end-to-end. Lato mittente si spezzano i messaggi delle app in segmenti che vengono passati alla rete sottostante. Lato ricevente bisogna riassemblare i segmenti per ricostruire i messaggi e inviarli alle app in esecuzione. Quando necessitiamo di un protocollo di livello trasporto affidabile, connection oriented e che gestisce il traffico e come i segmenti vengono inviati utilizziamo TCP. UDP non implementa nulla e' un'estensione del livello rete. I servizi non garantiti dal livello trasporto sono il minimo ritardo garantito e throughput garantito.

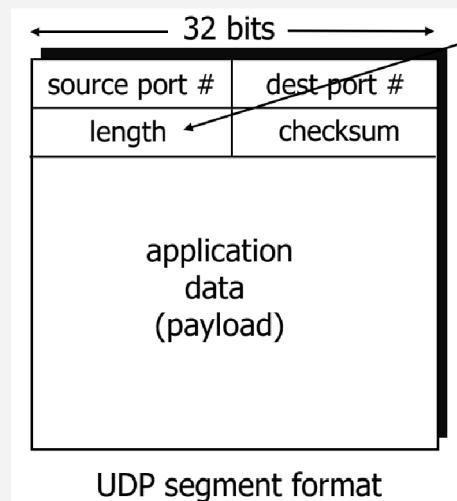
Multiplexing e Demultiplexing

Se abbiamo più applicazioni in esecuzione in un host, sia client che server, a livello trasporto possono esserci più flussi in arrivo/partenza che devono arrivare ognuno in una specifica applicazione. Il concetto di "imbuto" verso le applicazioni e' detto **multiplexing** e **demultiplexing**. Questa operazione viene gestita tramite i socket. Il demultiplexing usa le informazioni di header per indirizzare i segmenti verso il giusto socket, mentre il multiplexing prende i dati da più applicazioni, aggiunge le informazioni di header che saranno poi usate dal demultiplexing. Una volta che il pacchetto e' arrivato a destinazione il livello trasporto guarda la busta di livello trasporto. Le prime informazioni che trova sono **numero di porta sorgente** e **numero di porta destinazione**, **ognuna da 16 bit**. Il numero di porta destinazione indica la porta dove mandare i dati verso le applicazioni. Poi ci sono altre informazioni di header ed infine l'application data. Il lato client che effettua la richiesta non specifica la porta del socket perché viene battezzata nel momento in cui effettua la chiamata. Nel caso di UDP quando si riceve un segmento si definisce la porta di destinazione e si mandano i dati nel socket corrispondente a quella porta. Se arrivano dati da diversi IP ma verso la stessa porta UDP li manda alla stessa applicazione. Nel caso di TCP bisogna specificare attraverso il socket la sorgente e la destinazione e bisogna

mantenere lo stato di trasmissione tra i due estremi, quindi i segmenti in arrivo vanno mantenuti e gestiti. Un socket deve essere definito in modo univoco sia sul client che sul server in modo che si conosca sia la porta sorgente che destinazione. Questo permette a un server di controllare più connessioni TCP verso client diversi. Nel caso di UDP invece quello che arriva ad una porta viene mandato verso la porta da cui è stato ricevuto. Se più client fanno richiesta per una pagina web questi la fanno sulla porta 80, poi il server crea una porta alta dove realmente avviene la comunicazione (client socket). La porta 80 è detta **welcoming socket**. Questo viene fatto perché il server può effettuare più operazioni in parallelo (multithread).

UDP

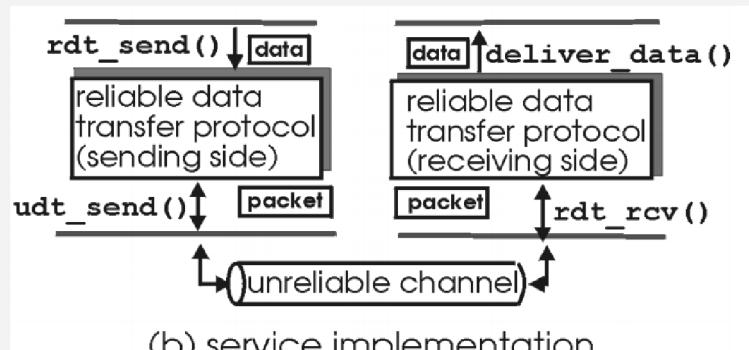
UDP è best effort, ovvero fa del suo meglio ma non offre assicurazioni circa il trasferimento dei dati. In alcuni casi può essere reso un poco più affidabile aggiungendo l'affidabilità a livello applicazione (reliable UDP). Un segmento UDP è costituito da 16 bit per la porta sorgente, 16 bit per la porta destinazione, 16 bit per dire la lunghezza del segmento in bytes che include l'header ed infine il checksum che ci dice se la cosa che inviamo è corretta o sbagliata e nel caso fosse sbagliato provvederà a cestinare il contenuto. Il meccanismo di **checksum** lato sender tratta il segmento intero in sequenze di 16 bit ed effettua la somma delle colonne e poi il complemento ad uno. Il ricevente effettua il checksum sul segmento ricevuto e se non coincide con quella calcolata dal sender vuol dire che c'è un bit errato.



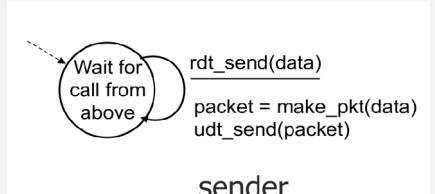
UDP segment format

Costruire TCP

A livello rete abbiamo costruito una rete inaffidabile, e vogliamo colmare costruendo un protocollo che renda il canale tra sender e receiver affidabile. Se un'applicazione richiede il trasferimento dei dati affidabile il livello trasporto deve offrire una primitiva che svolga questo lavoro (`rdt_send()`). Questa primitiva dovrà generare dei pacchetti che spedirà tramite una primitiva offertagli dal livello sottostante (`udt_send()`). Quando si riceve il pacchetto il livello trasporto provvederà a ricevere i dati dal livello sottostante e riordinarli con una primitiva apposita (`rdt_recv()`) e poi spedirli all'applicazione che li ha richiesti. Dobbiamo creare un algoritmo che implementi tutto ciò e per farlo avremo bisogno di una macchina a stati. Se supponiamo che sotto abbiamo

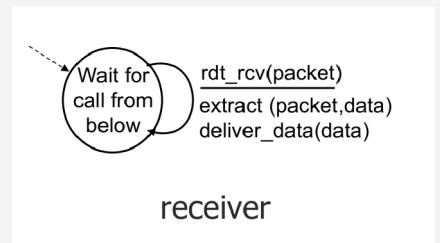


(b) service implementation



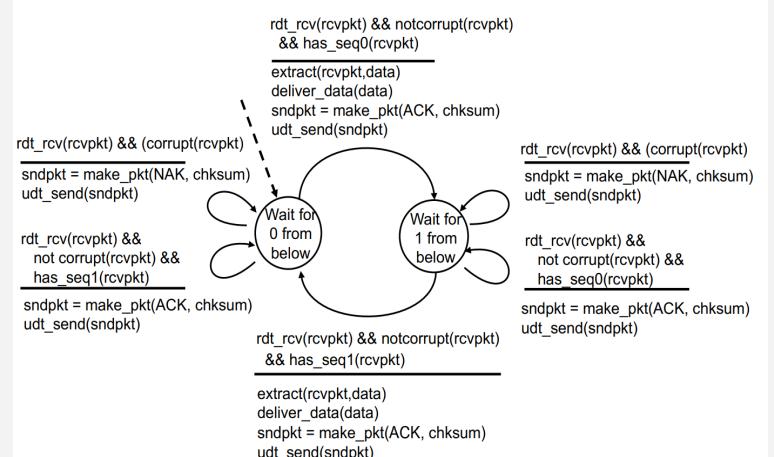
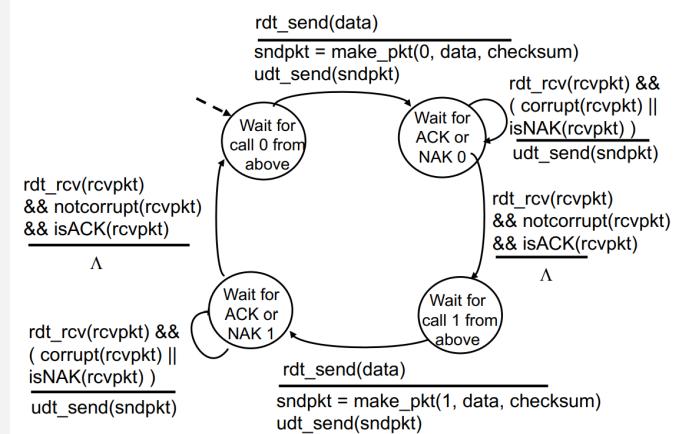
sender

già un canale affidabile, il protocollo non dovrà fare molto lavoro. Lato sender, si aspetta la chiamata da sopra, quando viene chiamata la `rdt_send(data)` il protocollo provvederà a dividere con una primitiva `make_pkt(data)` il messaggio in pacchetti e ad usare `udt_send(packet)` per spedire i pacchetti. Lato ricevente il protocollo è in attesa all'infinito. Quando viene chiamata la `rdt_rcv(packet)` viene estratto il campo `data` con `extract(packet, data)` e viene passato all'applicazione il messaggio tramite `deliver_data(data)`.



Ora per avvicinarci alla situazione reale, immaginiamo che nel canale rete ci possano essere errori sui bit. Il checksum rileva gli errori e vogliamo capire come risolvere gli errori. Se non si rilevano errori il receiver invia al sender un messaggio di acknowledgements (ACKs) per dire esplicitamente che il pacchetto è arrivato senza errori. In caso contrario si manda un negative acknowledgements (NAKs) se il pacchetto arriva con errori. Se il sender riceve un NAK manderà di nuovo il pacchetto. In questo caso il protocollo `rdt_send 2.0` resta sempre in attesa di chiamate dall'alto.

Quando crea il pacchetto aggiunge un campo checksum e lo invierà tramite `udt_send`. A questo punto però aspetta la ricezione del messaggio di conferma. In questo stato, se dal basso si riceve un pacchetto e questo pacchetto è un NAK richiama un'altra volta `udt_send` per rispedire il pacchetto arrivato con errori. Se riceve un pacchetto e questo è un ACK allora torna nello stato di attesa dall'alto. Lato receiver, se riceve dal basso un pacchetto e questo pacchetto è corrotto, allora invia un NAK usando `udt_send`, altrimenti se non è corrotto estraе dal pacchetto i dati, li manda all'applicazione e manda indietro un ACK. Il problema principale è se l'ACK o il NAK arrivano con errori al sender. Nel caso in cui il sender rispedisca il pacchetto anche nel caso in cui arriva un ACK/NAK sbagliato, potrebbero generarsi duplicati. Quindi il sender deve aggiungere nei numeri di sequenza ad ogni pacchetto così che il receiver scarti tutti i pacchetti duplicati. In questa nuova versione del protocollo `rdt` che chiamiamo `rdt_send 2.1`, inizialmente il protocollo è sempre in attesa di dati dall'alto. In questo caso però differenziamo le chiamate dall'alto in chiamata 0 e chiamata 1. Quindi si numera il primo pacchetto come pacchetto 0 e lo si spedisce come prima. Poi si aspetta la risposta dal receiver del pacchetto 0, se questa è corrotta o è un NAK allora rispedisce il pacchetto, altrimenti se non è corrotta ed è un ACK non si fa niente e si va nello stato "aspetta chiamata 1" e si fa lo stesso ma applicato al caso pacchetto 1. Quindi si cicla all'infinito



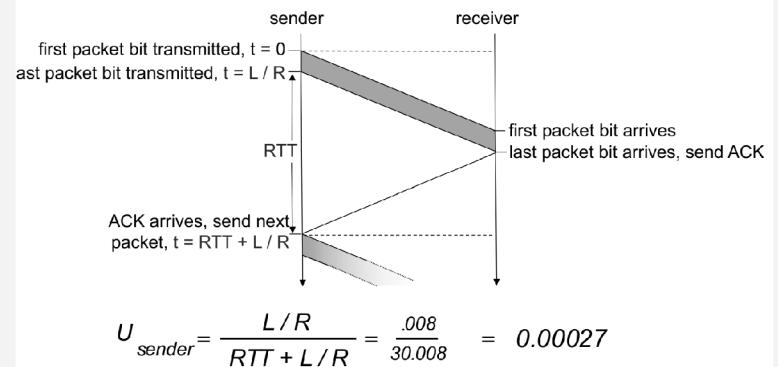
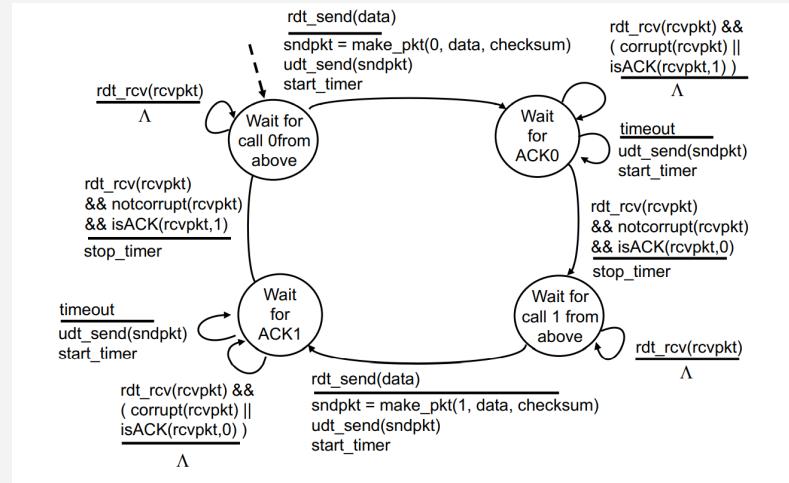
alternando i pacchetto 0 e 1. A questo punto lato receiver il protocollo aspetterà i pacchetti 0 e 1. Se è nello stato “aspetta pacchetto 0” e gli arriva un pacchetto non corrotto ma con numero 1, allora questa è la ritrasmissione di un pacchetto precedentemente arrivato, quindi si ritrasmette un ACK (a cui si associa anche un checksum). Se arriva un pacchetto corrotto si spedisce esplicitamente un NAK sempre con checksum. Se invece arriva un pacchetto corretto con numero di sequenza 0 si trasmette l'ACK e lo si spedisce con deliver_data all'applicazione. A questo punto si passa nello stato “aspetta pacchetto 1” che è la versione duale del caso precedente. Bastano solo due bit perché è sufficiente differenziare i pacchetti tra quello attuale e quello precedente, in quanto il sender è definito nella modalità **stop and wait**, di conseguenza può trasmettere o il pacchetto attuale o quello precedente nel caso non sia arrivato un ACK corretto. Nella versione 2.2 del rdt vogliamo eliminare il NAK. Questo è possibile facendo sì che il receiver invii solamente l'ACK dell'ultimo pacchetto ricevuto correttamente, e quindi si deve esplicitamente dire il numero di sequenza del pacchetto nell'ACK che inviamo. In questo caso sono possibili duplicati del ACK, che lato sender verranno interpretati come NAK. Quindi lato sender, si aggiorna il controllo dicendo che se siamo nello stato 0 e ci arriva l'ACK identificato con numero 1, allora vuol dire che l'ultimo pacchetto corretto arrivato era il pacchetto 1 e quindi dobbiamo ritrasmettere. Lato receiver, se ci arriva un pacchetto corrotto oppure un pacchetto di una sequenza errata, allora basterà inviare l'ACK con numero di sequenza dell'ultimo pacchetto arrivato correttamente.

