Introduzione ~ Corso di Architetture di Reti

Introduzione. Il corso è di tipo semantico (tutto teorico, tutto in aula). Riguarda i paradigmi ed i modelli per l'implementazione dei protocolli.
L'architettura di una rete di calcolo è basata su diversi modelli, nella sostanza nel momento corrente non possiamo scegliere un modello rispetto all'altro perchè di fatto c'è un unico standard imperante che è TCP/IP. Prima esistevano molti tipi di protocolli differenti, con il passare del tempo la diversità tra le varietà si è sempre più appiattita. Studieremo una architettura a strati partendo dal presupposto di gestirlo come modello.
Durante il corso i lucidi seguono le informazioni del libro di testo. Sono in inglese, per tutelare gli studenti stranieri e per sottolineare che la lingua dell'informatica non è l'italiano (i termini formali e tecnici sono da imparare in inglese, i libri non sono sempre tradotti).
L'esame consta di un test di ammissione scritto seguito da un orale. Il voto se lo si gioca all'orale, è evidente che anche lo scritto ha un peso, drasticamente inferiore se è perfetto e l'orale è disastroso. Uno scritto penoso ma seguito da un orale buono non pesa in maniera negativa. Lo scritto serve per evitare di venire all'orale senza essere consapevoli di non sapere nulla. Si fanno 2 parziali, uno a metà corso ed uno alla fine, se si superano entrambi equivale al totale, se si supera uno solo allo scritto intero si fa quello che si è perso. Lo scritto vale sempre, in eterno (nei limiti). Nello scritto si rispondono a domande di teoria, risposta aperta in poche righe riassuntive.
La prof ha una memoria di ferro, non prende presenze perchè si ricorda di tutti gli studenti, di cosa fanno e di cosa studiano. Le domande sono ammesse e non verranno usate contro di noi in tribunale. Anche stupide (per intenderci). I comportamenti stupidi però non se li dimentica. Non ci sono preferenze, per cui tutto è possibile.

Lezione 1 ~ Uno sguardo d'insieme

Cos'è Internet. La rete per noi coincide con quello che chiamiamo Internet, che però è solo il nome di un protocollo della rete. Nel linguaggio comune si indica con Internet la rete globale. Si parla di *host* o *end system*, e sono loro che costituiscono la rete (composti da hardware e software dedicati). Il *link* è una qualsiasi modalità di comunicazione, al di là del cavo fisico. I *router* sono dispositivi che fungono da centralini, hanno il compito di permettere la comunicazione tra le varie macchine. Per far funzionare tutto sono necessarie delle regole (o protocolli), sono regole rigide che permettono di distinguere e di concordare sul significato di suddette regole.
Internet è costituita da reti interconnesse tra di loro. Reti locali costituiscono reti geografiche, nella struttura esistono tutte le macchine che consentono la comunicazione tra host e computer. Questa infrastruttura ci permette di distribuire su varie macchine le applicazioni (dette *distribuite*).

- ✔ **Connection oriented.** Si fa in modo che l'applicazione funzioni solo quando le macchine siano in grado di essere consapevoli che tra queste macchine c'è una comunicazione in atto. Tale comunicazione resterà stabile per tutto il tempo della connessione in entrambi i lati.
- ✔ **Connectionless.** Comunico e se c'è qualcuno che ascolta bene, sennò pace.

Protocolli. Sono un concetto fondamentale. Un **protocollo** è un insieme di regole che permette alle entità in causa di comprendersi e di scambiarsi informazioni, che siano essere entità uguali o diverse. I protocolli decidono il formato e la dimensione dei messaggi, come comportarsi per riceverlo e per mandarlo poi indietro. Senza regole condivise non saremmo in grado di avere la Rete.

Network Edge.

- ✔ **Peer to peer.** È una comunicazione che riesce meglio alle ragazze che ai computer. Prevede che le comunicazioni passino da un pari ad un altro pari anche frazionate e spezzettate, allora non c'è una gerarchia tra i server. Ha bisogno di qualche aggiustamento per funzionare.
- ✔ **Client/Server.** Funziona in modo formale.
- ✔ **End Systems.** Sono tali tutti i nodi terminali della rete, chiamati anche host. Gli host sono a loro volta client oppure server e nulla vieta che possano essere prima l'uno e poi l'altro,

indistintamente a seconda delle circostanze.

Il cuore del network. La rete è fatta da una griglia che collega dei punti fermi, detti router. I router si collegano tra loro garantendo l'esistenza della rete. Hanno sempre più collegamenti tra loro, garantendo la ridondanza della rete e lo smistamento dei messaggi (detto *routing*).

- ✔ **Circuit switching.** Commutazione del circuito: per comunicare tra loro gli elementi della rete prima di mandare dati scelgono la strada ottimale, ed i dati poi verranno sempre mandati per quella strada fino alla fine della connessione. Se abbiamo tante connessioni i dati fanno sempre la stessa strada. L'hardware di base è quello dei centralini telefonici tradizionali. Se la connessione perdura non si perde nessun pacchetto, i pacchetti poi arrivano sempre in ordine. Le risorse degli end node sono riservate ad una data comunicazione (si parla di un circuito fisico e non virtuale).
- ✔ **Packet switching.** I dati sono mandati come per posta, posti in una busta con indirizzo. Ogni pacchetto prende tendenzialmente una strada diversa. Non c'è garanzia che arrivino in ordine, andrebbero numerati. I pacchetti si potrebbero anche perdere. Non c'è nessuna suddivisione di banda od allocazione di risorse limitate, il problema è che c'è concorrenza nell'uso delle risorse e la richiesta potrebbe essere superiore all'offerta. Se non ci sono regole si può incorrere in congestione.

Abbiamo a che fare con due situazioni diverse dalle quali vogliamo ottenere lo stesso risultato: i dati devono andare da A a B in maniera corretta. In molte circostanze non c'è un metodo migliore dell'altro, dipende strettamente dalle esigenze.

Accesso alla rete. Non è fisicamente possibile connettere tutti contemporaneamente, per cui è meglio fornire una rete un po' malconcia alla maggior parte degli utenti, invece che fornire il miglior servizio a pochi.
Come possiamo connette i sistemi alla infrastruttura di confine? Esiste una specie di spina dorsale della rete dove ci sono molti router che fanno girare i nostri pacchetti. Esistono anche gli host che stanno in periferia. In una rete locale serve una "cosa" che parla con gli host, prende i messaggi e dall'altra parte parla con l'infrastruttura. Due host finali di aree locali interagiscono tramite backbone. Gli *edge router* sono router locali. Esistono diversi tipi di accesso:

- ✔ **Accessi residenziali.** Zone non raggiunte da nessuna specifica tecnologia che utilizzano il modem telefonico, si dispone di una banda fino a 56 Kbps. **ISDN**: teleconferenze in banda dedicata. **ADSL**: il provider fornisce un sistema di comunicazione asimmetrico, perchè il link in upload (da casa verso il router) è considerato meno probabile rispetto a quello in download. Esistono anche accessi residenziali che utilizzano le TV via cavo (in Italia però non esistono).
- ✔ **Accessi istituzionali.** Dal punto di vista istituzionale, le istituzioni hanno delle LAN (reti locali che collegano macchine locali tra loro ed all'esterno tramite router). Ethernet: tecnologia che dedica link (in cavo di rame o fibra ottica) alla comunicazione; è uno standard che è saputo crescere con il tempo, possiamo far funzionare alle stesse regole messaggi piccolissimi o grandissimi, Ethernet è l'unica tecnologia che è riuscita a favorire la comunicazione hardware compatibilmente con la gestione e la comprensione del pacchetto. Anche in casa possiamo mettere una piccola LAN.
- ✔ **Accessi mobili.** Accesso alle reti di tipo wireless è condiviso. Non è possibile dedicare delle risorse di comunicazione quando si ha a che fare con delle onde che si disperdono nell'ambiente. C'è la necessità di tener conto della condivisione. La banda è fornita dall'ampiezza delle onde elettromagnetiche (quando va bene sono 108 Mb, in realtà sono le bande degli access point locali senza impedimenti fisici).

Bisogna tenere conto delle esigenze di banda, e vedere se vogliano condividere o dedicare.
Nelle *home networks* abbiamo modem e routers.

Link fisici. Trasmettono dati attraverso cavi. La loro portata determina la loro banda. È detta frequenza la velocità con cui si propagano i dati. I mezzi possono essere confinati nello spazio e sono detti guidati, confinare i dati vuol dire farli passare per un cavo. Un mezzo non guidato è per esempio il wireless. Nello spazio tridimensionale è necessario tenere conto anche delle locazioni degli access point wireless. Il mezzo fisico guidato più diffuso a livello locale è il **doppino**

telefonico (che ha poco a che fare con il doppino reale, in origine però erano fatti con due cavi arrotolati su se stessi ed uno attorno all'altro, anche per tentare di schermarli vicendevolmente). I cavi Ethernet sono:

- ✔ **CAT 3** non va preso per le reti ma solo per i telefoni. Era compatibile con l'Ethernet vecchia maniera che arrivava massimo ai 10 Mbps.
- ✔ **CAT 5** è ormai limitato.
- ✔ **CAT 7** ha una fisica diversa, ha un connettore diverso, lavora molto bene ma vanno cambiate tutte le schede di rete delle macchine. Un **CAT 6** invece lavora molto bene, è simile al CAT 5 possono reggere frequenze diverse. Il CAT 6 ha sezioni e schermature diverse e non serve cambiare le schede di rete.

I **cavi coassiali** fanno la storia dell'Ethernet, il coassiale ha un singolo cavo (il doppino ne ha due), è bidirezionale, ha un'anima di rame avvolta in strati di plastica. È stato il punto di partenza delle connessioni di tipo LAN. I **cavi in fibra ottica** sono fatti in fibra di vetro o plexiglas che utilizza fasci di luce per trasmettere dati. Raggiunge grandi velocità. È il mezzo ideale per combattere il rumore elettromagnetico, essendo la fibra immune a qualsiasi interferenza. Ha bisogno di schede speciali, ed è estremamente delicato, basta poco per spezzarlo od ammaccarlo rendendo la fibra opaca e benedire l'acquisto dispendioso con un paio di bestemmie secche. Le **onde radio** sono segnali portanti nello spettro elettromagnetico, sono bidirezionali, sono fondamentalmente dipendenti dagli ambienti in cui si trovano per cui sono interessate ad interferenze (può subire forti attenuazioni oppure distorsioni inaspettate). Le onde radio permettono una comodissima fruizione delle reti, se dobbiamo lavorare intensamente e serve affidabilità di servizio, non ha senso connetterlo con connessioni wireless. Il **satellite** è una ricetrasmittente di segnale. Utile per raggiungere aree difficili da cablare. Fornisce bande un po' migliori ma non tanto, è chiaro che si usa quando non si riesce a fare altrimenti. È molto dispendioso in termini di denaro.

Ritardi nelle reti packet switch. Dove possiamo perdere tempo durante la comunicazione? Non è facile stabilire il ritardo da un sistema all'altro, che dipendono dalla topologia e dagli hop (=salto da una macchina all'altra). Ogni hop è soggetto a 4 fonti di ritardo:

- ✔ **Nodal processing.** Controllo degli errori sul pacchetto, e si stabilisce a quale porta di uscita mandarlo.
- ✔ **Tempo di accodamento.** Tempo che il pacchetto aspetta per essere immesso nella rete, e dipende dalla congestione del router che stiamo aspettando. I tempi di accodamento non si possono migliorare perchè dipende dal traffico fatto da altri. Il tempo dipende dall'intensità del traffico, se ho un link con una certa banda R e pacchetti di dimensioni fisse e standard L, abbiamo A pacchetti che entrano nella rete, abbiamo una intensità del tipo (AxL)/R. Se il numero totale dei bit è piccolo rispetto alla banda non c'è problema. Se i bit aumentano la coda diventa ingestibile con massimo ritardo, ad un certo punto si perdono pacchetti ed il ritardo diventa infinito. Un pacchetto perso significa pacchetto ritrasmesso... aumento la congestione e si crea un circolo vizioso. L'unica possibilità è data dal tenere il numero di pacchetti in arrivo sotto la soglia.
- ✔ **Ritardo di trasmissione.** Tempo necessario a mettere nella rete tutti i bit del pacchetto. Dipende dalla dimensione della banda e dalla lunghezza del pacchetto. Una volta che i bit sono sul cavo, dal punto di vista fisico ci mettono del tempo ad arrivare dall'altra parte del cavo.
- ✔ **Ritardo di propagazione.** Dipende dalla velocità del segnale nel mezzo e dalla distanza da attraversare. È la velocità fisica. Si tratta del tempo che impiegano i dati per arrivare a destinazione.

Gli ultimi tempi sono tempi ineliminabili che non possono essere annullati, dipendono dalla dimensione del messaggio e dalle condizioni del mezzo.

Livelli di protocollo. Le reti sono complesse. I protocolli si costruiscono su livelli. Ogni livello implementa un servizio e compie delle azioni caratteristiche del suo compito interno. Si relazionano con gli altri livelli superiori ed inferiori. Il servizio diventa poi un servizio distribuito indipendentemente dal fatto che i protocolli siano separati o distribuiti tra il lato di partenza o di arrivo. I servizi di più basso livello sono quelli di infrastruttura, quelli più alti riguardano le applicazioni. Si divide in strati per semplificare il sistema. Ci permette di modificare l'applicazione

o di migliorare il servizio senza intervenire su tutti i livelli. La suddivisione in livelli ci permette di mantenere e migliorare il sistema senza complicare ogni singolo intervento.

Un modello a strati. È un modello vincente, tuttavia con il passare del tempo sono nate delle implementazioni concorrenti. Per cui si parla di modello di riferimento, ben fatto e tuttora utilizzato soprattutto per mappare e studiare su di esso le architetture sviluppate quasi indipendentemente. È diverso sviluppare un modello e costruirci sopra di tutto contro all'avere architetture che si vogliono forzare sopra un modello che non gli appartiene.
Il modello di rete dominante oggi è costituito da 5 strati, detti *layers*. Alcuni layer si confondo tra loro per via della similitudine dei loro servizi.

- ✔ **Livello fisico.** Definisce la versione hardware.
- ✔ **Livello data-link.** Stabiliste le regole del protocollo per trasferire dati da un host ad un altro direttamente connessi. Due macchine che sono collegate tra loro con un cavo o wireless sono collegate in modo diretto. Se le facciamo comunicare attraverso uno switch esse non sono più collegate direttamente. Livello fisico e data-link sono implementati entrambi sulla scheda di rete. Nello sviluppo di queste schede non si è mai fatta una divisione netta tra i due layer. Dal punto di vista dello studio del protocollo si parla sempre di due cose estremamente differenti.
- ✔ **Livello di rete.** Parte che si occupa dell'instradamento dei pacchetti (detti anche *datagram*) dall'host sorgente a quello di destinazione. Verrà scelta e mantenuta una strada (route) diagnosticando eventuali congestioni e ritardi. Nell'instradamento sono coinvolti anche tutti i livelli superiori.
- ✔ **Livello di trasporto.** Si occupa del trasferimento non dei pacchetti ma dei dati da un host che parla a quello che ascolta. Si occupa di tutta la parte importante del trasferimento dei dati. Si utilizzano due protocolli: TCP e UDP. Le loro caratteristiche sono diverse e più o meno affidabili a seconda delle necessità.
- ✔ **Livello applicativo.** Il primo che vede l'utente, gestisce tutte le regole ed i protocolli per far comunicare tra di loro le applicazioni presenti sulla rete (posta e web).

Lo stack protocollare del modello è diviso però in 7 strati, si aggiungono a quelli appena visti:

- ✔ **Livello presentazionale.** Dovrebbe permettere alle applicazioni di essere indifferenti dalle caratteristiche specifiche delle singole macchine. Lavorano con dati che vanno bene su qualsiasi macchina. È stato eliminato dagli sviluppatori perchè si riusciva ad integrarlo nel livello applicativo.
- ✔ **Livello di sessione.** Doveva preparare il lavoro al livello di trasporto, preparando sessioni e connessioni, recuperando dati e gestendo la sessione di comunicazione tra due unità. Sostanzialmente tali operazioni sono gestite dai livelli di trasporto.

Non sempre servono tali servizi ed hanno bisogno di una organizzazione formale.
Perchè ISO/OSI è basato su 7 layer quando il modello è di 5?? All'epoca del suo sviluppo il modello dominante era quella aziendale di IBM. L'architettura proprietaria di IBM era fatta su 7 layer. Faceva anche cosa diverse. Si è concordato che le regole da seguire fossero di IBM. Ma all'epoca ci son voluti anni per definire le regole del sistema e le reti dominanti erano dell'IBM, di Digital e di altre. TCP/IP l'hanno sviluppato le università ed era open, gracile ma aperto a tutti e migliorabile da chiunque. Poi si è smesso di avere macchine con architetture particolari (CISC) con sistemi operativi con processori sviluppati dalle aziende per compiere operazioni complesse in un solo ciclo di calcolo (RISC = adesso il processore è in grado di fare un numero ridotto di istruzioni semplici ad una frequenza molto elevata in modo da essere competitive). Le aziende puntavano solo sui loro processori CISC e si sono ritrovate in concorrenza con macchine più economiche e snelle fornendo performance non tanto differenti. Il sistema operativo che si adattava meglio per questi nuovi processori era UNIX, sviluppato dalle università e non proprietario (in parte ci ha lavorato anche SUN). Ci hanno messo un'architettura di rete... che non era quella delle grandi aziende, molto belle e molto migliori ma anche chiuse e costose. La Digital ha adattato il sistema operativo alle macchine RISC, ma quando ha adattato le reti non ha usato TCP/IP... ha usato ISO/OSI. È stata l'unica azienda a fare così e non è stata una buona scelta: richiede banda maggiore e router più potenti. Digital è fallita così. Talvolta i modelli sono teoricamente raffinati ma disastrosi da implementare.
Middelware: fa qualcosa di quello che suggeriva ISO/OSI ed anche qualche cosa si più, ma non è implementata su tutta l'architettura.

Comunicazione logica. Le entità son distribuite e ciascuna ha al suo interno un software di rete diviso nella sua parte di layer e protocolli. Gli end node devono avere tutta la pila protocollare, nel core invece non è necessaria l'intera implementazione ma solo i primi 3 layer. Non è detto che però non ci siano... diciamo che non è obbligatorio. La comunicazione logica avviene solo tra livelli omogenei (tra pari). Dal punto di vista fisico i layer comunicano in direzione verticale.