Se ora il canale può avere anche i pacchetti, allora progettiamo il protocollo rdt 3.0, inoltre possiamo perdere sia i dati

che l'ACK. In questo caso bisogna stabilire un intervallo di attesa temporale, ovvero un timeout, per decidere se rispedire un pacchetto che è andato perduto. Dobbiamo cercare di tarare il timeout in modo che un pacchetto se impiega più tempo per arrivare ma non è andato comunque perduto questo non sia ritrasmesso. Quindi in questo caso, se si riceve un ACK sbagliato o un ACK fuori sequenza non bisognerà fare nulla, perché non appena scatterà il timeout il pacchetto verrà rinviato e sarà fatto ripartire il tempo di attesa. Se arriva l'ACK

corretto allora non faremo altro che passare allo stato successivo, e se dovessero arrivare pacchetti allora non faremo nulla perché si riferiscono allo stato precedente. Questo potrebbe succedere perché ad esempio il timer viene settato troppo breve, quindi potevano esserci pacchetti in sospeso.

Il problema del protocollo rdt 3.0 è che anche se rende la rete sicura, utilizza una frazione piccolissima delle sue potenzialità'. Se definiamo l'utilizzo del mittente (o del canale) come la frazione di tempo in cui il mittente è stato

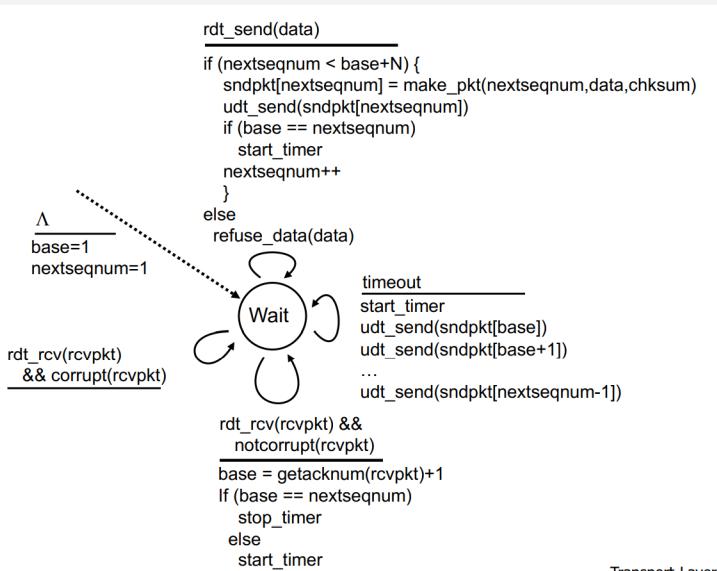
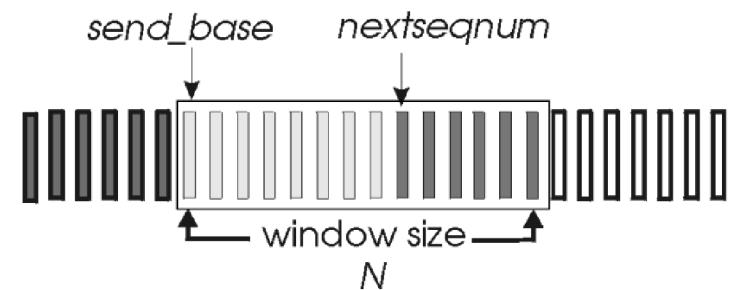


effettivamente occupato nell'invio di bit sul canale, se il RTT e' molto più grande del ritardo di trasmissione L/R, rischieremmo di sprecare troppo tempo a causa dei protocolli di rete (la quantità L/R diviso RTT + L/R e' l'effettivo utilizzo della rete). La rete così fatta trasporta solo un pacchetto alla volta end-to-end. Quindi dobbiamo utilizzare dei protocolli che permettono di trasmettere più dati in parallelo (**pipelining**).

Le due forme di pipelining implementabili sono **go-Back-N** e **selective repeat**. Se spediamo 3 pacchetti alla volta ad esempio, il ritardo di trasmissione dei 3 pacchetti e' 3L/R, il RTT e' lo stesso ma non appena riceveremo l'ACK del primo pacchetto potremo inviare il successivo (quindi l'utilizzo effettivo della rete lato sender diventa 3L/R diviso RTT + L/R che e' tre volte maggiore del precedente). Il protocollo go-Back-N permette di inviare N pacchetti alla volta prima di ricevere l'acknowledgment. Il ricevente non invia un ACK alla volta ma invia un ACK cumulativo. Il sender setta il timer solo per il pacchetto più vecchio che non ha ancora ricevuto l'ACK. Quando questo timer scade vengono rinviiati tutti i pacchetti ancora unacked, perché se il primo pacchetto e' andato perduto il receiver dovrà bufferizzare i pacchetti successivi e questo richiede grossa memoria se fatto per ogni socket. In questo modo se non arriva il pacchetto più vecchio il receiver può non bufferizzare i pacchetti successivi perché saranno ritrasmessi dal sender. Nel protocollo Selective Repeat possono essere mandati N pacchetti la volta ma vengono mandati un ACK per ogni singolo pacchetto e quindi va mantenuto un timer per ogni pacchetto che non e' arrivato. In questo caso serve un buffer per mantenere tutti i pacchetti che sono arrivati prima del primo pacchetto della sequenza.

Go-Back-N funziona in questo modo: nell'header del pacchetto si aggiunge un numero di sequenza. Abbiamo una finestra scorrevole di dimensione N che indica il massimo numero di pacchetti di cui non abbiamo ancora

ricevuto l'ack che possiamo inviare in parallelo. Di questi pacchetti di alcuni abbiamo già ricevuto l'ack e quindi sono già stati trasmessi. Poi abbiamo alcuni pacchetti, al più N che sono stati trasmessi ma di cui non abbiamo ancora ricevuto l'ACK. Potenzialmente nella finestra scorrevole ci potrebbero essere degli spazi vuoti, ovvero se dall'applicazione non arriva nulla e i pacchetti trasmessi ma senza ACK sono meno di N, la differenza sono gli spazi vuoti. I pacchetti oltre la finestra scorrevole non possono essere utilizzati. Se arriva un ACK(n), questo significa un ACK cumulativo che include implicitamente l'ACK di tutti i pacchetti fino ad n. Si mantiene il timer per il primo pacchetto nella finestra scorrevole. Se non arriva l'ACK di conferma per il pacchetto più vecchio e finisce il timer, tutti i pacchetti dal primo fino all'ultimo nella finestra scorrevole vengono ritrasmessi. Con base



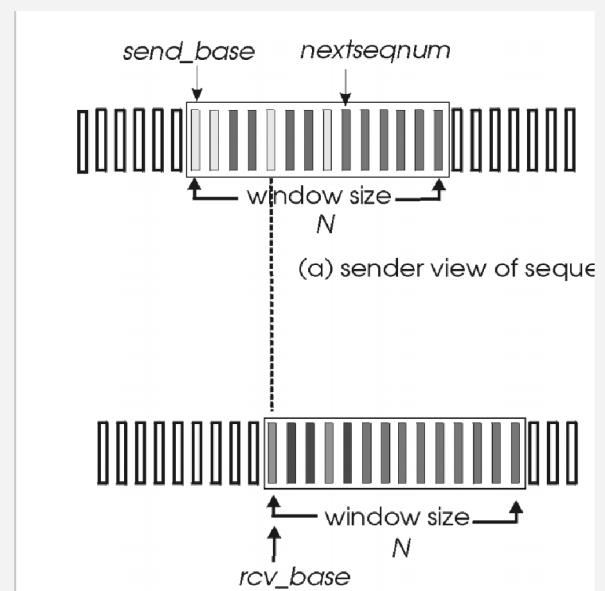
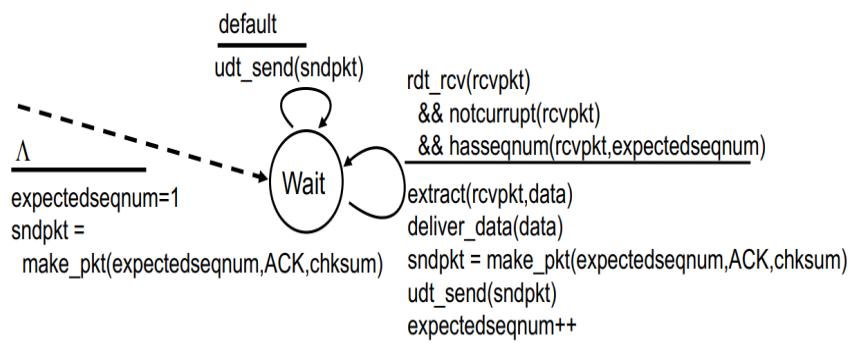
indichiamo il primo elemento inviato di cui non abbiamo ancora ricevuto l'ACK, con nextseqnum indichiamo il primo elemento dello spazio ancora disponibile nella sliding window. Inizialmente il protocollo e' nello stato wait. Quando dall'applicazione viene chiamata rdt_send, lui controllerà se c'e' spazio nella sliding window, ovvero se il

valore nextseqnum e' minore di base + N. In tal caso lui creerà un pacchetto associandoli il numero di sequenza dettato da nextseqnum, se base e' uguale a nextseqnum, ovvero se il pacchetto mandato e' il primo della sliding window inizierà il timer, ed infine incrementera' nextseqnum. Se la sliding window e' piena si rifiuta momentaneamente il data in ingresso.

Se scade il timeout, tutti i pacchetti da base fino a nextseqnum-1 vengono rinviati. Quando si riceve un ACK corretto, si estrae il numero di ACK e si fa scorrere la finestra incrementando la base, ovvero ponendo come primo elemento della sliding window il pacchetto con numero di sequenza extractACK+1. Se in seguito a quest'operazione la sliding window risulta vuota (ovvero base = nextseqnum) si stoppa il timer, altrimenti si fa ripartire da capo. Nel caso di ACK corrotti non si fa nulla e si aspetta il termine del timer per ritrasmettere. Lato receiver, durante il three way handshake viene impostato il valore extectedseqnum, ovvero il numero di sequenza atteso come primo pacchetto. A questo punto, quando si riceve un pacchetto con numero di sequenza identico a quello atteso, corretto senza errori, allora si invia all'applicazione (non c'e buffering) e si manda l'ACK indietro, per poi incrementare l'expectedseqnum. In caso contrario si manda indietro sempre l'ACK dell'ultimo pacchetto correttamente ricevuto. In questo modo il mittente capisce che il problema e' che il primo pacchetto della finestra scorrevole non e' arrivato e quindi bisogna rimandare tutta la finestra scorrevole (gli ACK arrivano in "ordine").

Nel caso di selective repeat si inviano gli ACK per ogni pacchetto giunto a destinazione.

Quindi lato sender c'e' bisogno di importare il timer per ogni pacchetto inviato, così da rinviare solo quelli di cui non si riceve l'ACK. Lato receiver c'e' bisogno di un buffer per memorizzare i pacchetti arrivati fuori ordine, mentre quelli che arrivano in ordine possono essere mandati verso l'applicazione. A questo punto nella slides window del sender potremo avere dei pacchetti di cui abbiamo ricevuto l'ACK intervallati da pacchetti inviati senza ACK ricevuto. La finestra scorrevole scorre se si riceve l'ACK del primo pacchetto unACKed nella finestra scorrevole. Se arriva un pacchetto in ordine lo si manda all'applicazione, altrimenti viene memorizzato

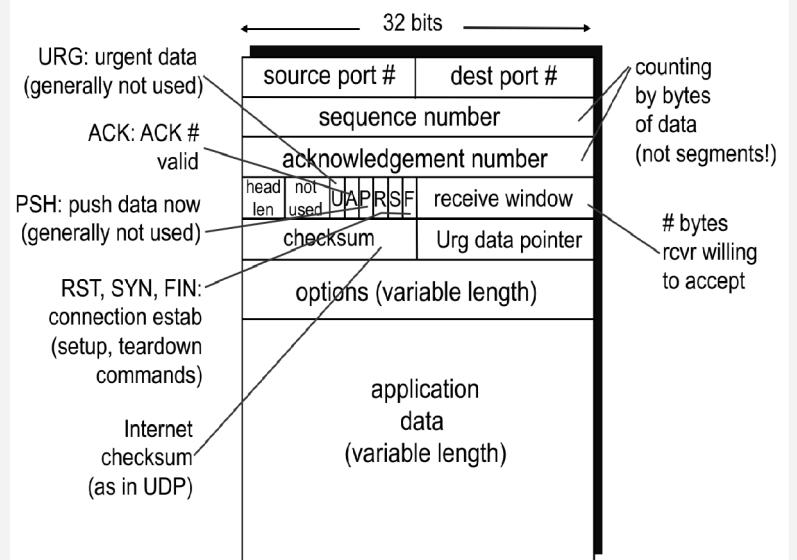


in un buffer. Quando si riceve il pacchetto corrispondente alla posizione `recv_base`, che indica il primo pacchetto nell'ordine non ancora ricevuto, tutti i pacchetti successivi a `recv_base` già arrivati vengono spediti all'applicazione e si scorre la finestra fino al primo pacchetto non ancora arrivato. Se arriva un pacchetto precedentemente inviato all'applicazione puo' significare che il relativo ACK e' andato perduto, quindi si rimanda l'ACK del rispettivo pacchetto. Un problema del selective repeat e' che tra il sender e il receiver vi e' un sipario e l'uno non sa cosa succede dal lato dell'altro. Se impostiamo male i numeri di sequenza potrebbe succedere che lato receiver la finestra scorrevole scorre, ma se gli ACK vanno perduti il sender invia pacchetti con un numero di sequenza che puo' essere accettato dal receiver ma che non corrisponde al reale pacchetto atteso ma ad uno già arrivato, e il receiver non può sapere questa cosa. Per evitare questo bisogna usare numeri di sequenza molto maggiori rispetto alla dimensione della slide window.

TCP

TCP e' un protocollo che stabilisce una connessione punto-punto in cui non ci sono limiti per la dimensione dei messaggi, ed e' basata su pipeline ma il controllo di flusso decide la dimensione della slide window. Permette inoltre di scambiare dati nello stesso flusso in modo bidirezionale. **MSS** (maximum segment size) indica la dimensione massima del segmento. Quando si inizializza la connessione avviene un handshake per setizzare le informazioni cruciali per avviare la connessione. Un pacchetto TCP e' strutturato in questa maniera:

- 16 bit per source port e 16 bit per destination bit.
- 32 bit utilizzati per il numero di sequenza, ovvero il primo byte scambiato nel segmento (ad esempio se un file e' di 50000 byte e MSS=1000, il primo segmento ha numero 0, il secondo 1000 e così via).
- Acknowledgment number che contiene il primo bytes atteso dall'altro lato. Ad esempio che A ha ricevuto da B i bytes da 0 a 535 in questo campo si scriverà 536. Significa che i bytes da 0 a 535 sono stati ricevuti e quindi e' un ACK cumulativo.
- altri 16 bit sono usati per indicare la grandezza dell'header, alcuni non sono utilizzati e poi ci sono 6 bit di cui il primo indica se il pacchetto contiene dati urgenti, il secondo indica se il pacchetto contiene un ACK, il terzo e' il push bit e significa che sono dei dati necessari all'applicazione e quindi vanno spediti all'applicazione anche se fuori ordine, poi gli ultimi 3 bit servono per l'handshake: il primo (RST) se e' ad uno significa che il server non vuole accettare o vuole buttare giù la connessione, il secondo (SYN) per inizializzare una nuova connessione o la risposta ad una



Transport Layer 3-58

richiesta (in questo caso va ad 1 anche il bit dell'ACK), il terzo (FIN) si setta ad uno per richiedere di finire una connessione che va confermata anche dall'altro lato.

- 16 bit che indica il numero di bytes che il ricevente e' disponibile ad accettare, ovvero il buffer residuo (receive window).
- 16 bit di checksum e altri 16 che indicano nel caso in cui vi siano dati urgenti da dove vengono considerati importanti i byte in quel pacchetto.
- campo di options di dimensione variabile ed infine i data.