Protocolli e dati. Quando usiamo un protocollo ogni strato prende i dati dal livello superiore per passarli a quelli inferiori. I dati vengono presi in carico dal livello applicativo che compie sui dati le operazioni caratteristiche di quel protocollo, poi ci "attaccano" un **header** e passano tutto al livello successivo. I dati prendono nomi diversi man mano che scendono nello stack: **messaggio**, **segmento**, **datagram**, **frame**, poi son solo una *serie di bit in un cavo*. Il messaggio passato di mano in mano viene un po' alla volta incapsulato di header in header, che verranno letti e "spacchettati" una volta arrivati a destinazione.

Struttura di Internet. La nostra Rete è una rete di reti. In origine e dal punto di vista logico le reti sono ancora reti indipendenti, perchè sono gestite a livello locale anche se ogni rete magari vede tutte le altre reti. Ormai però ci sono talmente tante parti condivise (le istituzioni non comperano più i cavi e la topologia) che le nostre reti sono sì gerarchicamente divise e separabili ma tutte abbastanza connesse. Le possiamo pensare, nella loro singolarità, in maniera strettamente gerarchica:

- ✔ **Backbone.** Dorsali nazionali od internazionali che son gestite da provider istituzionali (gestori telefonici) a cui le istituzioni chiedono una quantità di servizio di banda per soddisfare le richieste della nazione. Sono interconnessi da punti di accesso di reti.
- ✔ **Reti regionali/istituzionali/nazionali.**
- ✔ **ISP.** Come Telecom e Fastweb, Vodafone. Sono quelli che permettono le suddivisioni delle bande radicalmente alle piccole aziende ed istituzioni.

Lezione 2 ~ Livello Applicativo

Livello Applicativo e di Trasporto. I modelli di servizio non sono *l'implementazione ma il modello su cui è stato studiato il servizio da implementare*. Di solito il modello è difficile da implementare e molte funzioni non sono necessarie, allora si opta per una implementazione più efficiente e che funziona. Nello specifico, vederemo HTTP, FTP, SMTP (permette lo scambio di messaggi di posta elettronica tra server di posta), POP (protocolli che gestiscono la posta tra utente finale e client di posta), DNS (protocollo di applicazione fondamentale per destinare i messaggi tramite la strada giusta, serve a mappare i nomi delle macchine conosciute in modo mnemonico sulla parte di rete usata per stabilire la strada finale del messaggio). Il DNS è usato sulla rete, il servizio a livello di applicazione serve a far coincidere nomi ed indirizzi numerici.
Le applicazioni in questo senso sono applicazioni di rete, che possono essere *distribuite* (ovvero processi che prevedono che il software giri su macchine remote l'una all'altra, se ci son messaggi da scambiare). Il formato del messaggio è la parte sostanziale, il collegamento logico tra le diverse applicazioni su macchine diverse.
I **protocolli** son parti fondamentali delle applicazioni, che determinano i formalismi dello scambio delle informazioni. I **sevizi** sono forniti in modo logico tra i livelli dell'applicazione.
Le applicazioni di rete coinvolgono processi e comunicazioni tra di essi. Il messaggio è la parte concreta, unità di cambio durante la comunicazione. La comunicazione tra processi normalmente implementata con memoria condivisa non è possibile nella rete, allora devono essere creati dei pacchetti sulla rete. Uno user agent è una interfaccia, un programma tra utente ed applicazione. Il protocollo è HTTP, ed il browser per esempio è uno user agent che parla con l'utente e codifica i comandi da mandare nella rete.
Come avviene la comunicazione?? Tra le parti in causa, viene utilizzato il paradigma **client-server**, che prevede che ci siano due parti in causa uno che recita il ruolo del client e l'altro del server. Il client è quello che fa la richiesta e parla per primo, chiede un servizio al server e spesso il client è privo di informazioni mentre il server gliele fornisce, anche se non sempre è così (ci son circostanze con informazioni bilanciati). Il server fornisce il servizio solo se è attivo. Dal punto di vista

architetturale non è necessario che siano macchine distinte, esistono solo ruoli che prevedono certi comportamenti. Il server deve essere sempre in ascolto, il client si attiva solo se necessario. Il server per essere sempre in ascolto deve avere un indirizzo IP permanente. Ora molte macchine hanno indirizzi nascosti, anche se permettono alla macchina nascosta di vedere il resto del mondo. I server devono essere sempre dichiarati. Inoltre, ci sono alcuni server di larghissimo utilizzo, hanno bisogno di "scalare" ovvero mantenere le proprie capacità di risposta attive anche quando i client sono tantissimi e molto sparpagliati. Il client può essere intermittente (può collegarsi e scollegarsi a piacere) e deve essere in grado di comunicare, il client può essere un nodo nascosto. In un meccanismo client-server standard un client non comunica mai direttamente con un altro client.
Le architetture **P2P pure** prevedono che la comunicazione avvenga tra due host, nonostante siano macchine client con caratteristiche da client. *Ma come è possibile??* Dal punto di vista del modello è una soluzione ottima, dal punto di vista dell'implementazione è difficilissima. Di solito usiamo architetture ibride con client-server: Skype e Istant Messenger. *Skype* si basa su un servizio centralizzato per trovare gli interlocutori, instaurando una connessione diretta dal punto di vista logico, anche se la ricerca degli interlocutori è fondamentalmente una comunicazione con server. Le architetture ibride sono abbastanza "scovabili".
I programmi hanno bisogno di comunicare ed i processi client e server idem.
API (*Application Programming Interface*): la socket è un API, definisce l'interfaccia tra applicazione (livello applicativo) e livello di trasporto. Il nome proprio a livello di architettura è appunto **socket**. Due processi comunicano tra di loro tramite una o più socket, mandando dati dal livello applicativo a quello di trasporto. Un processo identifica l'altro processo con cui comunicare tramite un *indirizzo IP* ed un *numero di porta* del processo esatto presente su quella data macchina. È più importante il lato server che riceve la nostra richiesta (il server è un software!!!) .
Quali sono le eventuali richieste che una applicazione può aver da fare al livello di trasporto??

- ✔ **Necessità di non perdere dati.** Alcune applicazioni non possono permetterselo, altre sì. Il trasferimento di un eseguibile non può perdere dati a rischio di compromettere l'integrità del file.
- ✔ **Necessità di aver dati affidabili e non affidabili.**
- ✔ **Avere una banda specifica.** CI sono applicazioni che hanno necessità di una banda minima e costante. Se i pacchetti arrivano troppo fitti o troppo radi non siamo capaci ci comprenderli. Altre applicazioni sono interessate che alla fine arrivi tutto e bene, non importa se in tempo reale.
- ✔ **Necessità di una buona tempistica.** Per quanto riguarda le applicazioni in tempo reale è necessario che i dati arrivino in un buon tempo di reazione (il tipico "lag" del videogioco online).

Esistono due protocolli che forniscono servizi a livello di trasmissione:

- ✔ **TCP.** Fornisce servizi affidabili perchè è orientato alla comunicazione (instaurando la connessione). R*eceiver* e *Sender* sono in comunicazione ed il protocollo si preoccupa di vedere che dati arrivino tutti ed in ordine; si controlla che il receiver non si intasi di dati che non può gestire; controlla che non avvengano congestioni. Non garantisce nessuna temporizzazione e una data quantità di banda. Anche se il TCP è il meglio dei due è in grado solo di evitare perdita di dati.
- ✔ **UDP.** Non fa niente, facile da implementare. Non fa trasferimento affidabile dei dati, se arrivano bene sennò pace. Non fa nessuna connessione. Non fa controllo di flusso o congestione. Non fornisce nessuna garanzia di banda o di tempo. In realtà la sua semplicità lo rende vantaggioso... perchè è incredibilmente svelto. La sua mancanza di controlli e di cose extra lo rende adatto alle interazioni su rete che richiedono incredibile reattività e velocità. Non ci sono garanzie... ma se tutto funziona la velocità è assicurata.

Il software, i protocolli che non lavorano bene muoiono solo se nessuno li usa.

Il Web. Le pagine web sono oggetti che si trasferiscono, individuate da URL. Una pagina può essere anche solo di HTML, ma son orrende. Oltre all'HTML delle volte ci son oggetti referenziati, immagini, musica... tutti oggetti trasferiti nella pagina. L'URL è composto da *hostname* e *pathname*.
Il protocollo HTTP ci permette di scambiare messaggi che i browser usano per comporre le pagine. Il protocollo HTTP si basa su TCP, significa che è un protocollo che prima di mandare pacchetti instaura una connessione. Il client deve creare un socket e chiedere una connessione TCP specificando una porta in ascolto, che è la porta 80. Alla fine della comunicazione questa si chiude.

HTTP usa un servizio orientato alla connessione che viene aperta e chiusa, ma è un servizio *stateless*: quando il server riceve una richiesta non tiene traccia di cosa gli sia stato chiesto, ad ogni richiesta si apre e si chiude la connessione. Viene valutato che il tipo di servizio richiesto è più efficiente in *stateless* che in *statefull*, mantenere lo stato della connessione è difficile ed è fonte di perdita di tempo.

HTTP. Protocollo atto a visualizzare le informazioni sul web. HTTP chiede al servizio TCP di aprire una connessione sulla macchina che fornisce il servizio di server HTTP (la porta di richiesta è la porta 80, per funzionare sul server dev'esserci un servizio già attivo in ascolto), se la connessione è accettata il server manda una ricevuta. Il client riconosce la connessione e manda la vera richiesta che contiene URL ed indirizzo file all'interno del canale di comunicazione interfaccia TCP chiamata socket. Il server allora preleva la pagina e la manda al client, che riceve il file HTML e chiude la connessione. Avviene a livello di protocollo base tutte le volte che viene richiesto un oggetto. All'interno di una pagina web ogni oggetto (suono o immagine) si vedere soggetto dello stesso procedimento spiegato sopra.
Non-Persistent HTTP (**HTTP 1.0**): non si mantiene aperta la connessione per tutto il tempo di trasferimento della pagina, ma esiste una connessione diversa per ogni oggetto. Il tempo totale è dovuto ad un insieme di cose:

- ✔ **Round Trip Time.** Tempo necessario per raggiungere il server ed ottenere da lui una risposta.
- ✔ **Response Time.**
- ✔ **File Transmission Time.**

il tempo totale è dato da *2RTT+transmit time*. Se il file è piccolo possiamo avere un overhead anche molto grande. È possibile evolvere l'HTTP in una versione non non-persistente: **HTTP 1.1**. esso di default fa ancora una connessione TCP con il tempo relativo, ma all'interno della connessione il server manda tutti gli oggetti facenti parte di quella pagina all'interno della stessa connessione. Il round trip time è ancora minore, con pochi RTT abbiamo anche pochi slow start.

Formato del messaggio HTTP. Per la maggior parte si parla di architetture client-server, ma non è detto che le richieste debbano essere codificate tutte dagli stessi protocolli, e che tali protocolli siano tutti uguali. L'importante è che le regole di scambio delle info siano sempre le stesse. In HTTP si definiscono 2 tipi di messaggi, uno di richiesta e l'altro di risposta. Tutti i messaggi tranne il loro contenuto (intestazioni) sono in un formato standard leggibile, ovvero in caratteri ASCII. Si usano i caratteri perchè sono uno standard indipendente dal formato delle macchine, adatto a farne oggetto di scambio tra macchine anche tra loro diversissime.
La *richiesta* deve arrivare con all'inizio la cosiddetta **linea di richiesta**, composta dal verbo (tipo di operazione richiesta al server), poi con la **linea contenente il file** che si vuole ottenere ed il **tipo di protocollo** che si sta utilizzando (versione del protocollo). Ci sono poi un certo numero di **linee di intestazione** che non sono in numero fisso, composte da una struttura opzione-oggetto (user-agent, connessione, linguaggio, cosa accetto). Alla fine per concludere ci si deve mettere un separatore costituito da una **linea vuota** che va a capo: significa che è finita la parte del messaggio che fa la richiesta. Segue poi il *corpo del messaggio*. Nel caso si mandino (e non solo si ricevano) informazioni vanno inserite nel corpo del messaggio, nel caso per esempio di un form tramite metodo POST, con GET posso mandare informazioni sono in piccole quantità.
Nella *risposta* c'è una linea minima obbligatoria detta **status line**, c'è poi un codice numerico con la versione ed uno alfanumerico che corrisponde sempre al codice. C'è un numero variabile di **header line**, poi un **separatore** per distinguere il tutto dal corpo del messaggio. I codici di risposta sono un certo numero (es. 404 pagina non trovata).
Quello che importa dopo aver aperto la connessione è che quella serie di informazioni deve arrivare alla porta giusta nella successione giusta. Con un *telnet* apro una connessione TCP su una porta a scelta (equivalente di SSH, procedura non protetta di login remoto sulla porta 22).

Interazione utente e server. È possibile aggiungere delle funzionalità al nostro sistema, come l'introduzione di richieste speciali tra client e server per richiedere all'utente di autenticarsi per ricevere alcune informazioni. Usiamo una normale richiesta HTTP senza nessuna indicazione, se una delle info che vogliamo è protetta il server nella risposta manderà una richiesta di autenticazione (401, richiesta di autenticazione), niente a che vedere con quelle autenticazioni

serie criptate. Il client (browser) allora compone una richiesta all'utente di inserimento username e password, che vede lui di inserire per bene nel messaggio da spedire al server. L'oggetto è *stateless*, se pensiamo che dobbiamo comunicare con un server il client effettivamente ripresenta tutte le informazioni con la linea di autorizzazione, ma semplicemente invece di chiedere sempre le prende dalla **cache**. L'uso dei **cookies** (usati dai siti web) concerne una linea aggiuntiva che viene messa a fronte di una richiesta ad un server, dal server che risponde alla richiesta. Il browser che usa i cookie mette nella headerline il nostro numero di cookie: serve a rendere tracciabile e statefull un tipo di comunicazione decisamente stateless. *A che cosa servono??* Tracciano le informazioni con più alta quotazione. L'informazione è utile se vengono riportate informazioni interessanti, lo è meno se alla fine si riceve solo spam, posta e popup schifezze. Essere identificati attraverso il cookie è un arma a doppio taglio. Non è solo inutile e dannosa, ma può essere efficace per fare autorizzazioni, utile per "il carrello della spesa", può servire per mantenere uno stato in una comunicazione altrimenti stateless. È un problema di privacy, non sappiamo se alla fine le nostre info saranno ben utilizzate o protette oppure no.
Un **GET condizionale** è una richiesta che ci permette di non riutilizzare la banda della rete per scaricare un file che sia eventualmente già presente in cache, per non perdere tempo inutilmente. Nella richiesta possiamo mettere la linea IF-MODIFY-SINCE: DATE, per evitare di perdere tempo oppure di richiedere cose che possediamo già. Ha senso se abbiamo un sistema ben fatto. È utile quando non abbiamo a che fare con la cache del nostro pc ma quando viene istituzionalizzata la figura di un intermediario che cacha le informazioni: una **web-cache** è un proxy server che fa da intermediario tra client e luoghi di stoccaggio informazioni ufficiali (server), è un servizio frapposto tra client e server raggiungibili come intermediario nel trasferimento. Quando un client chiede ad un server si viene reindirizzati al proprio proxy server che prima di inoltrare la richiesta al vero server va a vedere se il file esiste già nella sua cache, il proxy allora inoltra al server ed in funzione della risposta aggiorna il file oppure mantiene quella cachata. Serve anche per controllare cosa la gente scarica, ma in generale serve per velocizzare le interazioni con il server quando ci sono moltissimi client che devono riferirsi sempre alle stesse pagine, con tempi di risposta brevi e traffico ridotto al minimo solo per le info fresche.
HTTP è uno di quelli complicati come protocollo.

FTP. Acronimo di *File Transfer Protocol*, protocollo che gestisce le regole di comunicazione qualora l'applicazione che si desidera riguardi il trasferimento di file verso o da una macchina remota. Nella realtà i due elementi in gioco sono due macchine concettualmente paritarie, comunque il modello è client-server e le macchine non hanno caratteristiche fondamentali ma sono i ruoli che verranno attribuiti di volta in volta, ed il client è colui che richiede il file come il server è chi lo spedisce. La porta ftp è 21, la sftp è 22.
Come funziona?? L'FTP contatta il server sulla porta 21 facendo una richiesta di connessione. La connessione si apre ed è detta connessione di controllo (caratteristica di FTP anche nella sua versione sicura è quella di avere due connessioni diverse una per il trasferimento file e l'altra per i comandi) che manda i comandi per vedere, mandare e scegliere i file. Una volta aperta ed usata questa connessione di controllo, viene aperta anche una connessione TCP "out of band" (fuori dalla banda della connessione di controllo) che permette di trasferire i dati. FTP è assolutamente statefull: per tutta la durata della connessione il server si ricorda dell'autenticazione e di tante anche cose. È un protocollo che si basa su TCP ma è anche statefull.
I comandi sono molto semplici, con relative risposte. I comandi sono interattivi ma nulla impedisce che esista uno user-agent che intermedia tra utente e server impacchettando i comandi in caratteri ASCII per mandarli all'interlocutore.
All'inizio si mandano username e password (anche in caso anonimo, perchè lo prevede il protocollo), poi si mandano i comandi (LIST per vedere tutti i file disponibili, RETR per ottenere un file, STOR per immagazzinare un file e altri), si riceverà allora un codice di risposta (es: 331 per dire che l'autenticazione è andata bene, 125 ed altri).

Mail elettronica. Nulla impedisce di usare un browser per impacchettare messaggi di posta o di qualsiasi altra cosa utilizzando protocolli precisi. Quando parliamo di posta elettronica parliamo di:

- ✔ **User-agent.** Interfacce utente per gestire visivamente i messaggi di posta. Sono interfacce tra utenti ed applicazioni, non tra applicazioni ed applicazioni.
- ✔ **Server di posta.** Server entro i quali si trovano le mailbox degli utenti, hosy che fisicamente

si scambiano tutti i messaggi di posta degli utenti.

- ✔ **Protocollo del server di posta (SMTP).** Gestiscono lo scambio di messaggi di posta da server a server.