Il protocollo TCP non gestisce i pacchetti arrivati fuori ordine, quindi spetta a noi come gestirli. TCP deve settare il valore di timeout che deve essere più grande del RTT, tuttavia quest'ultimo e' variabile. Quindi si fa un campionamento del RTT, ma questo **sampleRTT** puo variare da un test all'altro. Perciò si calcola un tempo di RTT stimato che viene calcolato con la formula $(1-a) * \text{estimated RTT} + a * \text{sampleRTT}$, dove estimated RTT e' il tempo stimato al passo precedente, sampleRTT e' il campionamento calcolato con il test attuale ed a e' una costante spesso pari a 0.125, in questo modo il test attuale influenzera' solo per $\frac{1}{8}$ il tempo di RTT stimato. Il timeout viene impostato quindi maggiore rispetto al tempo di RTT stimato, in particolare di un certo margine di grandezza. Questo valore detto **devRTT** e' calcolato attraverso la formula $(1-b) * \text{devRTT}(\text{passo -1}) + b * \text{abs}(\text{estimated RTT} - \text{sampleRTT})$. Questo ultimo valore assoluto indica la distanza tra quanto sta cambiando l'RTT e il valore "medio" dell'RTT, e il devRTT cambia per solo $\frac{1}{4}$ in base a questo valore, perche' b spesso e' 0.25. Quindi il timeout e' dato dall'estimatedRTT + 4 * devRTT, 4 cosi da avere abbastanza margine. TCP lato sender inizialmente e' in attesa di dati dall'alto.

Sendbase e Nextseqnum vengono impostati con un valore intialseqnum che viene deciso durante il three way handshake. Quando vengono passati dati dall'applicazione si costruiscono i pacchetti basandosi su Nextseqnum e gli si spedisce. Nextseqnum si aggiorna aggiungendo la lunghezza dei dati che abbiamo mandato. Se il timer e' fermo viene fatto partire. Se il timeout scade si ritrasmettono tutti i segmenti a partire da quello con il più basso numero di sequenza e viene fatto ripartire il timer. Se riceviamo un ACK con valore $y > \text{Sendbase}$ che significa che riceviamo un ACK di un segmento successivo del primo di cui aspettiamo ancora l'ACK. Quindi e' un ACK cumulativo e sendbase viene impostata = y . Se a questo punto ci sono ancora segmenti non ACK si resetta il timer, altrimenti si stoppa.

Le politiche di generazione di ACK lato ricevente dipendono da diverse casistiche. Se arriva un segmento con numero di sequenza atteso e tutti i segmenti precedenti hanno già mandato ACK, allora si aspetta il segmento successivo usando un ACK ritardato. Se dopo 500ms non e' arrivato alcun segmento si spedisce l'ACK.

Se arriva un segmento con esatto numero di sequenza e c'e' un altro segmento ordinato in attesa di trasmissione ACK, viene mandato l'ACK solo per l'ultimo segmento arrivato realizzando così un ACK cumulativo.

Se arriva un segmento fuori sequenza si manda un ACK duplicato relativo al primo bytes atteso in sequenza.

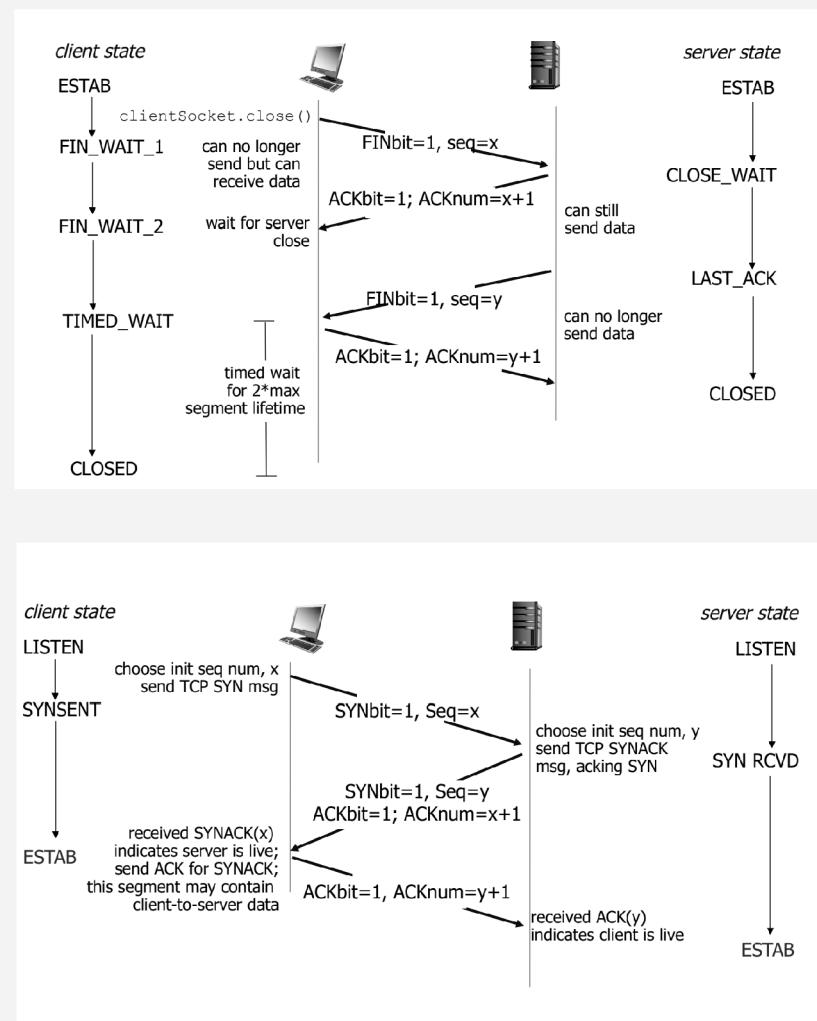
Se arriva un byte che colma parzialmente o completamente un buco si invia un ACK se e solo se il segmento coincide con l'inizio del buco.

Se mandiamo una serie di pacchetti ma riceviamo 3 volte un ACK duplicato, significa che la rete funziona (perché sono arrivati 3 pacchetti che hanno scaturito l'invio degli ACK duplicati) quindi non c'e' congestione, ma probabilmente solo il primo segmento e' andato perduto. Quindi questo spinge lato sender a inviare il più velocemente possibile il primo pacchetto senza rallentare l'invio dei dati perché non ce' congestione (**fast retrasmmitting**).

Ogni socket ha allocato un buffer in cui vengono memorizzati i segmenti in arrivo dalla rete. Questi vengono poi letti e passati all'applicazione e quindi il buffer verrà successivamente sovrascritto. Tuttavia se l'applicazione ad esempio richiede i dati troppo lentamente mentre arrivano molto velocemente dalla rete, e non abbiamo un buffer molto grosso potremmo perdere i pacchetti. Quindi questo richiede un controllo del flusso (**flow control**). Per ogni socket dobbiamo allocare un buffer che chiamiamo rcvbuffer che ha una dimensione tipicamente di 4096 bytes. Questo buffer al momento t avrà una certa quantità di memoria occupata da dati ancora non letti e uno spazio libero che chiamiamo **receive window** (rwnd). Il ricevente deve comunicare la grandezza di questa finestra al sender così che esso regoli la grandezza della finestra scorrevole e non la faccia più grande di quanto il buffer possa sopportare, questo garantisce che non vi sia overflow nel buffer del ricevente.

Per creare una connessione TCP bisogna prima stabilire una connessione effettuando un handshake. Durante questa fase sia lato client che server verranno importanti informazioni importanti come i numeri di sequenza e le dimensioni dei buffer. Una connessione a due vie presenta diversi punti di fallimento, ad esempio se effettuiamo la richiesta di connessione ma la conferma arriva troppo tardi dal server, potremmo nel mentre mandare un'altra richiesta. Ma se arriva la conferma della prima e nel mentre finiamo la connessione, e al server arriva anche l'altra richiesta, questo potrebbe accettarla ma sarebbe una connessione aperta solo lato server. Ancor peggio se nel mentre inviamo dei dati che vengono accettati ma li ritrasmettiamo perché scade il timeout e nel mentre finiamo la connessione ma arriva l'altra richiesta inviata, questa potrebbe essere accettata e potrebbero persino arrivare i dati che avevamo ritrasmesso due volte.

Il processo per stabilire una connessione è: l'applicazione lato client dice al protocollo TCP di voler iniziare una connessione verso un server. Questo fa sì che TCP mandi un segmento contenente $SYN=1$ al server con numero di sequenza x ed entra nella fase **SYN_SENT**. In questa fase attende un ACK dal server che certifichi che la connessione è avvenuta. Quando tale pacchetto è ricevuto lato server, questo genera un pacchetto con $SYN = 1$, $ACK = x+1$ e $seq = y$. Questo pacchetto è chiamato anche **SYNACK** e quando arriva al client per lui la connessione è **ESTABLISHED**. Durante questa fase può inviare e ricevere tutti i dati che vuole. Supponiamo che il client adesso voglia chiudere la connessione. Questo spinge ad inviare un segmento con $FIN = 1$ ed entra nella fase **FIN_WAIT1**. Durante questa fase attende un segmento con un ACK e quando lo riceve entra nella fase



FIN_WAIT2 in cui attende un segmento dal server con FIN = 1. Dopo questo momento il TCP manda un ACK ed entra in TIME_WAIT in modo da poter rimandare eventualmente questo ACK. Questa fase dura dai 30 secondi fino ai 2 minuti generalmente, dopodiché la connessione è chiusa definitivamente. Anche nel caso in cui sia client che server mandino FIN simultanei la situazione può essere gestita correttamente.

Controllo della congestione

Congestione vuol dire ritardi lunghi e perdite di pacchetti. Supponiamo due sorgenti A e B che condividono un link ed inviano dati a C e D passando per un unico router con capacità del buffer infinita. La velocità dell'output link è di R e sia A che B producono pacchetti alla velocità di V bit/s. Il throughput cresce linearmente in funzione di V, tuttavia quando V arriva ad R/2, ovvero la velocità massima di connessione condivisa, il throughput non può crescere più semplicemente perché è il massimo che l'output link può supportare. Il problema diventa che man mano che V si avvicina ad R/2 il ritardo nel router inizia ad aumentare e per V che tende ad R/2 tende ad infinito. Ipotizzando che ora il router abbia buffer finito, V ha lo stesso significato di prima ma introduciamo V' che è il tasso di invio di pacchetti originali + pacchetti ritrasmessi, visto che ora possono andare perduti. Vogliamo fare in modo che non ci siano ritrasmissioni, ovvero andando piano e cercare un punto in cui spingiamo il più possibile senza congestionare il router. Infatti se ci sono ritrasmissioni la rete viene usata non solo per spedire dati originali ma anche dati ritrasmessi, quindi aumentando V, non è detto che il throughput aumenti linearmente con V, perché parte dei dati inviati potrebbero essere pacchetti andati precedentemente perduti e che quindi facciano sprecare risorse (questo nel caso in cui V' > V). In un caso ideale noi dovremmo sempre avere spazio libero sul buffer dei router, e vorremmo che la somma dei dati inviati dagli host sia inferiore del buffer intermedio. Purtroppo non conosciamo né quanto buffer il router intermedio abbia né quanti pacchetti perde. Il congestion control di TCP parte inizialmente basso, ovvero il valore N della finestra inizialmente è basso. Se noi riceviamo gli ACK dei pochi pacchetti possiamo iniziare ad aumentare N, se invece almeno un pacchetto è stato perduto si riduce la finestra drasticamente. Grazie a questo crollo della velocità di invio permettiamo al router di liberare il buffer che si era riempito. Il funzionamento di TCP è quindi, mandare un numero di byte pari alla finestra scorrevole, aspettare un RTT e poi mandare ancora più bytes, quindi il sending rate è approssimativamente pari alla dimensione finestra scorrevole in bytes/RTT in secondi. La tecnica **slow start** fa raddoppiare ogni volta la dimensione della finestra. Se scatta un timeout si richiude la finestra al valore 1. Se noi abbiamo 3 ACKs duplicati, vuol dire che manca un pacchetto al destinatario ma la rete sta funzionando ancora, quindi la slide window viene dimezzata non azzerata completamente (TCP **Reno**). Nella forma TCP **Tahoe** si ripristina sempre la dimensione della finestra ad 1 sia se scatta un timeout sia se riceve 3 pacchetti duplicati. Inoltre con TCP si setta una variabile di ambiente detta **ssthresh** per dire che da lì in poi rischiamo la perdita di pacchetti, quindi da ssthresh in poi TCP aumenta di 1 non raddoppiando la finestra scorrevole (**congestion avoidance**). Supponiamo di avere dimensione di pacchetti 1500 bytes, RTT 100ms e volessimo 10Gb di throughput. Questo richiederebbe 83 pacchetti spediti contemporaneamente, perché possiamo spedire 830 pacchetti al secondo. Per ottenere tale throughput si utilizza la formula $1.22 \times \text{valore dato} \times \text{MSS (1500)} / \text{RTT} \times \text{rad(L)}$ dove L è la percentuale di perdita pacchetti.

TCP presenta un problema di fairness, ovvero dati due host che condividono risorse (buffer dei router) essi iniziano nella fase slow start a mandare pacchetti sempre più numerosi esponenzialmente. Poi il primo host che rallenta perché perde un pacchetto inizia a riaumentare il ritmo, tuttavia è bloccato in questa impresa dall'altro host che non ha mai fermato il suo ritmo di invio e che anzi continua ad aumentarlo, consumando la maggior parte delle risorse. Un altro problema è che pacchetti che utilizzano UDP ignorano la congestione della rete, quindi l'applicazione non fa nulla per rallentare i pacchetti UDP, mentre quelli TCP devono gestire questa problematica. Una delle funzioni implementate a livello rete per gestire il congestionamento è l'Explicit Congestion Notification, che prevede che nei pacchetti a livello 3 sia presente un campo detto ToS che se marcato notifica ai router vicini che quel router è congestionato. Tuttavia questo lo vedono solo i router ma non le sorgenti quindi non fa altro che spostare il punto di congestione.

CAPITOLO IV

Prologo capitolo reti

A livello tre, la rete diventa una rete di reti gestita gerarchicamente. Per fare questo abbiamo bisogno di un nuovo indirizzamento, l'indirizzo IP esso permette di identificare univocamente un dispositivo all'interno di una rete. Grazie all'indirizzo IP si può astrarre la struttura di una rete locale, e per l'instradamento dei pacchetti basta un rappresentante, ovvero un router, che ne fa le veci dall'esterno. L'indirizzo IP è formato da 4 ottetti di bit, con valori da 0 a 255. Vi è un'associazione univoca tra indirizzo IP e scheda di rete. Più schede di rete hanno associato un indirizzo IP per ognuna. Se ad un indirizzo MAC viene associato sempre lo stesso IP si chiama indirizzo IP statico, altrimenti dinamico. Dall'indirizzo IP si possono individuare più **classi di reti**:

- Le reti di classe A sono al massimo 126 e ognuna può contenere fino a oltre 16 milioni di host. Per le reti di classe A, il byte di indirizzo più significativo (a sinistra) ha sempre il primo bit uguale a zero, e può assumere i valori da 1 a 126 (network number) rispetto ai 128 valori possibili. I tre byte rimanenti possono assumere oltre 16 milioni di combinazioni, ognuna associabile a un host della rete.
- Le reti di classe B sono al massimo 16.382 e ognuna può contenere fino a oltre 64.000 host. Per le reti di classe B, il network number è dato dai due byte di indirizzo più significativi (a sinistra), che hanno sempre i primi due bit uguali alla coppia (uno,zero). I network number di classe B possono assumere i valori da 128.0. a 191.255. I due byte rimanenti (host number) possono assumere oltre 64.000 combinazioni, ognuna associabile a un host della rete.
- Le reti di classe C sono oltre 2 milioni, e ognuna può contenere fino a 254 host. Per le reti di classe C, i tre byte di indirizzo più significativi (a sinistra) rappresentano il network number, e hanno sempre i primi tre bit uguali alla terna (uno,uno,zero). I network number di classe C possono assumere i valori da 192.0.0 a 223.255.255. Il byte rimanente (host number) può assumere 254 combinazioni utili, su 256 possibili, ognuna associabile a un host della rete.

Un indirizzo IP può essere diviso in due componenti logiche, una che individua il numero di rete, una il numero di host nella rete, questo grazie alla maschera di rete. Per istruire ogni router subordinato sulla dimensione e sull'interpretazione degli indirizzi IP da amministrare,

ogni router subordinato deve essere fornito di una **maschera di rete** (netmask). Per ogni dispositivo di rete o calcolatore connesso a una rete IP (es. Internet), la maschera di rete, l'indirizzo del default router e l'indirizzo IP, costituiscono i tre parametri fondamentali di configurazione del protocollo IP, e devono essere forniti, al momento della connessione, al livello IP. La trasmissione su un segmento di rete locale avviene mediante frame di livello MAC/LLC, indirizzati mediante gli indirizzi MAC dei dispositivi, e non attraverso gli indirizzi IP. Un router deve quindi saper gestire l'associazione tra indirizzo IP di un dispositivo a livello rete e il suo indirizzo MAC a livello MAC/LLC. Il protocollo Address Resolution Protocol (**ARP**) risponde in modo standard a questa esigenza. In altre parole il protocollo ARP è il protocollo che lega l'indirizzamento a livello MAC con l'indirizzamento a livello IP. Il router spedisce sui segmenti della rete locale un frame in broadcast (cioè ricevuto da tutti i dispositivi) contenente il codice di richiesta ARP, e l'indirizzo IP del destinatario del pacchetto. Tale frame equivale quindi al rivolgere a tutti i dispositivi la domanda: "quale indirizzo MAC ha il dispositivo corrispondente al seguente indirizzo IP"? Il dispositivo in questione, se esiste, risponde con un frame indirizzato all'indirizzo MAC del router, contenente il codice di risposta ARP, e con allegato l'indirizzo MAC richiesto. Esiste anche una versione analoga del protocollo ARP, detta **Reverse-ARP**, che risponde alla domanda: "quale indirizzo IP corrisponde al dispositivo con questo indirizzo MAC"?

Le funzioni del livello rete sono il **forwarding** e il **routing**. Il data plane e' un insieme di funzioni implementato in ogni router. Queste funzioni devono determinare le destinazioni di ogni pacchetto, e fare in modo che ogni pacchetto che arriva in una porta di input deve essere inoltrato in una porta di output il più velocemente possibile. Il control plane e' una logica del livello rete, quindi non di un singolo router ma dell'intera rete. Determina come scrivere le tabelle dei router in modo che siano il più efficienti possibile. Ci sono due approcci per realizzare le funzioni di control plane, uno che prevede di implementarle in ogni router, un'altro di implementarle a livello software gestito da un server remoto. La prima consiste nel dotare di ogni router di un algoritmo di routing. Il router conosce la situazione della rete attorno a se, quindi puo usare questi algoritmi per scrivere le tabelle di inoltro localmente. Nella versione centralizzata, i routing algorithm non sono eseguiti da ogni router intermedio, ma piuttosto ogni router e' dotato di un control agent che comunica con un server remoto che conosce lo stato generale della rete e di conseguenza comunica con i router costantemente per tenere aggiornate le tabelle di inoltro.

Cosa c'e' in un router

Un router prevede più input/output port, una switching fabric ad alta velocità, ovvero una centrale di commutazione che connette le porte di input con quelle di output, e un routing processor, sul quale girano gli algoritmi di routing. La switching fabric impiega nanosecondi mentre il routing processor opera nell'ordine di millisecondi. Le funzioni di una **porta di input** sono un terminatore di linea per il link comunicativo che passa i dati da elaborare al

layer protocol del livello 2 ed infine in un buffer di ricezione per i dati in arrivo sulla porta. Per inoltrare i pacchetti in coda si usano i campi di header e si fa un'operazione di lookup il più velocemente possibile per decidere che porta usare consultando le tabelle di inoltro (match plus action). Vorremmo che queste operazioni siano fatte alla velocità del collegamento, ovvero come se il pacchetto viaggiasse sulla linea mentre è elaborato. Se arrivano più velocemente di quanto possano essere elaborati si accodano e di conseguenza possono andare perduti. Il forwarding può essere fatto sia semplicemente guardando la sola rete di destinazione sia guardando anche altre informazioni di header (generalized forwarding).

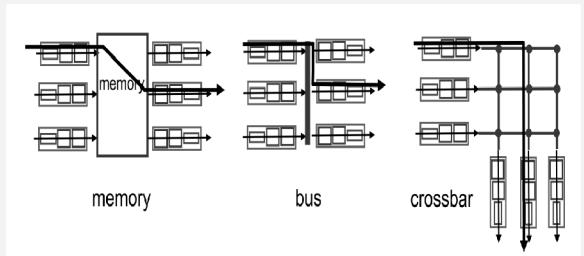
Una tabella possibile ma troppo lenta è costituita da righe che contengono un range di indirizzi IP a cui corrisponde un destination port. Un'alternativa è non scrivere esplicitamente tutti i bit di una riga (lasciando asterischi), confrontando i prefissi e facendo match con la riga che ha il prefisso più lungo che faccia match (**longest prefix matching**). Esistono memorie che permettono di confrontare in un unico ciclo di clock tutte le righe della tabella. La **switching fabric** realizza il trasferimento da un input buffer ad un output buffer. Può essere implementata tramite una memoria, un bus o a crossbar con circuiti elettrici. Lo switching rate è il ritmo con il quale i pacchetti possono essere trasferiti da input ad output. Se la switching fabric ha un rate lento può diventare un collo di bottiglia. I router di prima generazione erano computer con funzioni di switch, quindi avevano una **memoria interna** in cui venivano messi i pacchetti interni che venivano letti dal processo che gestiva i pacchetti in uscita. Questo sistema ha un collo di bottiglia nella fase di

lettura/scrittura. Switching **via bus** utilizza un bus condiviso tra le varie porte di input e le varie porte di output. Il problema è la contesa per utilizzare il bus che può essere utilizzato uno alla volta. Nella connessione a **crossbar** vi è una rete programmabile che viene comandata similmente a dei ponti elevatoi che si alzano o abbassano in base a dove devono andare i bit. Il vantaggio è che è possibile un grado di parallelismo se non si incrociano i dati in transito. Inoltre permettono di frammentare i datagram in pezzetti chiamati celle che quindi possono essere trasferiti a gruppi anche contemporaneamente.

Head-of-line blocking: il primo datagram nella coda impedisce ai datagram dietro di muoversi. Inoltre se due pacchetti all'inizio di due code di ingresso sono destinati verso la stessa porta di uscita sperimentano una contesa di quest'ultima perché solo uno dei due può essere trasferito.

Nell'**output port** c'è sempre una coda e un'architettura speculare rispetto a quella dell'input port. I pacchetti vengono presi dalla coda con un algoritmo di scheduling, con politiche FIFO o priority. Qualunque algoritmo di scheduling prevede vantaggi e svantaggi. Si verifica buffering nelle code di uscita quando la switching fabric fa arrivare dati più velocemente dell'output link e questo può causare congesti e perdita di pacchetti. Il buffer della coda di uscita viene tarato con la formula $RTT \text{ medio} * \text{capacità link} / \text{rad}(N)$ dove N sono i flussi possibili sul router, e visto che da ogni coda di input si può andare ad ogni coda di output si usa la radice perché c'è un andamento quadratico.

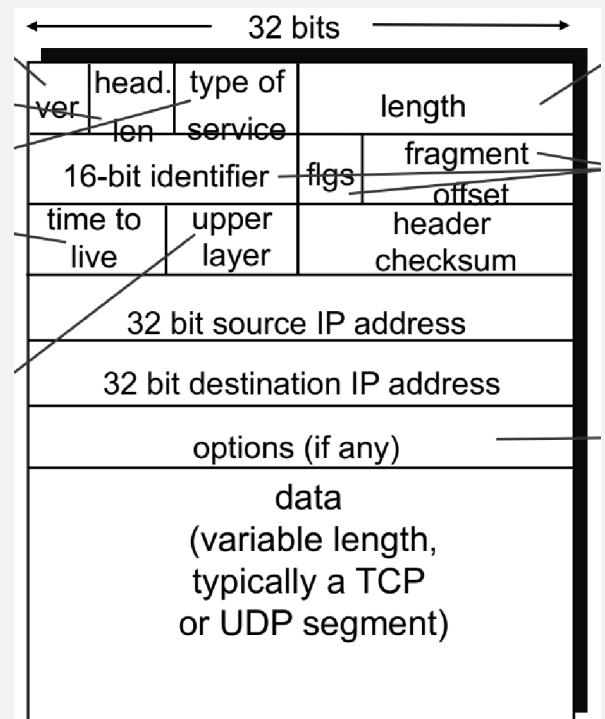
I meccanismi di scheduling consistono nel modo in cui scegliere quale e quando spedire il prossimo pacchetto della coda. La politica **FIFO** spedisce quelli che arrivano per primi. Quando si stabilisce una coda FIFO si deve decidere nel caso in cui fosse piena qualche pacchetto scaricare. **Tail drop** si scarica chi arriva, priority drops in base alla priorità più bassa, random rimuove uno a caso. Nel caso di **priority scheduling** abbiamo due code per



ogni output, una per i pacchetti urgenti una per i pacchetti meno urgenti. Se c'e' almeno un pacchetto nella coda di priorità alta si sceglie lui, altrimenti uno dalla coda normale. Il rischio del priority scheduling e' che quelli a bassa priorità subiscano grossi ritardi. Nella politica **round robin** si suddividono i pacchetti in più classi e ciclicamente si scansionano le code di ogni classe e si manda, se possibile, un pacchetto di ognuna. Il vantaggio e' che non ha azione bloccante come la priority nei confronti di pacchetti di bassa priorita'. Infine la politica **weighted fair queuing**, basata su round robin, da' diverse priorità ad ogni coda, quindi magari si puo dare il doppio di priorità ad una coda rispetto alle altre e mandare i pacchetti con un rapporto 2:1:1 se abbiamo tre classi.

IP

I primi 4 bit indicano la versione se IPV4 o IPV6, il campo successivo la lunghezza dell'header in bytes (per specificare la lunghezza del campo options), poi type of service per indicare la priorità di un pacchetto ma che e' largamente inutilizzato. Poi 16 bit per la lunghezza totale, compreso l'header, del pacchetto in bytes (massimo 2 alla 16 bytes). 16 bit per l'identificatore del pacchetto (una specie di targa), poi i flags (che se a 1 ci dice che questo non e' un pacchetto originale ma solo un frammento) e i fragment offset. Questo viene fatto perché i pacchetti troppo grandi devono essere spezzati a livello 2 in frames più piccoli e dopo vanno riassemblati. Tutti i pacchetti con lo stesso identifier significa che sono pezzi dello stesso pacchetto e l'offset e' utilizzato per capire quale parte del pacchetto e' quella ricevuta. Poi 8 bit per il time to live, che rappresenta il numero massimo di router attraverso cui può passare. L'upper layer e' byte che indica il protocollo di livello trasporto a cui deve essere passato il pacchetto (TCP o UDP). Poi 16 bit di header checksum che controlla solo se l'header e' giusto. Poi 32 bit per source IP e 32 bit per destination IP. Infine il campo option e poi data. L'header totale escludendo i campi options sono 20 per TCP + 20 per IP + l'app overhead. I network links hanno un max transfers size, di conseguenza se arriva un pacchetto di grandi dimensioni, questo va spezzato in frammenti più piccoli, ed ognuno di questi pacchetti ha vita propria nella rete e saranno riassemblati nel datagram originale solo a destinazione, che quindi necessita un buffer per questo lavoro. Se dobbiamo spedire un pacchetto da 3980 (+20 header), in ethernet MTU=1500. Quindi il pacchetto va spezzato in 2 frammenti da 1480 (+20 header) ed uno da 1020 (+20 header). L'identificativo per i frammenti e' lo stesso, e il flag va ad 1. L'offset rappresenta un puntatore al primo byte del frammento ma va misurato a blocchi di 8 bytes, quindi bytes/8. L'ultimo frammento ha flag zero. Per distinguere i pacchetti unici dall'ultimo frammento di un pacchetto basta vedere se sia flag che offset sono 0.



L'indirizzo IP e' un identificatore da 32 bit. Ogni interfaccia di rete ha associato il suo indirizzo IP, gli host hanno generalmente un paio di interfacce, ethernet e wi-fi, mentre un router ha varie interfacce. Se in una rete e' presente uno switch, a livello 3 questo può essere ignorato perché non lavora a livello 3, quindi in una rete si vedono il router e gli host connessi a questo router. Una subnet e' un insieme di dispositivi che si connettono tra loro senza passare da un router, che appunto distingue le varie subnet. **CIDR** indica Classless InterDomain Routing, che permette di identificare porzioni di subnet di dimensione arbitraria con il formato a.b.c.d/x, dove x indica il numero di bit dedicati alla subnet.

Un indirizzo IP puo' essere configurato sia manualmente ma anche automaticamente.

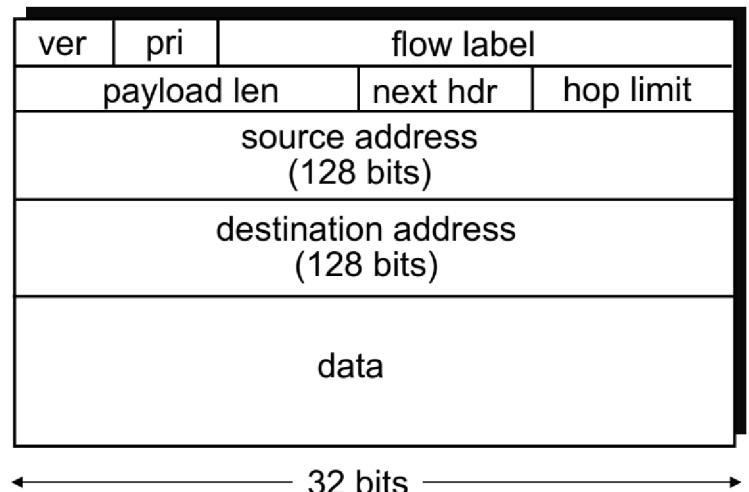
DHCP dynamic host configuration protocol permette di ottenere gli indirizzi dinamicamente quando un host si collega ad una rete, in maniera "plug and play". Ci sono 4 messaggi DHCP, il primo detto DHCP discover viene mandato in broadcast da un client su tutta la LAN quando si connette ad una nuova rete. Tutti i DHCP server che lo ricevono rispondono con un DHCP offer in cui offrono un indirizzo IP e la configurazione per la rete, e mandano questo pacchetto in unicast verso il client. A questo punto il client se riceve più offer effettua un messaggio DHCP request indicando il server di cui ha scelto l'indirizzo IP offertogli e quest'ultimo manda un DHCP ack per confermare l'assegnazione dello stesso. Un DHCP server può ritornare più del solo indirizzo IP della subnet, ad esempio l'indirizzo del primo router per quel client, un nome e l'indirizzo IP del DNS server e la network mask. Una richiesta DHCP viene incapsulata utilizzando UDP, poi in un pacchetto IP, in un pacchetto livello 2 e viene mandata in broadcast sulla LAN. Quando raggiunge il DHCP server, apre tutti i pacchetti e lo manda sulla porta 67/68. Durante una comunicazione DHCP viene assegnato un transaction ID per identificare quella specifica trasmissione e differenziarla dalle altre DHCP. Un ISP che compri un indirizzo di rete, può vendere a delle organizzazioni delle sottoreti specificando la netmask. Se queste sottoreti non presentano buchi, può fornire al control plane di internet tutti gli indirizzi IP appartenenti alla sua rete e sottoreti. Se poi una di queste organizzazioni si spostasse sotto un altro ISP (quindi le sottoreti non hanno più indirizzi contigui), quando si fa richiesta per il control plane bisogna specificare di fornire le informazioni non solo delle sottoreti di quell'ISP ma anche dell'organizzazione che si e' spostata. Quindi idealmente bisogna che gli ISP gestiscono blocchi contigui di indirizzi.