Tutta la posta in ingresso ed in uscita è gestita dai server. In un server c'è:

- ✔ **Mailbox.** Contengono tutti i messaggi di posta letti e non degli utenti;
- ✔ **Coda di messaggi.** Serve ad accodare messaggi che devono essere ancora spediti;
- ✔ **Protocollo.** Permette di mandare "via" i messaggi.

Il client ed il server non sono user-agent. Il server è chi riceve ed il client è chi spedisce, ma chi riceve e chi spedisce è sempre un server mail!! E' venuta prima la posta del web. SMTP è un protocollo molto efficiente nonostante sia molto vecchio. Ascolta sulla porta 25, trasferisce direttamente tra server a server il messaggio. Esistono 3 fasi:

- ✔ iniziale di convenevoli;
- ✔ trasferimento del messaggio;
- ✔ chiusura.

Le interazioni sono domande e risposte, il comando è in caratteri alfanumerici, anche la risposta però è alfanumerica. Il messaggio di risposta è tutto composto da caratteri ASCII. Tutti i comandi sono al massimo 4 caratteri che fanno 4 byte, quindi una parola.
SMTP è più furbo di HTTP: HTTP instaura connessioni non persistenti, mentre SMTP ne usa di persistenti obbligatoriamente; header e messaggio devono essere in ASCII su SMTP con punto da solo per dire che il testo è finito; il messaggio verrà codificato per far in modo che il messaggio venga visto come un insieme di caratteri. HTTP serve per fare richieste di trasferimento tra server e client, SMTP parte sempre dalla parte del client che deve trasferire il messaggio al server. SMTP permette di inviare più pezzi di un messaggio di posta all'interno della stessa connessione TCP.
Il messaggio SMTP ha un **header** con le informazioni di chi ha spedito il messaggio, ed un **corpo del messaggio** separato dall'header da una linea bianca.
MIME: SMTP si è esteso in modo semplice per non cambiare la propria struttura adeguandosi alle esigenze. Nell'header del messaggio si aggiunge la versione di MIME che si sta usando (intanto si rende noto che si userà una estensione multimediale) per comunicare le informazioni necessarie per interpretare una foto, un suono o qualsiasi altra risorsa multimediale.

Mail Access Protocol. Protocollo specifico che recupera i messaggi di posta dal server. È un sistema fornito da una interfaccia grafica (User Agent) che permette di accedere alle informazioni stabilmente memorizzate nelle cartelle di posta degli utenti. Per accedere si utilizzano o il **POP** (*Post Office Protocol*) che con previa autorizzazione permette di scaricare i messaggi e **IMAP** (*Internet Mail Access Protocol*) che fa la stessa cosa di POP ma in più permette la manipolazione dei messaggi non solo sullo user agent ma anche nella casella del server. Oltre ai tipici user agent software esistono anche interfacce web.
In entrambi i casi abbiamo una fase di autorizzazione, con tipico scambio di messaggi che consistono (client) nella informazione di username e password ed in (server) codici di errore o di ok se va tutto a buon fine. Una volta esaurito lo scambio di messaggi tra client e server esiste la fase di transazione che permette la visualizzazione della posta. Le possibili richieste inerenti a questa fase sono poche e semplici (es: lista dei messaggi, eliminazione dei messaggi, uscita dalla procedura, ecc). È lo user agent che conoscendo il nome va a guardare quella parte dei messaggi che serve ad impostare correttamente la pagina dove troviamo l'elenco dei messaggi con il proprio nome, e non solo numerati. Il client può scaricare tutti i messaggi della casella localmente mentre il server può solo scaricarli oppure anche cancellarli.
POP3 è stateless, utilizza a scelta entrambi i metodi di scarica e cancella o solo scarica e tieni. IMAP mantiene sempre tutti i messaggi sul server e li organizza al suo interno (cartelle e sottocartelle, segnalare alcuni messaggi, ecc) ed il mantenimento delle mail sul server è agevole, funziona in modo statefull sennò non ricorderemmo il nome di cartelle ed altre varie modifiche apportate ai messaggi.

DNS (Domain Name System). È uno dei protocolli di base, anche se scriverne il codice è molto complesso. Si tratta di un server che fornisce un servizio di domain name system, ovvero è un *sistema che a livello di applicazione permette alle persone di identificare le macchine in rete*. Per i computer è più semplice usare indirizzi in codice binario, ma le persone non riescono a ricordare gli

indirizzi numerici. Gli indirizzi IP sono a 32 bit, l'identificatore di una macchina in rete è composto da un numero risultante dalla manipolazione di questi 32 bit. Come facciamo a mappare un nome "umano" su un indirizzo IP? Serve un meccanismo ad hoc, la soluzione è il **DNS** *costituito da un database globale distribuito* (= non è in un unico posto monolitico ma è sparpagliato su tante macchine nel mondo) *organizzato in modo gerarchico* (= c'è una struttura a strati ognuno dei quali con uno strato superiore più importante che permette, scendendo, di entrare nel dettaglio e salendo di risalire alle info di carattere generale). Il sistema è un sistema client-server dove tutti i DNS possono lavorare sia come client (coloro che interrogano) che come server (coloro che sono interrogati), se il server non sa rispondere diventa client e chiede al server superiore e così via. È un servizio di livello applicativo anche se sembrerebbe di livello di rete, ma a livello 3 non interessa nulla del nome, serve l'indirizzo IP. Questo servizio è servito a livello applicativo perchè è l'interazione con l'umano che bisogna interfacciare, inoltre i router devono essere rapidi ed evitare di essere appesantiti. I server di livello più basso prendono due nomi differenti, a seconda che siano quelli con l'informazione primaria o quelli ai quali noi ci riferiamo.
Il sistema è organizzato in modo gerarchico e distribuito perchè, se fosse altrimenti, avremmo una macchina enorme e moltissimo utilizzata nello stesso istante, ma soprattutto localizzata per alcuni molto vicino ma per altri anche dall'altra parte del mondo rendendo così differentemente efficiente il servizio; inoltre, le reti sarebbero sempre intasate con il tipico "collo di bottiglia"; i computer non sono eterni e se si guastasse tutto il mondo rimarrebbe "al buio"; ci vorrebbe moltissima gente per mantenere aggiornato ed operativo il DNS centralizzato.
I DNS vicini agli utenti sono detti DNS locali, sono i DNS che le macchine di una rete o di una istituzione (o provider) interrogano perchè contengono tutte le info delle macchine di quella istituzione e sono quelli interrogati dalle macchine di quella istituzione. È quello che per default si interroga per assegnare alla macchina un indirizzo IP associato al loro nome. Una macchina locale può comunque chiedere un indirizzo IP della sua rete oppure un IP esterno. I DNS autoritativi stoccano per gli host i loro IP ed in loro nomi e performano la traduzione nome/IP per ogni host.
I DNS Root sono pochi (solo 13) e sparpagliati, il grosso si trova negli Stati Uniti ed in Europa, nella parte più abitata dell'Asia ma pochissimi in Africa. Sono quei DNS contattati in un qualche modo tramite intermediari dai DNS locali quando nessun'altro è in grado di risolvere i nomi.
Autoritativo è il server che ha la copia originale dell'indirizzo di quel nodo. Il Root conosce l'ultimo pezzetto dell'indirizzo. L'utilizzo incrociato con i server locali ci permette di avere sempre le informazioni. Il livello più alto ha le informazioni che vedono l'insieme più grande. I DNS locali sono il livello più basso della gerarchia ai quali viene fatta la richiesta dalle macchine, quando usiamo il DNS autoritativo lo interroghiamo come un proxy.

(causa "mal di panza epocale" non da cagotto ma da mestruazioni, a questo riguardo mancano diverse informazioni perchè detto tra noi avevo altro a cui pensare. Voi uomini non capirete mai cosa vuol dire avere l'utero che si sfalda periodicamente...)

Protocolli informali. Oltre alla struttura tipica client-server esistono altre architetture non convenzionali. L'estensione logica del client-server è il **Peer-to-Peer**, argomento di larga diffusione che però non rientra negli standard e nei protocolli standardizzati.
Quando tempo è necessario per distribuire un file da un server ad un client (o da un server ed un suo pari)? In modo standard, dobbiamo tenere conto di:

- ✔ *Banda di upload del server.* Capacità che ha il server di scaricare verso la rete il file;
- ✔ *Banda di upload del peer.*
- ✔ *Banda di download del peer.*

Ognuno ha la sua banda, e quindi il suo tempo di upload e download.
Il server, se deve mandare n copie a n client, mette in sequenza n file ed il tempo necessario è il numero di file per la dimensione dei file diviso la banda di upload. Il client avrà bisogno di un tempo pari alla dimensione del file sul tempo di download. Il tempo totale che è richiesto è il massimo tra questi due.

P2P Pura. È una architettura magnifica di difficile realizzazione, già spiegata in precedenza. Nel caso in cui gli stessi file di prima vengano trasmessi in P2P succede che: il peer distribuisce una copia che gli altri peer si distribuiscono poi tra loro, allora il server non manda n file ma uno solo (dove il server è il peer che possiede il file), dopodichè il server ha esaurito il suo compito. Ogni

client ha bisogno di un tempo pari alla dimensione del file sulla banda di download. La quantità aggregata di dati di cui consideriamo il download è comunque il numero di file moltiplicato per la loro dimensione. La velocità massima è data dal singolo peer più la somma dell'upload di tutti gli altri.