Un router ha almeno due interfacce, una verso la rete privata ed una verso internet. Verso la rete internet il router ha un suo indirizzo IP che gli fornisce l'ISP. A questo punto il router e' un client dell'architettura di internet. Dall'altro lato la rete privata cambia. Qualunque pacchetto da e verso la rete privata viene gestito dal router senza andare fuori. Se invece la destinazione e' esterna verso internet, il router fa evadere il pacchetto dalla rete locale, ed e' facilmente gestibile dal router attraverso il suo indirizzo IP. Tuttavia quando torna un pacchetto indietro si utilizza l'indirizzo IP del router come destinazione. A questo punto entra in gioco il **NAT** creando un numero di porta che definisce chi fa la richiesta e di conseguenza chi deve ricevere la risposta dentro la rete privata. Ogni router ha una tabella di traduzione NAT in cui per ogni richiesta effettuata dalla rete privata verso l'esterno c'e' una corrispondenza tra chi fa richiesta all'interno della rete privata con numero di porta associato e l'indirizzo IP del router che deve inoltrare questa richiesta con associato un numero di porta libera che si inventa il NAT server. Quindi quando dall'esterno torna il pacchetto, questo arriva al livello trasporto del NAT server in cui trova il processo del NAT che traduce l'IP di destinazione con il reale numero di porta e a quel punto invia nella sua rete privata il pacchetto facendo risultare come mittente sempre il server esterno. Il limite e' il numero di porte del NAT server. In questo modo si possono avere fino a 64000 connessione con un

unico "rappresentante" a livello di LAN, tuttavia in questo caso i router che implementano NAT devono prendere pacchetti fino al livello 4. Inoltre il NAT, grazie all'azione di camuffamento della sottorete, protegge gli host che rappresenta in quanto l'unico modo per indovinare un host e' cercare la riga corrispondente nella tabella del NAT, che pero' non e' accessibile a nessuno.

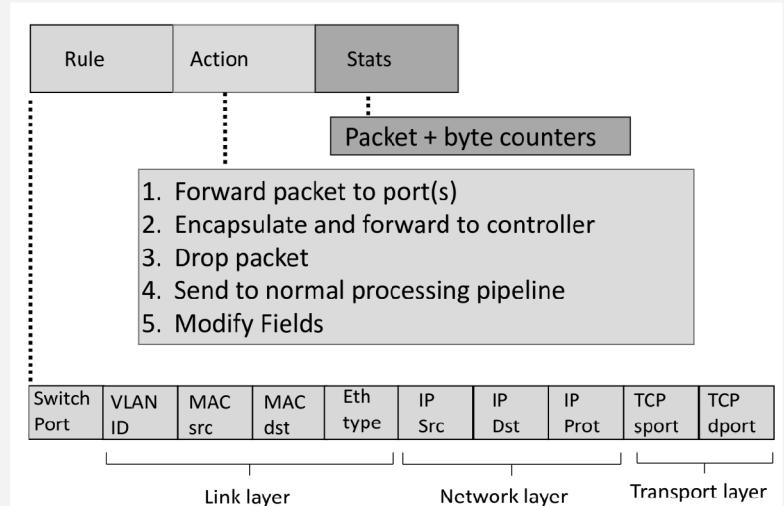
Le motivazioni per l'utilizzo di **IPv6** sono che i 32 bit per l'indirizzamento sono pochi visti tutti i dispositivi connessi ad internet. Inoltre l'overhead dovuto alla frammentazione dei pacchetti IPv4 e' elevato e non fornisce qualità del servizio in IPv4. I vantaggi di IPv6 sono che l'header ha 40 bytes e non c'e' frammentazione. I primi 4 bit sono sempre per la versione, poi in 4 bit IPv6 fornisce un campo addizionale per la priorità dei pacchetti all'interno dei flussi. I restanti 24 bit sono la flow label che identifica un flusso di comunicazione, poi il payload len ovvero la dimensione del campo dati, il next header indica il punto in cui iniziano i bit appartenenti ai protocolli di livello superiori all'interno del campo dati. Il campo hop limit e' equivalente al TTL e infine source address e destination address da 128 bit. Non c'e' più il checksum, il campo options e' gestito fuori dall'header di IPv6 dal next header. ICMPv6 e' una nuova versione di ICMP che prevede un messaggio di errore che notifica la dimensione troppo grande del pacchetto

nel caso in cui superi la dimensione massima del data link. Infatti, non essendoci frammentazione bisogna gestire la dimensione dei datagrammi. In questo caso viene notificata la dimensione massima alla sorgente originale che comincerà a creare pacchetti di quella dimensione, evitando così la frammentazione. Il **tunneling** permette di incapsulare un pacchetto IPv6 in un pacchetto IPv4 trattandolo come payload. Questo e' necessario perché durante il tragitto si potrebbe dover passare per router IPv4. Esistono router speciali capaci di comprendere sia IPv4 che IPv6, così da fare da intermediari tra i due mondi internet.

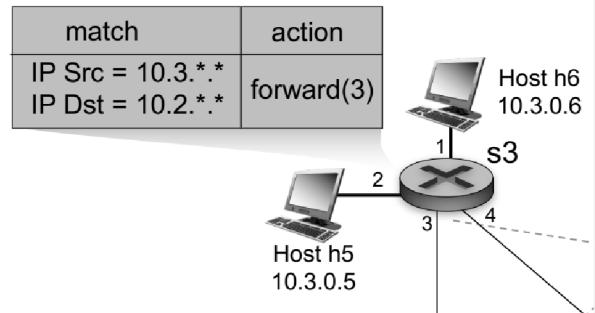


SDN (software defined network)

In un approccio SDN in cui i processi di routing sono controllati da un'unità centralizzata, quindi e' prevista la separazione netta tra il control plane ed il data plane, questi vengono gestiti da un software che scrive le tabelle di flusso locali (flow table) all'interno di ogni router. In altre parole il controllo del traffico viene trapiantato su uno o più



calcolatori che, mediante applicazioni specifiche, sono in grado di amministrare dinamicamente la rete come un'entità logica e virtuale. La porzione logica della rete si accentra quindi in un'unica entità dotata di una visuale panoramica di tutta la struttura, rendendo tutti i dispositivi di commutazione come switch o router semplice hardware di inoltro. Il router analizza degli header del pacchetto e in base a queste informazioni decide l'azione da intraprendere. **Openflow** è' una metodologia di definizione delle flow table basate su SDN. I flussi sono definiti dai campi dell'header. Le regole per il forwarding sono standard, il pattern corrisponde a far matching tra valori dell'header fields, actions: per i pacchetti matchati si può decidere se dropparli, inoltrarli, modificarli o essere mandati al controller centrale magari per notificare al server che c'è una nuova condizione da introdurre, priority per distinguere i pattern che hanno bisogno di gestioni diverse e i counters che sono informazioni sul numero di bytes e pacchetti scambiati. Le regole possono essere definite sul livello 2, 3 e 4. Quindi ci possono essere regole definite ad esempio a livello 2 che determinano le azioni da intraprendere quando arrivano pacchetti da un determinato MAC address, oppure regole di livello 3 simili alle classiche forwarding table, oppure regole per firewall che droppano tutti i pacchetti in arrivo da una sorgente o destinati ad una determinata porta. Quindi il paradigma match+action unifica differenti tipi di dispositivi: router tramite match con il longest destination IP prefix e le azioni di inoltrare i pacchetti in un output link, switch tramite il match tra MAC address di destinazione e azione di inoltro verso una porta o tutte le porte di uscita, firewall facendo match con IP address o TCP/UDP port e azione di concedere o negare e NAT facendo match tra IP e porta e azione di riscrivere un indirizzo e una porta. Rischi SDN, single point of failure e congestione.

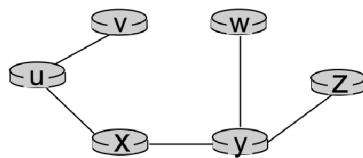


CAPITOLO V

Esistono due metodi per strutturale il control plane, uno tradizionale, in cui il routing è' calcolato localmente su ogni router, ed uno globale, in cui il routing viene controllato da un'unità centrale (SDN). Quest'ultima è' preferibile perché le decisioni prese da singoli router possono entrare in contrasto, mentre se è' gestito da un'unica unità centrale c'è una migliore coordinazione. La funzione dei protocolli di routing è' determinare il path migliore da un host sorgente ad un host destinazione attraverso una rete di routers. Un cammino buono può essere quello più veloce o meno costoso (dipende da diverse metriche). Una rete può essere vista come un grafo pesato, dove il peso può dipendere dal grado di congestione o da altri parametri. In un approccio centralizzato tutti i router posseggono informazioni su tutta la rete, in questo caso gli algoritmi in cui la conoscenza della rete è' globale sono detti **link state algorithms**. In un approccio decentralizzato i router conoscono la situazione della rete solo per lo stato dei loro router vicini, quindi il processo di calcolo può essere iterativo e prevede scambio di informazioni con i propri vicini, in questo caso si parla di **distance vector algorithms**. I cammini possono essere **statici** ovvero cambiare poco in funzione del tempo o **dinamici**, ovvero soggetti a continui aggiornamenti, perché ad esempio i pesi dei

link cambiano in continuazione. I link state algorithms sono usati quando ogni nodo conosce l'intera situazione della rete. L'algoritmo di Dijkstra può essere usato per link state. Quando l'algoritmo LS termina, abbiamo per ciascun nodo il suo predecessore lungo il percorso a costo minimo dal nodo origine. Per ciascun predecessore abbiamo il rispettivo predecessore, e in questo modo riusciamo a costruire l'intero percorso dall'origine a tutte le destinazioni.

resulting shortest-path tree from u:



resulting forwarding table in u:

destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Mentre LS usa informazioni globali, l'algoritmo distance vector è iterativo, asincrono e distribuito. È distribuito nel senso che ciascun nodo riceve parte dell'informazione da uno o più dei suoi vicini direttamente connessi a cui, dopo aver effettuato il calcolo, restituisce i risultati. È iterativo nel senso che questo processo si ripete fino a quando non avviene ulteriore scambio informativo tra vicini. L'algoritmo è asincrono nel senso che non richiede che tutti i nodi operino al passo con gli altri.

Per i distance vector si utilizza l'algoritmo di Bellman-Ford. Per decidere il prossimo nodo v da prendere nel cammino da x a y si prende il nodo v che minimizza la distanza da x a v e da v a y, dove v è un nodo vicino di x. L'idea è che di tanto in tanto ogni nodo vicino manda la propria distanza stimata ai vicini e quando x riceve la nuova distanza aggiorna le tabelle usando l'equazione di Bellman Ford. Ogni volta che un router cambia le distanze, lo notifica anche a tutti i nodi vicini causando l'effetto domino. Il problema di questo algoritmo è che raggiungere uno stato stazionario non è detto che arrivi mai. La velocità di convergenza dei link state è quadratica ma è preferibile a quella dei distance vector in quanto questi potrebbero entrare in routing loops. Nel caso di link state inoltre, anche se un nodo calcola un costo sbagliato, ogni nodo calcola solo la sua tabella, mentre in distance vector se un nodo sbaglia il calcolo l'intera rete può essere compromessa.

OSPF

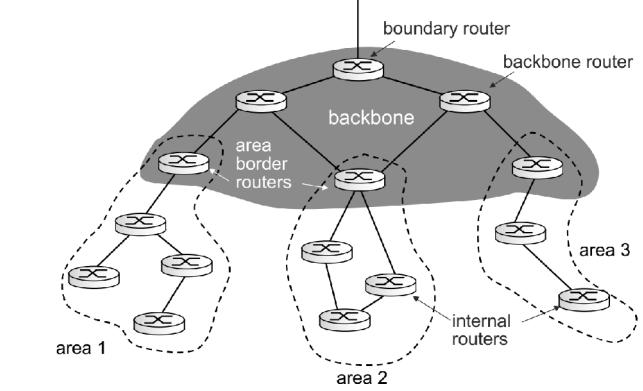
Nella realtà dei fatti vi sono distinzioni gerarchiche che rendono il calcolo degli algoritmi di routing più agevole. Queste distinzioni gerarchiche sono dette autonomous system o anche domini. Tutti i router nello stesso AS utilizzano lo stesso algoritmo di routing. Se ad un certo punto i pacchetti devono entrare in un altro AS devono essere gestiti da altri algoritmi.

Quindi gli algoritmi che gestiscono quest'eterogeneità sono detti **inter-AS routing**, mentre gli algoritmi all'interno dello stesso AS sono detti **intra-AS**. Nel caso di inter-AS le metriche del costo includono anche eventuali costi sul traffico di rete o accordi politici. **IGP** (interior gateway protocol) è una categoria di protocolli intra-AS. I più famosi sono **RIP** (routing information protocol) e **OSPF** (open shortest path first, che usa Dijkstra e i messaggi scambiati con OSPF sono autenticati e quindi sicuri), e **IGRP**. Anche all'interno dello stesso AS si possono suddividere aree e gerarchie.

In ogni area, uno o più router di confine d'area (area border router) si fa carico dell'instradamento dei pacchetti indirizzati all'esterno. Un'area di un sistema autonomo OSPF è configurata per essere l'area di dorsale (backbone area), il cui ruolo principale è quello di instradare il traffico tra le altre aree nel sistema autonomo. La dorsale contiene sempre tutti i router di confine del sistema autonomo, ma non necessariamente soltanto quelli.

L'instradamento nell'area di un sistema autonomo richiede che i pacchetti siano instradati dapprima verso il proprio router di confine (instradamento intra-area) e da questi, attraverso la dorsale, al router di confine dell'area di destinazione, che provvede a farli pervenire alla destinazione finale.

Hierarchical OSPF



BGP (border gateway protocol)

E' il protocollo utilizzato per connessioni inter-AS. Viene considerato il collante di internet. Il BGP fornisce ad ogni AS le informazioni per la raggiungibilità dagli AS vicini (**eBGP**, che viene implementato solo dai boundary router, connessione che riguarda router appartenenti a diversi AS) e propaga ad ogni router all'interno dell'AS le informazioni circa la raggiungibilità (**iBGP**, comunicazione che riguarda router nello stesso AS). I gateway router usano sia eBGP che iBGP. BGP include nei percorsi "buoni" non solo le metriche prestazionali ma anche le policy. Inoltre c'è un meccanismo che permette alle sottoreti di notificare al resto della rete la sua esistenza. I router BGP si scambiano messaggi (sono chiamati BGP peer) in cui pubblicizzano che sono i router gateway di un AS e se un nuovo router si aggiunge al suo AS direbbero anche che sono i router da contattare per inviare. Le informazioni vengono propagate all'interno dell'AS poi usando iBGP. BGP usa TCP per scambiarsi informazioni. I messaggi che utilizza sono OPEN per aprire una connessione, UPDATE per avvertire di nuovi percorsi, KEEPALIVE per mantenere connessioni aperte anche in assenza di UPDATE, NOTIFICATION per segnalare errori in precedenti messaggi e per chiudere le connessioni.

Se noi volessimo decidere il percorso che i dati devono compiere all'interno di un cammino, non possiamo semplicemente cambiare il valore dei costi dei collegamenti, ma dobbiamo usare degli algoritmi di routing come ad esempio OpenFlow così che noi possiamo decidere le regole. Quindi solo con SDN. Il problema è che questo paradigma è molto suscettibile ad errori. Quindi più trattabile ma anche più pericoloso. Traffic engineering significa riservare il

traffico per determinati dati, andando a modificare le tabelle di instradamento, ovvero programmare la rete.

CAPITOLO VI

Confidentiality significa che solo chi manda e chi riceve il messaggio possono comprenderne il contenuto. Il sender modifica il messaggio in un messaggio criptato, il receiver deve decriptarlo. **Authentication**, sender e receiver vogliono confermare l'identità dell'uno e dell'altro. **Message integrity**, sender e receiver vogliono assicurarsi che il messaggio venga ricevuto senza alterazioni durante il transito o successivamente senza accorgersene. **Access and availability**, in quanto noi vogliamo che i dati siano accessibili e disponibili immediatamente. Un intruso potrebbe intercettare i messaggi (**eavesdrop**), inserire attivamente messaggi nella connessione, potrebbe far finta di essere qualcun'altro (**impersonification**), potrebbe intercettare la connessione e dirottarla a se (**hijacking**), **denial of service** impedire di usare un servizio ad altri facendo un overloading di risorse.

Crittografia

Un messaggio in chiaro e' detto **plaintext**. Questo va in input all'algoritmo di cifratura che lo trasforma nel messaggio cifrato che viaggia in rete. La metodologia algoritmica e' nota, quindi non e' la segretezza dell'algoritmo che ci fa comunicare in modo sicuro ma piuttosto quella della chiave di cifratura. La chiave di decifratura e' quella che serve al receiver per trasformare il **ciphertext** (cifrato) in plaintext. Per rompere questo schema di cifratura, l'unico modo e' analizzare il messaggio cifrato (che chiunque puo vedere) ed applicare tutte le chiavi possibili usando l'algoritmo noto, o anche utilizzando modelli statistici. Un altro modo e' se l'intruso conosce una parte del messaggio in plaintext e quindi puo ridurre drasticamente lo spazio delle chiavi di ricerca. Una **chiave simmetrica** e' quando la stessa chiave viene applicata sia per cifrare che decifrare. Il problema e' come fare a fare in modo che sia sender che receiver utilizzino la stessa chiave, quindi riuscire a comunicare la chiave simmetrica in modo sicuro. Una chiave semplice di cifratura e' quella di mappare ogni lettera dell'alfabeto in un'altra (cifrario monoalfabetico), e quindi applicarlo alle lettere del messaggio. Se abbiamo M codici possiamo applicare ciclicamente questi codici con un determinato pattern. Ad esempio per crittare dog possiamo usare d dal pattern 1, o dal pattern 3 e g dal pattern 5 (cifrario polialfabetico). Nella sicurezza informatica si prende l'intera sequenza di bit e la si trasforma in un'altra sequenza alla quale poi applicare la chiave di cifratura. Uno di questi algoritmi e' detto **DES**, che genera chiavi a 56-bit simmetriche e viene applicato su messaggi da 64bit. Ogni blocco viene cifrato a catena rispetto ai precedenti. Oggi per decifrare DES basta meno di un giorno, anche se non si conoscono ancora backdoor di questo algoritmo (ovvero che inverte la funzione). Un'evoluzione più sicura e' chiamata 3DES che applica 3 volte di fila l'algoritmo DES. Lo schema a blocchi dell'algoritmo DES e':

- Prendiamo 64 bit del messaggio;
- Mettiamo i primi 32 bit a D1 e gli ultimi 32 bit a S1

- La parte D1 del registro viene copiata pari pari a sinistra del nuovo registro S2, mentre la parte sinistra precedente S1 viene presa e calcolata come parametro di input insieme alla parte destra D1 e 48 su 56 bit della chiave di cifratura e il risultato viene messo in R2
- Si itera questo procedimento per 16 volte e il risultato è il messaggio cifrato.

La funzione si autoinverte, quindi riapplicandola 16 volte in cascata decifra il messaggio..

Un algoritmo di cifratura più solido è **AES**. È un algoritmo di cifratura diverso e la chiave può variare in quanto dimensione, da 128, 192 o 256 bit. I blocchi da processare sono da 128 bit. Gli algoritmi con chiave simmetrica sono più efficienti e sprecano meno risorse.

Un'alternativa alla chiave simmetrica è la **chiave pubblica**. Sia mittente che destinatario hanno una chiave pubblica (che conoscono a tutti) ognuno ed una chiave privata ognuno. Alice manda il messaggio m e tramite l'algoritmo applica la chiave pubblica di bob al messaggio e lo spedisce. L'unico modo per indovinare m è provare tutti i messaggi in input conoscendo l'algoritmo e la chiave pubblica e vedere se abbiamo in output lo stesso ciphertext o rubare la chiave privata. La funzione di cifratura dev'essere non invertibile e si usano la fattorizzazione dei numeri primi e le curve ellittiche. Quando il messaggio arriva a bob lui usa la chiave privata e riottiene m . I prerequisiti sono che data la chiave pubblica $K+$ applicata al messaggio m , grazie alla chiave privata $K-$ è possibile ottenere m applicando $K-(K+(m))$. Tuttavia data $K+$ dev'essere difficile calcolare $K-$ a partire da essa.

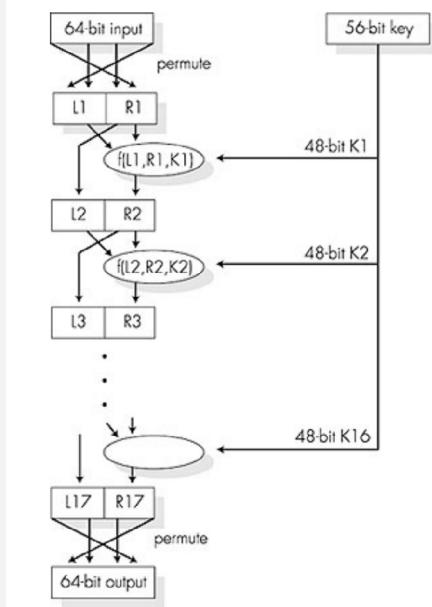
Sappiamo che $(a \% n)^{d \% n} = a^{d \% n}$.

Un messaggio è un pattern di bit, ed ogni pattern di bit può essere visto come un numero tramite conversione in binario. Crittare un messaggio è uguale a crittare un numero.

L'algoritmo RSA:

Prendiamo due numeri primi p, q rappresentabili in 1024 bit. $n=pq$. L'unico modo per scomporre n è trovare p e q . $z=(p-1)(q-1)$, che è pari perché p e q sono dispari. Scegliamo $e < n$ che non abbia divisori comuni a z . ovvero e, z sono primi tra loro (e dev'essere sicuramente dispari). Scegliamo d tale che $ed-1$ è divisibile per z , quindi $ed \% z = 1$. La chiave pubblica è $K+ = (n, e)$ e la chiave privata $K- = (n, d)$. Per cifrare $m (< n)$ si fa $c = m^e \% n$. Per decifrare $m = c^d \% n$. La cosa interessante è che vale la proprietà commutativa, ovvero $K-(K+(m)) = m = K+(K-(m))$, quindi il sender puo' certificare che m arriva da una certa persona se e solo se

applicando la sua chiave pubblica otteniamo il messaggio m . DES è 100 volte più veloce di RSA, quindi un metodo intelligente è scambiarsi chiavi simmetriche tramite chiavi private. RSA è sicuro perché anche se si



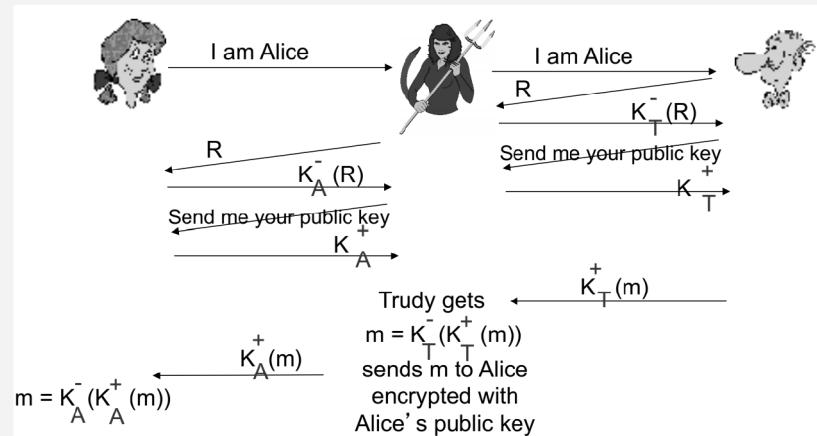
1. choose two large prime numbers p, q (e.g., 1024 bits each)
2. compute $n = pq$, $z = (p-1)(q-1)$
3. choose e (with $e < n$) that has no common factors with z (e, z are "relatively prime").
4. choose d such that $ed-1$ is exactly divisible by z . (in other words: $ed \bmod z = 1$).
5. public key is $\underbrace{(n,e)}_{K_B^+}$. private key is $\underbrace{(n,d)}_{K_B^-}$.

conosce la chiave pubblica (n, e), l'unico modo per trovare d e cercare i fattori di n senza conoscere p e q che è un problema difficile.

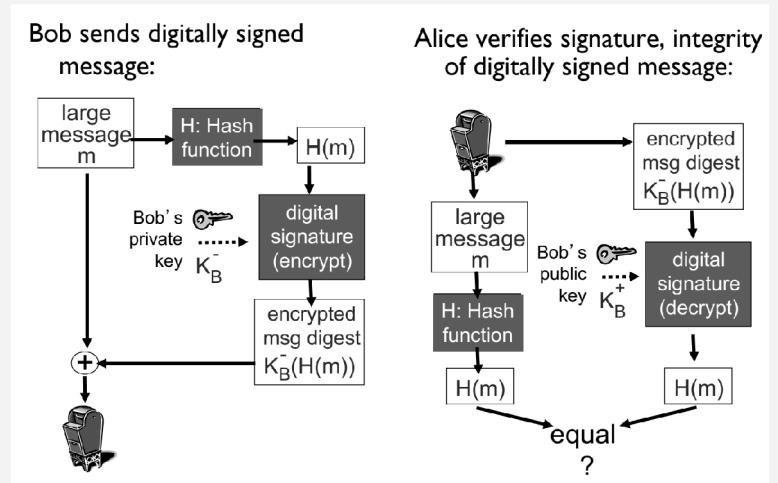
Authentication e message integrity

Il problema ora è garantire l'identità. Non basta dichiarare il proprio IP perché un intruso potrebbe usare un IP fittizio per fingersi qualcun altro. Anche usare una password non è abbastanza perché se qualcuno copia il messaggio lo può rimandare (**replay attack**). Anche cifrando il messaggio è inutile perché si può sempre copiare. Si utilizza quindi un **nonce**, un numero once-in-a-lifetime. Alice manda a Bob il messaggio e Bob per autenticare manda ad Alice un nonce R unico tale che Alice deve ritornare il messaggio crittato contenente R utilizzando una chiave condivisa che solo Alice conosce. Quindi per essere sicuri che la chiave simmetrica non venga rubata si usa la chiave privata. Alice manda un messaggio, Bob manda R e chiede ad Alice di cifrarlo con la sua chiave privata, in modo che solo la chiave pubblica di Alice riesca a ritornare il messaggio originale. Se non conosce la chiave pubblica può richiederla. Il problema è che se un intruso intercetta la richiesta potrebbe inviare la sua chiave privata che Bob assume sia di Alice, e quindi l'intruso potrebbe fingersi per Bob Alice e per Alice Bob e modificare a proprio piacimento i messaggi che si scambiano. Quindi questo schema può evitare un replay attack ma non un man in the middle. La debolezza di questo schema è quando si richiede la chiave pubblica ma si vede rispondere a sua insaputa dall'intruso. L'unico modo per certificare la chiave pubblica è averla dal **certification authority**. Queste sono presenti nella cache del browser ma hanno una scadenza, in modo che rinnovando i certificati un possibile intruso avrebbe la vita troppo difficile.

Per garantire l'integrità del messaggio è simile alla firma a mano, così da non rendere discutibile l'autore del messaggio. Per ottenerlo basta cifrarlo con la propria chiave privata. Si manda m sia in chiaro sia crittato, e se i due messaggi si equivalgono, ovvero $m = K^+(K^-(m))$, allora Bob è l'effettivo autore. Si manda anche il messaggio in chiaro perché se si manda solo il messaggio crittato, potrebbero comunque essere stati cambiati bit a caso e non avremmo nulla con cui confrontarli. L'intruso potrebbe ricevere $K^-(m)$ e modificarlo a caso, poi usare K^+ ed ottenere m' , così da sostituirlo anche ad m e mandare ad Alice questi due messaggi. Però appunto m' contiene valori casuali. Se m è molto grande $K^-(m)$ richiede molto lavoro. Per evitare ciò creiamo un **digests**, ovvero un valore $H(m)$ dove H è una funzione di hash che prende in input un m grande e da' in output un valore di dimensione costante. Se anche cambiasse un bit tra m ed m' , $H(m)$ e $H(m')$ saranno completamente diversi, quindi H genera valori "random". Anche le funzioni di hashing sono difficilmente invertibili. Gli m che generano collisione sono inoltre diversissimi tra di loro. Il checksum soddisfa la proprietà commutativa, quindi da un m possiamo facilmente ricavare gli m' con lo

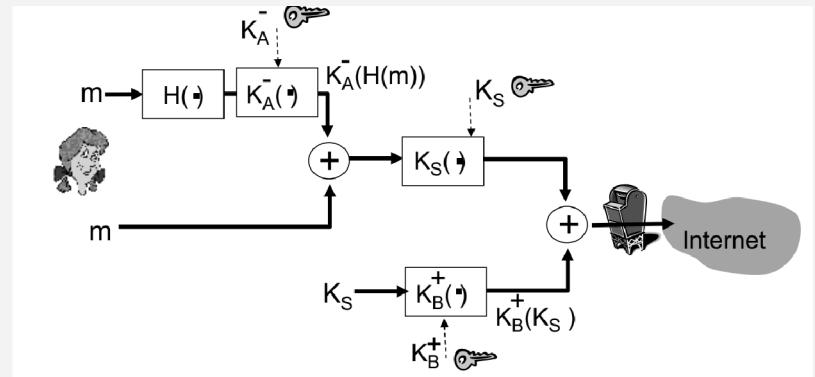


stesso valore di checksum. Quindi adesso quando spediamo il messaggio m , oltre al plaintext applichiamo $H(m)$ e poi $K-(H(m))$ per certificare che siamo noi gli autori del messaggio e quindi usiamo l'algoritmo di RSA solo ad una piccola parte rispetto ad m . Quando alice riceve il messaggio per certificare il messaggio applica $H(m)$ ad m ed estrae $H(m)$ con $K+$. Se le due funzioni hash si equivalgono allora è certificato. Alcune funzioni di hashing sono MD5 che genera un valore di 128 bit in 4 step. SHA-1 genera message digest da 160 bit. Per registrare una chiave pubblica in una certification authority bisogna innanzitutto provare la propria identità alla CA. Dopodiché la CA crea un certificato in cui lega l'identità che ne fa richiesta alla sua chiave pubblica firmandola con la chiave privata della CA. Chi riceve il certificato per garantire che non viene fregato applica $Kca+$ che richiede ad un'altra CA.



Secure e-mail

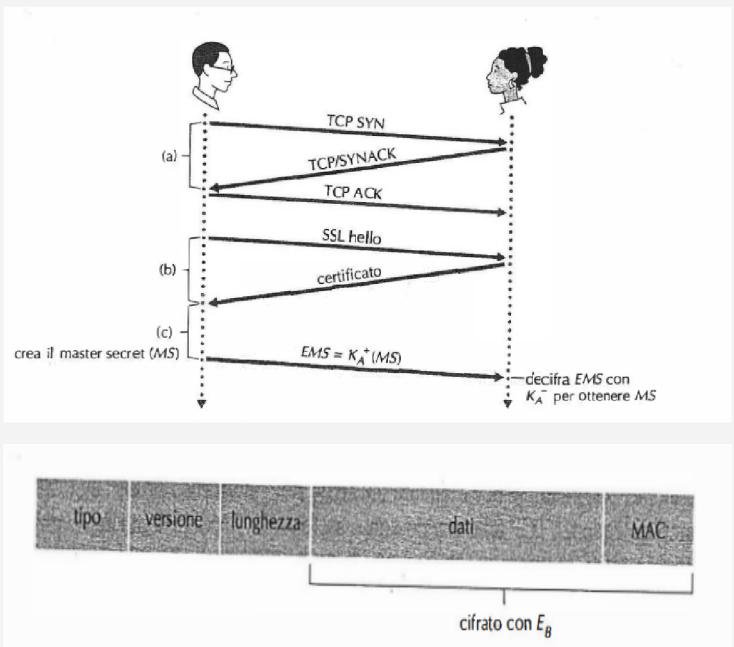
Alice vuole mandare una mail confidenziale a bob. Se m è molto grande si usano chiavi simmetriche e genera $Ks(m)$ e spedisce questo messaggio. Quindi si passa anche Ks utilizzando la $K+$ di bob ottenuta da una CA e si manda $K+(Ks)$ che quindi può essere estratta da bob e applicata per rigenerare m originale. A questo punto può continuare ad usare Ks nella stessa connessione TCP. Se alice volesse fornire l'autenticazione e l'integrità del messaggio, prende m e ne calcola il digest e viene firmato con la chiave privata di alice $K-(H(m))$ e lo invia insieme al plaintext. A questo punto bob chiede $K+$ ad una CA, confronta i due $H(m)$ e quindi se sono uguali m è autentico ed integro. Se volessimo comporre le due cose creiamo il digest e prendiamo anche m . Poi questi due messaggi vengono spediti in sequenza tramite Ks e quest'ultima viene spedita con $K+(Ks)$. Questo è detto schema **PGP** (pretty good privacy).