- ✔ **Bit Torrent.** I peer che si scambiano pezzi di file si chiamano torrent. I tracker tengono traccia di "chi ha che cosa". Supponiamo che un peer che vuole entrare in un torrent si rivolga ad un tracker (ad esso un nodo fa la richiesta di entrare in un dato gruppo di scambio file, allora il tracker distribuisce la lista dei file a disposizione). I chunk sono pezzetti di informazione, unità di file in cui viene spezzettata ogni informazione in trasferimento. Un nodo "novizio" può solo accumulare file, non avendo niente da scambiare. In seguito, si scelgono un numero di peer vicini ai quali inviare i propri chunk (sono i privilegiati ai quali rivolgersi per il download o per ricevere il download, non sempre a ricevere ed a inviare sono gli stessi peer). È necessario anche rimanere informati di quale sia l'insieme di "vicini contattabili" forniti dal tracker, ovvero di coloro che sono vivi ed attivi. Niente di tutto questo è standardizzato, in sostanza i peer possono esistere o non esistere durante il trasferimento: una volta completato i chunk che servono un peer può spegnersi troncando tutte le comunicazioni. Periodicamente ogni peer chiede ai vicini la lista dei file disponibili e farà quindi richiesta dei chunk che mancano. Tante copie dello stesso chunk saranno in diversi peer, allora si chiede "per primo" il "più raro" (l'aumento di copie dei più rare diminuisce la loro rarità, però scaricando la copia più rara subito evito di perderla di vista). Si scelgono 4 vicini referenziali, quelli su cui c'è possibilità che il trasferimento avvenga al massimo rate disponibile (non è un gruppo statico, può cambiare conformazione in continuazione a periodi molto brevi). I 4 elementi che fanno parte dell'insieme di peer privilegiati si chiamano "elementi non ???". Se un nuovo peer entra nell'insieme dei 4 si elimina dei precedenti il peggiore.
  NOTA BENE: *tutto ciò implica una grande complessità implementativa, e quanto ottenuto deve funzionare alla perfezione. Lo scopo è ottimizzare lo scambio di file ma usando la stessa banda per trasferire file e mandare tutti i messaggi indispensabili per rendere possibile il trasferimento dei file.*
  Dove troviamo tutte queste informazioni? Il tracker tiene conto di dove sono peer e chunk. Abbiamo bisogno di sapere *dove è la macchina* e *quale è la porta del servizio peer-to-peer*. Una volta individuate le macchine devo sapere *quali file sono disponibili alla condivisione*. I peer devono dichiarare ad un "indice" sistematico che dinamicamente tenga traccia di tutti i file dei peer. Ogni peer man mano deve indicizzare i suoi chunk di file e comunicare ogni aggiornamento, dall'altra parte questo indice deve essere consultabile da qualsiasi altro peer.
- ✔ **Istant Messaging.** Invece di cercare i chunk di file dobbiamo sapere *chi è in linea*. Le informazioni sull'indice riguardano anche *su quale macchina si trova quell'utente*.

Per creare l'indice, è più comodo avere un indice (database) centralizzato. Si prenda ad esempio la struttura del Napster originale: ogni peer per connettersi doveva informare un sistema centralizzato che funzionava da tracker e da indice di chunk. Esistono però dei problemi con i sistemi centralizzati: se si blocca il computer centrale muore il servizio; possono crearsi congestioni; la distribuzione si basava su contenuti protetti da copyright creando così problemi legali (non si può localizzarlo a tal punto da rendere le informazioni visibili a chiunque le chieda).

Query flooding. Si deve pensare ad un sistema completamente distribuito senza nessun server centrale: ogni peer fornisce anche un pezzetto del sistema di database distribuito rendendo distribuito l'indice dei file e dei chunk posseduti, non dando info su file che non possieda. I peer si devono dare da fare per costruire la rete e rendere noto cosa scambiare. *Overlay network*: tutti i peer appartengono allo stesso network, i nodi sono peer, la connessione tra i peer è virtuale e mai fisica (non a livello datalink, ma collegati tramite una connessione TCP), tipicamente quando il sistema è in funzione ogni nodo ha un collegamento con altri ma non con tutti e solo ad un numero limitato inferiore a 10 (altrimenti sarebbe ingestibile).
Ovviamente però costituire un sistema completamente decentralizzato è difficilissimo, se non impossibile. La filosofia è quella del **flooding** (= allagamento) *inondando la rete di messaggi*. Si mandano un sacco di query da tutte le parti, quando un peer viene "colpito" da una query il peer risponde, poi la inoltra ai vicini alla ricerca della risposta alla query. Si formano allora connessioni

TCP seguendo il percorso tra i peer che hanno inoltrato la query, per poi connettere direttamente la macchina richiedente con quella che possiede i file richiesti.
La scalabilità del sistema è indecorosa: si interrogano tutte le macchine, si aspetta che rispondano, si instaurano connessioni TCP ecc... per funzionare la cosa deve essere ripetuta e ripetuta ancora. L'idea originale infatti era quella di rimanere in un piccolo range di peer.
Overlay gerarchico: esistono relazioni tra gruppi a loro volta centralizzati. Ci saranno nodi detti "supernodo" a cui fanno riferimento gli altri, allora i supernodi si connettono tra loro e si mandano informazioni. Un supernodo però è anche un nodo normale (un nodo normale però non è anche supernodo!), i peer sono nodi base mentre il supernodo è anche peer ma è gerarchicamente più importante di un nodo normale perchè è un punto di riferimento. Le connessione TCP saranno più semplici ed instaurate solo tra nodi e supernodi. Le info scambiate dai supernodi sono rese disponibili ai peer normali.
NOTA BENE: *non esistono standardizzazioni, la gran parte dei prodotti peer-to-peer sono privati con codici chiusi. Non viene esplicitamente detto se il supernodo funziona solo come informatore o se regolarmente fa da tramite per il traffico dei nodi normali.*

(jump!)
Programmazione Socket. Interfaccia tra livello applicativo e livello di trasporto. Indica quali comandi utilizzare per passare le info al livello di trasporto. La socket permette di passare attraverso l'interfaccia un insieme di dati: si crea la socket, la si usa e la si rilascia perpetrando il paradigma client-server. Nel caso TCP si considera che il servizio si basa su un protocollo che fornisce un servizio affidabile di scambio byte da un processo all'altro. Con UDP non c'è nessuna connessione e nessun bisogno di aprire e chiudere connessioni, essendo il pacchetto abbandonato a se stesso.

Lezione 3 ~ Livello di Trasporto

Livello di trasporto. Essenziale è il *multiplexing/demultiplexing*. Altri possibili servizi che il protocollo potrebbe (ma non ci sono obblighi) fornire sono *reliable transfer*, *flow control*, *connection management*. I protocolli UDP (senza connessione) e TCP (con connessione e controlli vari). Per parlare della gestione e del controllo della congestione abbiamo bisogno di capire di che si tratta, come prevenirla e come "curarla".
Ogni protocollo parte dal presupposto che dal punto di vista logico la comunicazione avvenga direttamente tra protocolli tra loro pari a lato client e server. Sono due processi che girano sulle due macchine che comunicano tra di loro. Il trasporto ha senso e deve esistere solo negli end-node (dove girano anche le applicazioni), può esistere anche nei router ma non è necessario. Il livello di trasporto sta al livello di rete come il livello applicativo sta a quello di trasporto. Il livello di trasporto fa parlare processi tra macchine diverse.
I possibili servizi da chiedere al livello di trasporto possono essere:

- ✔ farsi mandare tutti i dati in ordine, quindi un sistema di trasferimento affidabile di dati ordinati e completi. Posso avere uno stream di dati (successione di bit) che voglio ottenere interamente e nello stesso ordine di erogazione.
- ✔ controllo della congestione per controllare che la comunicazione non sia "imbottigliata" da qualche parte nella rete.
- ✔ controllo del flusso per vedere se il mio interlocutore è in grado di ricevere tutti i bit in arrivo.

Il flusso dei dati deve essere compatibile e sostenibile, alla velocità massima consentita ma non di più; è importante che tutti gli attori siano capaci di ricevere e di mandare i dati perfettamente senza combinare pasticci. In multicast non si ha un solo interlocutore, allora si deve scegliere come mandare a tutti gli stessi dati (velocità minima? Dimensione minima?).
Il livello di trasporto inoltre potrebbe fare:

- ✔ una consegna unicast o multicast nei limiti del possibile. In questa visione non dettiamo preferenze sull'unicast o sul multicast. Tipicamente TCP fa solo unicast in maniera affidabile, UDP fa il meglio possibile per il resto.
- ✔ una consegna di dati nell'ordine giusto e nei tempi giusti rispetto ai tempi di partenza.
- ✔ bada garantita.

- ✔ multicast affidabile (con implementazione del protocollo molto sofisticata).