Secure TCP

SSL e' un protocollo largamente utilizzato, supportato da praticamente tutti i browser e utilizzato da http nella sua versione https. Una variazione si chiama TLS. SSL fornisce confidenzialità, integrità e autenticazione. Bob inizia una connessione TCP con alice. Bob manda un messaggio "SSL" al cui alice risponde con il proprio certificato contenente la chiave pubblica. Bob usa la chiave pubblica di alice per generare un **Master Secret** (MS) che poi bob critta con la chiave pubblica di alice (encrypted master secret EMS). Alice riceve EMS e ricava con la sua chiave privata MS. A questo punto bob ed alice possono scambiare messaggi sicuri generando chiavi ottenute grazie al segreto condiviso. La

connessione viene chiusa tramite speciali messaggi (autenticati) che assicurano che non sia un intruso a voler chiudere la connessione (**truncation attack**), e a questo punto le chiavi vengono distrutte. Anche in questo caso per evitare reply attack si usano nonce per ogni messaggio di andata e ritorno. Gli elementi di cifratura utilizzati da SSL sono gli algoritmi di chiave pubblica, di chiave simmetrica e i MAC algorithm (message authentication code). Un tipico messaggio SSL e' composto da 1 byte di content type che specifica il contenuto, 2 bytes per la versione di SSL utilizzata e 3 bytes per la lunghezza del campo dati e poi i dati (crittati) ed infine il MAC che e' l'equivalente dell'hash function. SSL usa RSA per chiavi pubbliche, AES per simmetriche e per hash MD5.



IPsec

Introduce un livello di sicurezza a livello 3. L'uso predominante di IPsec è la creazione di VPN (virtual private network). Le **VPN** sono reti private virtuali alle quali possiamo accedere da remoto come se fossimo fisicamente connessi in quella rete, per fare questo dobbiamo spedire i pacchetti a livello IP in modo che sia impossibile capire da dove stiamo comunicando. I pacchetti mandati su internet hanno un **IPsec header** oltre al normale header IP che leggono tutti i router su internet, e i router vedono solo le reti di destinazione e mittente. Mentre l'header IPsec crittato che contiene le informazioni su chi e' il reale host destinazione può essere analizzato solo dal router certificati che implemente il protocollo IPsec. Quando il pacchetto IPsec arriva al router dell'azienda viaggia all'interno della rete aziendale come se fosse stato generato da un host interno.

Firewall

Un firewall server viene utilizzato per isolare la rete interna di un'azienda dal resto di internet e filtrare i pacchetti che passano attraverso. Serve inoltre per prevenire un Dos o modifiche a dati della rete interna. I firewall possono essere suddivisi in tre categorie: **stateless** o **stateful** (che considerano il contesto) e **application gateways**. Un firewall stateless è un classico firewall con funzioni di filtraggio dei pacchetti. Viene installato a livello del router o vicino ad esso. Esso può decidere se inoltrare i pacchetti o dropparli basandosi su IP address e IP destination, TCP/UDP source e destination port number, richieste ICMP e richieste TCP SYN e ACK (per DoS). La tabella delle regole viene chiamata **Access Control Line** ed è simile ad OpenFlow.

Lo stateful packet filter traccia lo stato di ogni connessione TCP, quindi mantiene più informazioni. Le application gateways permettono di bypassare il controllo del packet filter, ad esempio se volessimo usare un telnet verso l'esterno ma il firewall blocca le connessioni telnet, possiamo connetterci all'application gateway tramite SSH e fare un telnet da lì che sarà fatto passare dal packet filter. Per ogni applicazione serve quindi un application gateway. La **DMZ** è un'area che dà servizi a chi fa richiesta (Web server, FTP server ecc.). Il firewall deve filtrare i pacchetti che arrivano alla DMZ facendoli passare e impedire il passaggio dei pacchetti verso la rete interna aziendale.

CAPITOLO VII

Le frequenze radio sono onde elettromagnetiche generate da correnti alternate all'interno di un circuito. Le antenne convertono la corrente elettrica in segnali a radiofrequenza e viceversa.

- L'**ampiezza** è il rapporto tra il picco positivo e negativo nell'energia del segnale rappresentata sotto forma di sinusoide. Tanto più la sinusoide è ampia tanto più il segnale è ampio. L'ampiezza della sinusoide corrisponde alla quantità di potenza energetica della trasmissione che si misura in watt.
- La **frequenza** è il numero di cicli che vengono effettuati in ogni unità di tempo. La **lunghezza dell'onda radio** è velocità della luce/frequenza (in Hz). Le antenne migliori hanno lunghezza di 1, ½, ¼ della lunghezza d'onda.

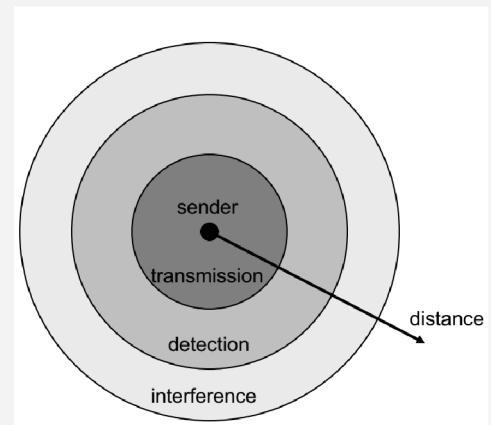
La potenza del segnale decade con l'aumentare della distanza. La **signal detection limit** è la sensibilità del ricevente di interpretare le onde radio. Se l'energia scende oltre il limite non si possono più capire i bit. Per raddoppiare la distanza di un fattore 2 bisogna almeno quadruplicare la potenza trasmissiva. Transmission range è dove la comunicazione è possibile e tutti ricevono dati con basso error rate. Detection range è dove si capisce che

Policy	Firewall Setting
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for institution's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets - except DNS and router broadcasts.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP packets going to a "broadcast" address (e.g. 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

qualcuno sta trasmettendo ma l'energia e' troppo piccola. Il range di interferenza e' dove esistono segnali ma sono incredibilmente deboli e potrebbero non essere rivelati.

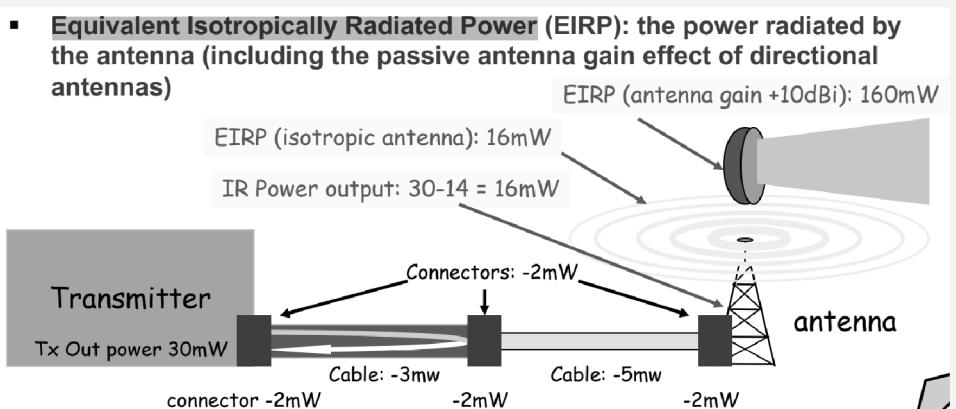
Oltre alla potenza del sender il raggio dipende anche dalla sensibilità del ricevente. Il MAC protocol si trova in queste 3 situazioni. Inoltre le onde radio ad alte frequenze sono buone per corte distanze ma subiscono la resistenza degli ostacoli, mentre le basse frequenze si comportano meglio su lunghe distanze e subiscono meno la resistenza degli ostacoli.

- La **fase** e' lo spostamento sull'asse del tempo dell'onda radio rispetto ad un segnale di riferimento, e che si misura in gradi o radianti. Una fase positiva significa che l'onda e' in anticipo (left-shift) una fase negativa significa che l'onda e' in ritardo (right-shift). Più vado lontano dall'origine più il segnale originale subisce uno sfasamento. Il problema e' quando il segnale arriva sia direttamente che attraverso un ostacolo, in questo caso il segnale risulta la somma vettoriale dei due. Questo puo' produrre sia aumenti dell'ampiezza dia diminuzione (ovvero se le onde arrivano in opposizione di fase).



Il campo magnetico ed elettrico si propagano su piani ortogonali tra loro, quello elettrico e' parallelo all'antenna. La **polarizzazione** indica l'inclinazione dell'antenna, si parla di horizontal e vertical polarization nel caso in cui il campo elettrico sia parallelo o perpendicolare al terreno. Quando le due antenne (sender e receiver) sono polarizzate allo stesso modo allora si cattura il maggior segnale possibile. Questo e' poco sentito in luoghi chiusi perché le onde rimbalzano ovunque e quindi la polarizzazione di auto-risolve. Gli amplificatori radio prendono in input segnali radio e ne cambiano solo l'ampiezza aumentandola grazie ad una sorgente di energia (guadagno attivo) o prendendola dall'ambiente (guadagno passivo). Le perdite sono intenzionali (cause da resistenze che trasformando l'energia in calore) o da ostacoli (muri, acqua e distanza). L'onda può subire **shadowing**, che impedisce che l'onda prosegua, **reflection** se rimbalza su una superficie, **refraction** quando cambia direzione, **scattering** quando rimbalza in maniera casuale colpendo un angolo spigoloso e **diffraction** quando colpisce bordi a curvatura molto ampia. Quando creiamo un sistema di comunicazione con il trasmettitore che manda un segnale grazie a dei connettori fino ad un'antenna, ogni volta che colleghiamo due dispositivi dobbiamo far attenzione al valore di **impedenza**, perché se hanno impedanze diverse si verifica il **VSWR** (Voltage Standing Wave Ratio) che fa dissipare l'energia. L'impedenza idealmente deve avere sempre un rapporto 1:1. Tutto l'insieme di dispositivi dal trasmettitore fino all'antenna esclusa e' detto **intentional radiator**. L'energia a cui facciamo riferimento e' quella che consegniamo all'antenna che e' l'energia del trasmettitore meno tutte le perdite nel percorso ed e' chiamata **intentional radiator power output**.

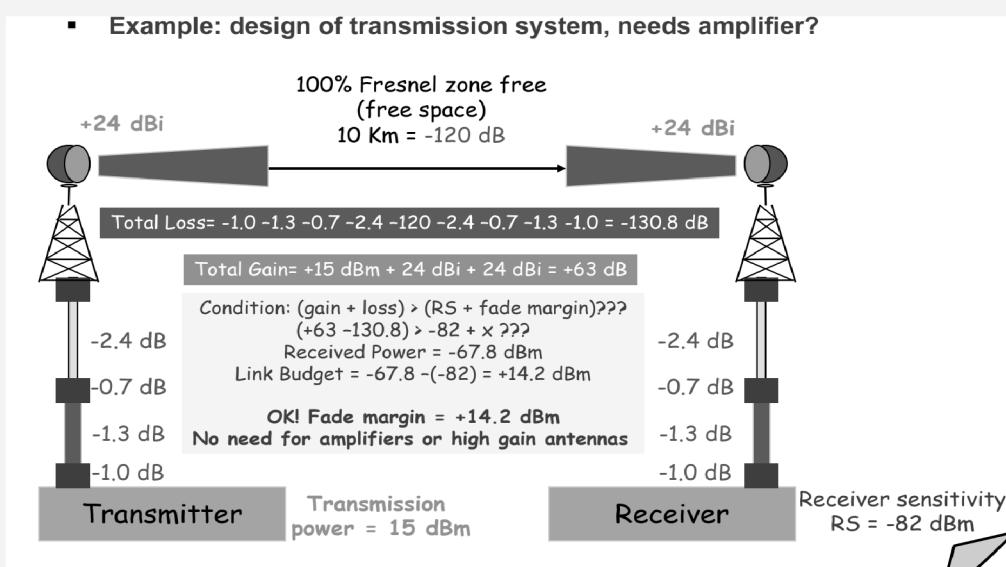
Un **radiatore isotropico** e' un'antenna che emette



l'energia omogeneamente in tutte le direzioni. Il valore **EIRP** (Equivalent Isotropically Radiated Power) e' l'energia che dovrei dare ad un radiatore isotropico per produrre lo stesso segnale di un'antenna direzionale.

Il **decibel** e' un'unità di misura per esprimere le perdite di potenza. Il decibel misura le differenze di potenza tra due segnali in scala logaritmica. 1 bel = 10 decibel. Ad esempio il rapporto tra 1 e 1000 e' 1:1000, ma $1=10^0$ e $1000=10^3$, quindi ci sono 3 bel di differenza ovvero 30 dB. Quindi nella scala esponenziale cresciamo ogni volta di un valore 10, mentre nella scala logaritmica aggiungiamo sempre 1. Per indicare la forza di un segnale in dB bisogna avere un segnale di riferimento. 10dB significa moltiplicare per 10, 20db significa moltiplicare per 100, 30dB per 1000 ecc. Se il valore in dB e' positivo abbiamo un guadagno altrimenti una perdita di potenza. Per misurare la differenza di potenza in dB tra il segnale trasmesso Tx e ricevuto Rx si utilizza la formula $10 \cdot \log(Rx/Tx)$. Sommare 10dB equivale a moltiplicare per 10, -10 equivale a dividere per 10, sommare 3 dB equivale a moltiplicare per 2 e sottrarre per 3 equivale a dividere per 2. Il **dBm** indica una misura assoluta di energia. $1\text{mW}(\text{milliWatt}) = 0 \text{ dBm}$. $P \text{ in dbm} = 10 \cdot \log(P \text{ in mW})$ e

$P \text{ in mW} = 10^{(P \text{ in dBm} / 10)}$. Un radiatore isotropico e' utopico. L'antenna che gli assomiglia di più e' il **dipolo**, che spara onda elettromagnetica forte in orizzontale ma quasi nulla in verticale. Quelli a basso guadagno producono più energia in verticale rispetto a quelli ad alto guadagno. La direzione nella quale un'antenna spara il massimo dell'energia si chiama direzione preferenziale. Il **dBi** misura la variazione di energia di un dipolo rispetto ad un radiatore isotropico. Le antenne omnidirezionali sono quelle che si avvicinano di più ad un radiatore isotropico, ovvero un dipolo. Antenna tilt significa inclinare la direzione dell'antenna. Le antenne semidirezionali concentrano l'energia verso una determinata direzione a cono. Alla base del cono, se prendiamo le estremità misuriamo una variazione di -3dB rispetto all'energia al centro, ovvero la metà. Un'antenna altamente direzionale concentra tutta l'energia lungo un cono con un'ampiezza molto piccola. La **fresnel zone** e' un'area centrale tra due antenne che comunicano che va lasciata libera da ostruzioni perché vi passano più segnali di tipo additivo sul ricevente. Il raggio della fresnel zone non dipende dal tipo di antenna ma solo dalla distanza e dalla frequenza, quindi e' lo stesso sia per dipoli che per parabole. Un antenna a settore e' un tipo di antenna che spara segnali in un settore circolare per coprire più spazio possibile. La perdita di segnale in funzione della distanza si misura in decibel con la formula $dB = 36.6 + 20 \cdot \log(Frequenza \text{ in Mhz}) + 20 \cdot \log(Distanza \text{ in miglia})$. Nella realtà, quando noi raddoppiamo la distanza il segnale decade di circa 1/4. Il **link budget** e' l'eccesso di segnale tra trasmittente e ricevente, ovvero la distanza tra l'energia in un certo punto e la soglia di sensibilità. Se il link budget e' positivo significa che arriva segnale al ricevente. Il delta di energia in eccesso (link budget) dev'essere superiore ad un **system operating margin** che e' tra 10 e 20 dB.