Multiplexing/Demultiplexing. **Demultiplexing** è quel servizio che permette al server di consegnare il messaggio inviato alla corretta applicazione che lavora sul server. Con le info messe nel segmento il livello di trasporto consegna la parte di dati all'applicazione giusta (essendo che in ogni macchina esistono tanti altri servizi attivi). **Multiplexing** vuol dire mettere i dati giusti in modo che l'applicazione quando prepara il pacchetto siano leggibili una volta arrivata a destinazione. A livello base è essenziale prendere il messaggio (ovvero tutti i dati che arrivano dal livello di applicazione) con la sua intestazione ed aggiungere l'intestazione di trasporto all'interno del quale devono esserci le info necessarie per il multiplexing ed il demultiplexing.

UDP (User Datagram Protocol). Il servizio di base richiederebbe solo i 32bit delle porte sorgente-destinatario. UDP non offre nessun servizio: i segmenti possono disordinarsi o non arrivare mai, non c'è nessuna connessione ed ogni pacchetto è lasciato a se stesso. Il pacchetto è sezionato ed inviato a segmenti numerati. Ogni pacchetto percorre strade diverse, anche ogni segmento percorre strade diverse senza nessuna garanzia che i segmenti arrivino tutti in ordine oppure no. UDP non perde tempo nell'instaurare la connessione, non controlla congestione e quindi guadagna anche questo tempo. È veloce e semplice, segmenti abbastanza piccoli.
Nella realtà UDP non ha informazioni a parte numero di porta sorgente e destinatario, il resto è informazione, lunghezza e checksum (controllo del contenuto del messaggio per controllare che non ci siano "danni" nell'informazione). Se un pacchetto non arriva, lo si richiede. Usare UDP per applicazioni importanti è possibile, se va male si può implementare il servizio di trasferimento affidabile dei dati a livello di applicazione. Il ***checksum*** è lungo 16 bit, come detto prima è un sistema rudimentale e veloce per controllare l'esistenza di eventuali errori nei bit di segmento (ma non nel messaggio intero!), se i bit sono corretti il segmento è buono. Si tratta di una somma di numeri a 16 bit (fittizi), la somma viene messa nel campo checksum e spedita. Il ricevente ricalcola il checksum alla stessa maniera e controlla se i dati corrispondono. Se i campi non sono uguali si scarta il pacchetto. Il checksum non è essenziale per il controllo della qualità dell'informazione. Il controllo però va fatto perchè a livello di rete non è garantita nessuna garanzia di correttezza dei bit (è vero però che a livello datalink esistono controlli di qualità molto migliori).

Trasferimento affidabile dei dati. È tutta roba teorica, facciamo ipotesi formali. Il trasferimento affidabile dei file è un problema basilare di ogni architettura. Il livello applicativo si basa sul fatto che esistono due processi remoti che si scambiano dati, per dar luogo alla comunicazione ci si affida ad un livello sottostante che deve per forza essere affidabile (così la pensa l'applicazione, che vede il sottostante come una scatola nera). Il server manda dati, si sa che il canale sottostante è dichiarato inaffidabile, allora devo rendere affidabile il servizio. Per rendere affidabile la comunicazione devo far passare le info attraverso un meccanismo affidabile.
Se tutto va bene (cosa che non succede MAI), il pacchetto si trasforma da inaffidabile ad affidabile e viceversa. Si introduce allora un errore alla volta e consideriamo macchine a stati finiti.
Se il canale non perde pacchetti ma è possibile che subisca delle interferenze, il pacchetto viene spedito e ricevuto ma il checksum non corrisponde (il pacchetto allora viene scartato). Bisogna informare in qualche modo che il pacchetto non è arrivato. Introduciamo allora il concetto di "ricevuta" che è positiva se il pacchetto arriva bene, negativa se arriva male. Il sender allora manda, il receiver prende e controlla, se il pacchetto è danneggiato manda la ricevuta negativa (feedback al sender). Il sender si stoppa ed aspetta la ricevuta.

(causa cottura mentale, mancano alcune informazioni finali su come il livello di trasporto gestisce gli errori e la perdita dei dati)

Si controllano i riscontri per vedere se anche loro sono corrotti. I numeri di sequenza controllano la successione dei pacchetti inviati, così da controllare i "doppioni" che eventualmente vengono immessi sulla rete se il sender non riceve una ricevuta e ri-spedisce il pacchetto.
Ci sono ampie distanze tra sender e receiver, così che loro sono a conoscenza solo di quello che LORO fanno e non ciò che invece fa l'interlocutore. Nel momento in cui il sistema può corrompere anche i riscontri, non ha senso distinguere tra positivo e negativo (soprattutto da quando mettiamo un numero seriale). Differenziare i pacchetti comporta differenziare anche codice.

Si possono creare protocolli che non usano riscontri negativi (ma solo positivi). In pratica se non ricevo un riscontro, vuol dire che non è successo niente.
Sono comunque ipotesi troppo ottimistiche (come se i problemi sulla rete fossero sono questione di bit danneggiati), i pacchetti potrebbero non solo corrompersi, ma anche non arrivare a destinazione.

Pacchetti persi. Esistono già num. sequenza, riscontri e checksum. La possibilità di ritrasmettere pacchetti è concreta. Come fare per capire se un pacchetto si è perso oppure no? Si introduce un tempo di attesa, mentre nell'ipotesi precedente i pacchetti arrivavano sempre ed il tempo di attesa terminava con la consegna del pacchetto. Se il pacchetto non arriva, il receiver non può sapere se e quando è stato inviato. Il sender allora attende una "ragionevole" quantità di tempo, se nel mentre non riceve ACK allora ritrasmette il pacchetto, che potrà essere stato perso o corrotto, ma non importa saperlo. Il tempo in esame può essere stimato male, e può succedere che il pacchetto venga ritrasmesso anche se l'originale non è andato perso ma solo rimasto in attesa magari davanti ad una congestione.
Consideriamo di avere un guasto nella rete quando stiamo per ricevere il riscontro. Il sender allora rimanda il vecchio pacchetto ed il receiver riceve la seconda volta lo stesso pacchetto, allora rimanda il riscontro del vecchio pacchetto e scarta il duplicato.
Se il tempo di timeout scade troppo presto, il sender rispedisce un pacchetto prima che arrivi il riscontro del receiver. Il sender allora potrebbe attende un riscontro vecchio ricevendo invece quello nuovo, allora per contraddizione si formerà un certo casino e bisognerà riavviare la procedura.

Performance. Le prestazioni del sistema di trasferimento dati senza perdita di informazioni sono scarse. Ne vale davvero la pena? Utilizzare un sistema dove chi manda e chi riceve si ferma ad attendere una ricevuta non è ottimale. Il tempo di attesa non deve essere minore del round trip time: se è giusto, appena un pacchetto è un po' grande non funziona, se il tempo è troppo corto non funziona per niente... allora il tempo d'attesa va settato maggiore del round trip time.

Protocollo di pipeline. I controlli vanno mantenuti ma si fa in modo che il trasferimento avvenga in "catena di montaggio". Le stesse operazioni di prima vanno ottenute senza sprecare banda. Le *pipeline protocol* permettono di accodare tanti pacchetti e di fare comunque tutti i controlli gestendo i riscontri non sul singolo pacchetto ma considerando che siano processati in maniera parzialmente asincrona (il sender gestisce un tot di pacchetti senza sapere cosa succede ad ogni singolo pacchetto). Il receiver allora deve essere in grado di gestirne un tot, senza badare al numero di successione di ogni pacchetto. I pacchetti adesso vanno bufferizzati (da una delle due parti) introducendo tutti i controlli con il numero di sequenza.
Le pipeline incrementano le prestazioni del trasferimento in rete.

- ✔ **Go-Back-N.** (*Torna a casa N).* Il sender si dota di un buffer detto *window* che memorizza un numero maggiore di 1 fino a N di possibili pacchetti in attesa di essere spediti e riscontrati. Finchè un pacchetto è spedito o riscontrato non esce dal buffer. Quando il sender riceve il riscontro per quel pacchetto lo elimina dal buffer. Ogni pacchetto ha un numero di sequenza al massimo uguale a N. Nella finestra del sender i pacchetti possono essere già riscontrati (ma sono fuori dalla finestra), pacchetti già spediti ma non riscontrati e pacchetti che non sono stati né spediti né riscontrati. Se il buffer è pieno il livello applicativo si ferma nel mandare pacchetti a quello di trasporto. Il riscontro è cumulativo: non viene mandato un riscontro per ogni pacchetto, ma viene mandato un riscontro per l'ultimo pacchetto ricevuto bene allora il receiver non ha bisogno di un buffer. Il sender allora manderà altri pacchetti a partire dal successivo all'ultimo che il receiver ha notificato.
  Sono asincroni: ad ogni invio non corrisponde una relativa notifica.
  Introdurre semplicemente buffer e numero di sequenza aiuta a migliorare la situazione, guadagnando molto in bassa probabilità di perdere pacchetti (prima si perdeva tempo e basta, a prescindere). Soluzione povera, nata in cui le risorse erano basse e costose.
- ✔ **Selective Repeat.** Spreca ancora meno banda ripetendo (=ritrasmettendo) in maniera selettiva, seleziona quindi i pacchetti che vanno rispediti da quelli che invece non vanno rispediti. Riscontra individualmente ogni pacchetto con il suo numero di sequenza, introducendo un buffer a lato receiver e mettendo da parte i pacchetti che arrivano con

num.seq. più alto. Il sender allora rispedisce solo quelli che non sono arrivati, e non tutta la sequenza a partire da quello non consegnato.
La finestra del sender ha un primo pacchetto seguito da un certo num. di pacchetti nella sequenza giusta che possono essere riscontrati o meno. La finestra di ricezione ha il primo pacchetto della lista ed un num. di slot occupati dai pacchetti arrivati anche in disordine, poi slot per i pacchetti attesi ed uno spazio rimanente importantissimo da usare per future ricezioni. Se il lato sender è pieno, deve attendere; se il receiver è pieno, non può fare niente (semmai scartare).
Come scelgo il numero di sequenza? Se il num. sequenza è maggiore della dimensione della finestra, vuol dire che può capitare che il sender non si possa più sbloccare. La finestra del sender avrà il buffer pieno.
Il numero di sequenza deve essere almeno il doppio.