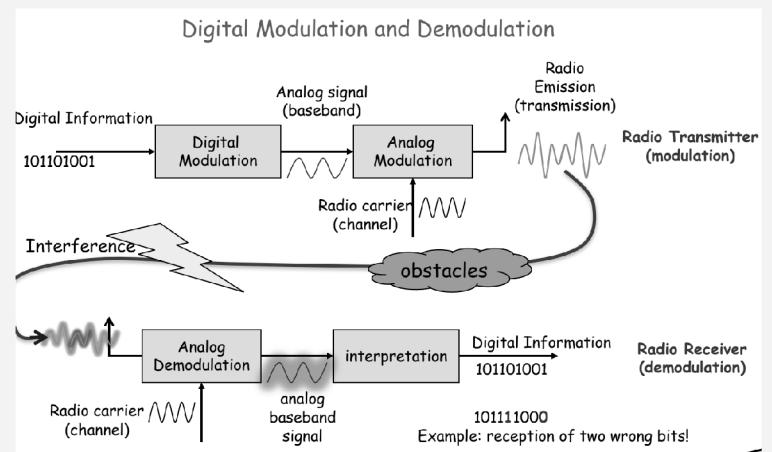
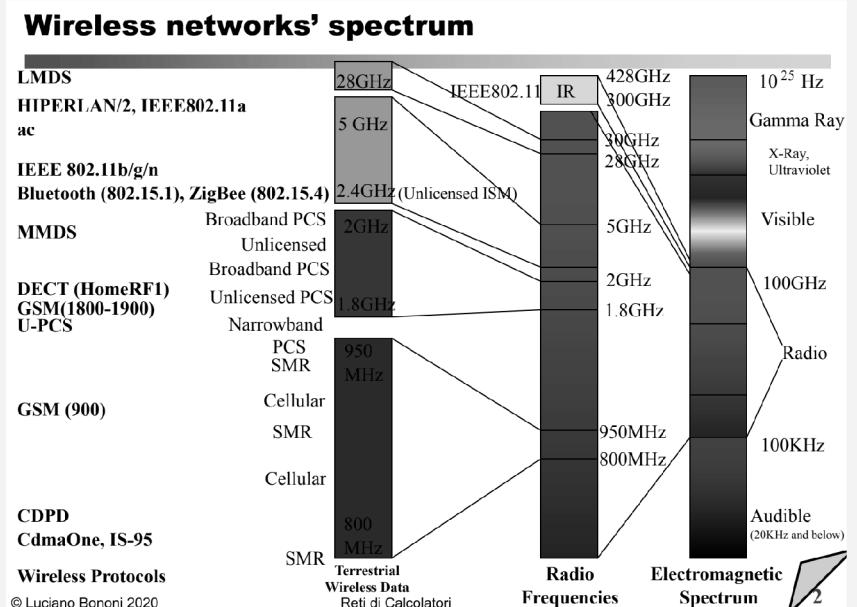


La larghezza di banda di un canale wireless dipende da quanto tempo e' necessario per rappresentare un bit. Un link unidirezionale e' quando da A arriva segnale a B ma da B non arriva segnale ad A. Un link bidirezionale simmetrico e' quando A e B comunicano con la stessa velocità l'uno con l'altro. Una tecnologia **narrowband** (banda stretta), usa una frequenza a banda stretta e si sfruttano dei filtri che permettono di isolare le energie che arrivano sulle frequenze radio ed estrarne solo il segnale da quella

determinata frequenza. **Frequency hopping spread spectrum** si differenzia dalla precedente perché un canale non usa sempre la stessa frequenza, ma effettua salti in maniera pseudocasuale (spesso con un pattern generato da una funzione hash). **Direct sequence spread spectrum** rappresenta ogni bit con un pattern di chip, ovvero valori binari. Quindi se 1 equivale ad un chip, 0 e' lo stesso chip capovolto. Si usa questa tecnica perché ogni chip viene trasmesso su tutto lo spettro della banda a disposizione. Le interferenze che si creano si annullano grazie alle differenze di pattern. Una rete ad hoc e' una rete senza un'infrastruttura predisposta, come quella bluetooth. Queste si contrappongono a quelle con infrastruttura. Un access point deve implementare sia stack per connessioni con filo sia per connessioni wireless, e permette di far comunicare questi due mondi tramite le bridging function. Per creare più canali sulle stesse risorse possiamo suddividere in base allo spazio, al tempo, alle frequenze o al codice.

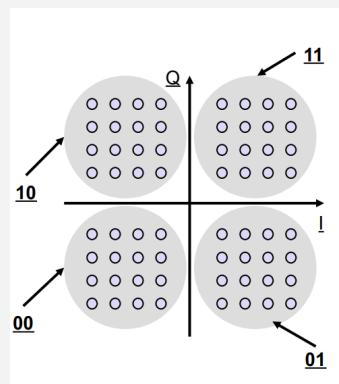
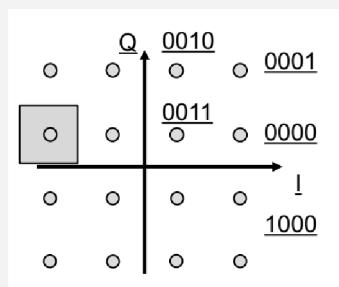
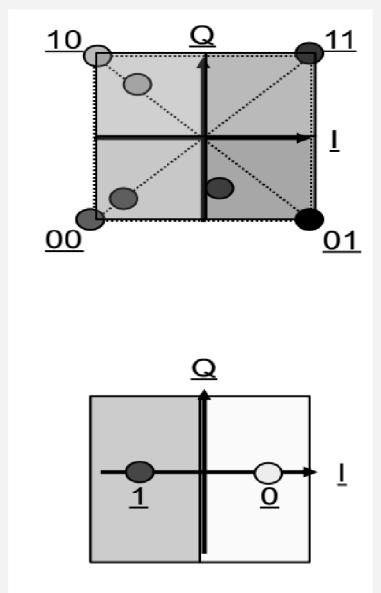
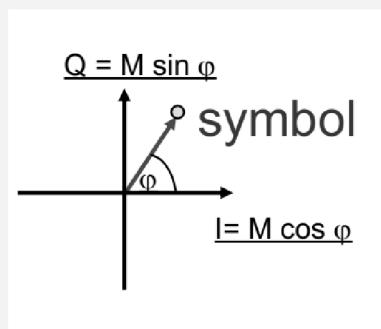
- **Frequency multiplex** significa che ogni canale utilizza un range di frequenze.
- **Time multiplex** si utilizzano tutte le frequenze ma per un periodo limitato di tempo. Time e frequency multiplex e' una fusione delle due tecniche, si fanno salti di frequenza nel tempo.
- **Code multiplex** tutti utilizzano lo stesso range di frequenza nello stesso tempo ma con diversi codici di chipping sequence.
- **Frequency planning** si riusano le stesse frequenze ad una certa distanza tale da non creare interferenza.

Usando le onde radio vogliamo riuscire a codificare dei valori digitali. Il sistema vede un calcolatore che vuole trasmettere un pacchetto di bit, il modulatore digitale cerca di rappresentarli con una sinusode e poi il modulatore analogico ne copia le caratteristiche per trasmettere sul canale che abbiamo scelto (decide la frequenze). Il segnale depotenziato che arriva al ricevente viene passato da un demodulatore analogico che ha come riferimento il canale radio in questione e a questo punto la sinusode che arriva,



anche se “sporca” è simile a quella di partenza e viene interpretata dall’interprete che la trasforma in bit. Un modo per trasmettere i bit potrebbe essere trasmettere 1 quando la sinusoide oscilla, trasmettere 0 come sinusoide piatta (tecnica **ASK**, amplitude shift keying). Questa tecnica è soggetta ad interferenza ma utilizza solo una frequenza di canale. Un’alternativa è variare la frequenza, frequenza alta significa bit 1 frequenza bassa significa bit 0 (**FSK**, frequency shift keying). L’ultima è utilizzare la stessa frequenza e ampiezza ma variando la fase, in opposizione di fase (**PSK**, phase shift keying).

Un **simbolo** è uno dei possibili stati in cui si trova l’onda, usato per rappresentare il segnale. Un simbolo ci dice l’ampiezza (distanza dall’origine degli assi) e fase (angolo con l’asse x). La frequenza dipende dall’input del modulatore. **BPSK** (binary phase shift (keying), ogni simbolo rappresenta un bit, fase 0 significa 0, fase 180 significa 1. **QPSK** (Quadrature phase shift keying) ogni simbolo rappresenta due bit. Avendo 4 simboli riusciamo ad etichettare ogni simbolo con due bit, quindi con un segnale riceviamo 2 bit. I simboli vengono interpretati associandoli allo stato più simile rispetto a quando arrivano. Quando l’errore del canale di trasmissione diventa abbastanza grande da far confondere un segnale con un altro stato allora si genera l’errore. Con una binary abbiamo una tolleranza all’errore del doppio rispetto a quella della quadrature. Quindi se le condizioni del canale lo permettono si usa la quadrature, altrimenti la binary. Di solito si comincia con la binary, se funziona bene si passa alla quadrature. Non tutte le codifiche sono identiche, infatti l’errore viene commesso con i simboli adiacenti, quindi i simboli adiacenti devono avere un bit di differenza (minima distanza di hamming), così da poter rilevare e correggere l’errore con un bit di parità’. Il bit di parità viene aggiunto in modo da rendere il numero di 1 pari, quindi se il receiver ha un numero di 1 dispari se ne accorge. Una matrice di parità mette alla fine di ogni riga e ogni colonna un bit di parità così se un bit cambia riusciamo non solo ad identificarlo ma anche a correggerlo. **Quadrature amplitude modulation** ha 16 simboli ognuno codifica 4 bit in cui ogni stato cambia sia fase che ampiezza. **Hierarchical modulation** ha 4 “nuvole di bit” ognuna contenente 16 simboli e ogni nuvola ha associati 2 bit, quindi ogni simbolo rappresenta 6 bit. Questa struttura dei simboli a gerarchia permette di differenziare due flussi di dati, ad esempio in una video call, in cui vengono trasmessi più dati video che audio, possiamo associare 2 bit di audio a 4 bit di video. In questo modo se non c’è interferenza trasmitteremo ad alta velocità e in modo corretto, mentre se c’è rumore, difficilmente un errore sarà tale da far confondere una nuvola con un’altra, quindi i dati video potrebbero arrivare sbagliati ma la voce sarà trasmessa correttamente.

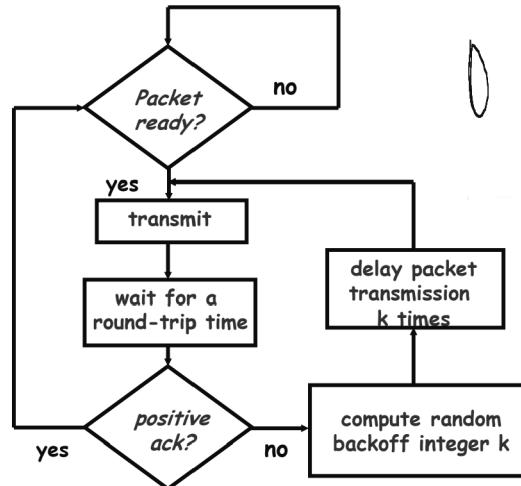


Le reti wireless sono broadcast naturali, il problema delle collisioni non è solo quando uno trasmette ma anche da dove sta trasmettendo. Inoltre nel mondo radio non si identifica una collisione. Le collisioni sono un effetto distruttivo sul ricevente. Il protocollo **Aloha** aspetta fino a che c'è un pacchetto pronto. Quando arriva lo trasmette subito ed aspetta un RTT, se arriva l'ACK manda il prossimo pacchetto, altrimenti genera un k casuale e riprova la trasmissione dopo k unità di tempo. Con questo schema la vulnerabilità di un frame è due volte la sua dimensione. **Slotted aloha** introduce gli slot, ovvero il tempo viene suddiviso ed il frame può essere trasmesso solo all'inizio di un nuovo slot di tempo.

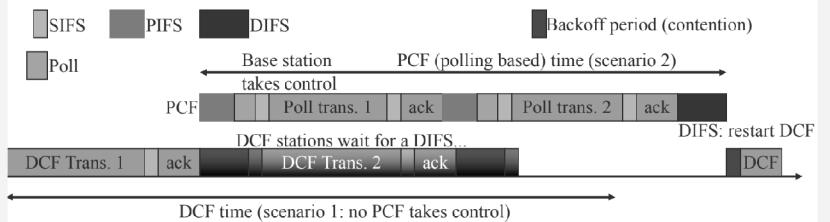
Quindi la vulnerabilità diventa 1 volta il frame size. **CSMA protocol** può ascoltare il canale grazie a dei chip e trasmette solo se nessuno sta trasmettendo. Se non arriva l'ACK aspettiamo dopo k unità. La vulnerabilità qui è due volte il propagation delay. Se usiamo **slotted CSMA** la vulnerabilità diventa una sola volta il propagation delay. Il problema però è che un ricevitore può avere segnali da più parti e quindi bisogna decidere non solo quando parlare ma anche chi deve parlare da dove. Chi trasmette manda un messaggio **RTS** (request to send), se il destinatario lo riceve manda un **CTS** che riceve sia chi trasmette che tutti gli altri, quindi chi è nel raggio del destinatario ora sa che non deve trasmettere perché qualcuno sta parlando. Questo viene fatto per proteggere i dati oltre una certa dimensione prestabilita. Lo standard **802.11** funziona con due schemi preesistenti, un **DCF** (distributed coordination function) basato su CSMA/CA (congestion avoidance) con binary exponential backoff, generalmente utilizzato per le reti ad hoc in quanto non necessita di una stazione di base, e **PCF** (point coordination function) basato su un controllore centralizzato, con un access point che decide chi parla.

Quando esiste un access point trasmette un pacchetto chiamato **Beacon** che contiene informazioni, che permette di schedulare chi parla. Quando finisce questa fase (PCF), fino al prossimo beacon si entra nella fase decentralizzata (DCF) in cui tutti provano a parlare. Se non esiste il beacon rimane solo la DCF. Dopo una trasmissione

DCF c'è un intervallo detto **SIFS** dopo il quale si da l'autorizzazione di mandare l'ACK. A questo punto trascorre un **PIFS** e l'access point se vuole può prendere il controllo del canale che può generare dei poll e chi viene chiamato può fare la trasmissione, poi si aspetta un



- Each station performs a carrier sensing activity when accessing the channel
- priority is determined by Interframe spaces (IFS):
 - Short IFS (SIFS) < Point IFS (PIFS) < Distributed IFS (DIFS)
 - after a SIFS only the polled station can transmit (or ack)
 - after a PIFS only the Base Station can transmit (and PCF takes control)
 - after a DIFS every station can transmit according to basic access CSMA/CA (DCF restarts)



altro PIFS e così via. Quando l'access point ha finito di chiamare le stazioni, l'intervallo di attesa è massimo ed è chiamato DIFS, dopo un **DIFS** chiunque può trasmettere. Se invece non c'è la gestione di un access point, dopo il tempo di DIFS ognuno genera un tempo casuale di attesa e si attendono tanti slot quando il valore generato prima di trasmettere (**backoff**). Il tempo di backoff è adattivo, quindi se non riceviamo un ACK si aumenta sempre di più. Se si arriva al 7 tentativi il MAC protocol si arrende e lo notifica al piano di sopra perché la comunicazione non è possibile. Non c'è collision detection.

- during the PCF time the base station has priority in accessing the channel
 - Base Station waits for a PIFS after a transmission and takes control (DCF stations must wait for DIFS>PIFS)
 - base station polls stations that reserved the channel
 - at the end of the PCF period the Base Station releases the channel and DCF restarts (after a DIFS)