Confrontando le due versioni, Go-Back-N ha un sender che può spedire un tot di pacchetti non ricontrati in pipeline nella rete, stessa cosa anche il Selective Repeat. Il SR però può segnalare l'unico che manca e ricevere solo quello.

TCP. Ne parliamo solo adesso perchè lui gestisce il trasferimento "sano" dei dati.

DOMANDA: i modelli di pipeline.
RISPOSTA: modelli di pipeline.
DOMANDA: modelli di trasferimento affidabile del TCP.
RISPOSTA: il TCP ed eventualmente a quale modello fa riferimento (come funziona).

TCP ha un funzionamento particolare, che dimostra che i modelli sono importanti ma per farli funzionare si deve scendere a compromessi. La comunicazione è ancora tra sender e receiver, fornisce un servizio di byte stream affidabile e completo, per cui i byte che partono arrivano tutti e tutti ordinati. Utilizza un sistema di pipeline, e per funzionare bene implementa il controllo del flusso e della congestione utilizzando le "finestre" indispensabili alle pipeline. Il TCP ha sender e receiver con buffer. È full-duplex. Il messaggio si chiama segmento e se è tanto grande viene spezzato in segmenti più piccoli. È orientato alla connessione quindi prima di iniziare il trasferimento instaura una connessione tramite handshaking. Fornisce un controllo del flusso (così che il sender non esageri a mandare pacchetti ad un receiver che non può riceverli).
Sender e receiver sono intercambiabili, in termini di ruoli, e come loro anche i loro buffer.
L'header di TCP comprende *checksum* (come in UDP), *lunghezza dell'header* perchè non è dimensione fissa, *porta sorgente* e *destinataria*, *numero di sequenza e di riscontro* che ci permettono di gestire lo scambio dei segmenti, *flag di dati/risposta*, e una serie di flag per definire dati urgenti, dicono se il numero di ack è valido, i dati sono prioritari ed altro, ed un campo che comunica la dimensione della finestra di chi riceve..

Numeri di sequenza. Come funzionano in TCP? Si usano gli stessi pacchetti sia in ricevuta che in invio. Il **numero di sequenza** è il numero del primo byte del messaggio da inviare. L'applicazione manda una richiesta di spedizione messaggio al livello di trasporto. Il TCP prende il messaggio e se è più grande delle dimensione massime del segmento lo divide in pezzi. Allora in numero del primo segmento è il numero del primo byte. Si riempie allora tutto il segmento ed il numero di segmento successivo sarà il primo byte della "seconda parte" e via di seguito. Si usa questa numerazione che non è sequenziale.
Il riscontro del TCP in risposta al primo segmento è della serie "ok aspetto il primo segmento della seconda serie". Questo vuol dire che è stato ricevuto tutto lo stream dei segmenti precedenti fino alla fine della serie. L'ACK è sempre relativo al numero di sequenza che ci si aspetta in seguito, della serie "mi è arrivato A, mandami B".
Funziona in pipeline ma non sfrutta Go-Back-N puro né Selective Repeat puro, poiché usa i buffer da tutte e due le parti ma anche l'ACK cumulativo, ottenendo una efficienza superiore. I pacchetti possono essere persi, che siano essi dati o ricevute; se utilizzare riscontri rigidi implica rispedizioni di pacchetti persi o danneggiati. Non rispedisco pacchetti se è andato perduto solo il riscontro.
In caso di guasto, ovvero se si impalla qualche cosa, si deve ritrasmettere. Se il pacchetto parte con un numero di seq. ed una certa dimensione viene correttamente accettato, viene spedito dall'host un pacchetto di riscontro per quel pacchetto. Se il pacchetto si perde lo dobbiamo ritrasmettere semplicemente.
Se i pacchetti vengono spediti e riscontrati correttamente, e per qualche motivo il sistema fallisce

nel definire il tempo di timeout, quando arriva il pacchetto ritrasmesso il receiver prende atto che lui era fermo ad aspettare e così invece di mandare un riscontro per quel pacchetto ritrasmesso lo manda per l'ultimo pacchetto arrivato chiedendo il prossimo.

Flow Control di TCP. Se i buffer sono pieni o quasi e l'interlocutore continua a mandare pacchetti, il sistema va in crisi. Il **controllo di flusso** è un meccanismo che rende corretto il funzionamento dei buffer di ricevuta e di invio. *È un controllo che ci permette di non intasare il buffer ricevente con più pacchetti di quelli che può gestire, allora il sender deve moderare quantità e velocità di emissione.* In ogni pacchetto scambiato esiste un campo che ci dice quanto spazio libero c'è nel buffer del ricevente (il campo presente nell'header è il *Receiver Windows* = parte rimasta libera nel buffer del receiver), è una info aggiornata dinamicamente per ogni pacchetto inviato. Il receiver informa direttamente il sender della quantità di spazio libero, il sender allora utilizza l'informazione per avere le dimensioni del buffer di spedizione minori o uguali della finestra di arrivo.

Round Trip Time (RTT) e Timeout nel TCP. Bisogna definire correttamente il tempo di attesa... ma come? Il tempo deve essere almeno maggiore del RTT però l'RTT non è di dimensioni fisse (anche se parliamo di TCP e di comunicazione logica tra sender e receiver, la comunicazione passa comunque nella rete ed è questione di link fisico ogni tipo di ritardo). Se la valutazione è troppo breve si perde il timeout e si deve ritrasmettere il pacchetto utilizzando risorse utili per pacchetti inutili (con allungamento dei tempi). Un tempo troppo lungo invece butto via la possibilità di reagire prontamente alla perdita di un pacchetto.
Bisogna fare una stima corretta del RTT, tenendo conto della variazione che accade in ogni momento. È una stima statistica, per arrivare alla stima di un tempo medio di percorrenza del pacchetto. Una valutazione corretta del vero RTT deriva dall'eliminazione di tutte le ritrasmissioni del pacchetto e di tutti i pacchetti dei quali abbiamo ricevuto un riscontro cumulativo. Nella selezione dei tempi si prende in considerazione non solo i timer ma anche i tempi effettivi di riscontro singolo eliminando i segmenti rispediti e non singoli.

Handshake a tre vie. TCP manda il suo primo pacchetto con tutte le variabili del caso per iniziare la connessione sincronizzando sender e receiver. Il ricevente manda allora un pacchetto di riscontro per dire che la connessione è accettata, tale pacchetto dichiara che la richiesta è stata ricevuta, accettata e che quelli che manda sono i suoi numeri di riscontro. Quando il sender riceve la contro-ricevuta può incominciate a trasmettere. Quando la connessione non è più necessaria il client che ha stabilito la connessione deve mandare un pacchetto al server per segnalare che ha finito. Il server allora prende il pacchetto e risponde senza riscontro la sua ricevuta di chiusura di connessione e chiude dopo questo invio la comunicazione. Dopo un certo tempo di timeout la connessione è completamente chiusa.
NOTA BENE: *una macchina che ha ricevuto tutte le richieste di connessione possibili e nessuna richiesta di fine, è una macchina che diventa cieca e che dismette il servizio.*

Controllo di congestione. Diverso la controllo di flusso. TCP gestisce il corretto ed affidabile trasporto di dati ed è specificatamente unicast. Comunque la comunicazione è da molti-a-molti, la congestione può comunque degradare la trasmissione dei dati. Quando i pacchetti si perdono o sono scartati oppure si intasano nella coda di attesa di qualche router.

- ✔ **Scenario #1.** Non abbiamo sender e receiver, ma 2 di uno e 2 dell'altro ed un solo router di mezzo. Nella realtà i seder ed i receiver sono infiniti. Ipotizziamo anche un buffer infinito. Finchè il numero di pacchetti che l'host spedisce è più piccolo o al massimo uguale alla metà della banda disponibile, il buffer non ha problemi e si può aumentare il numero di pacchetti in rete, fino alla saturazione. Nel momento in cui si arriva alla saturazione le richieste di aumento non attecchiscono più ed i pacchetti spediti sono in numero massimo definiti dalle possibilità della rete (= la richiesta aumenta ma il numero di pacchetti mandati è fisso). Finchè si sta sotto la quantità di banda disponibile si può crescere, ma poi no.
- ✔ **Scenario #2.** Se il buffer è finito, i dati possono andare perduti. Il tempo aumenta assieme al numero di pacchetti che andremo a ritrasmettere. Se gli andamenti prima erano lineari, ora non più. La banda che dobbiamo tener d'occhio non è più metà della quantità, ma ancora di meno. Aggiungendo ad essi non solo i pacchetti ritrasmessi ma anche un timeout

sottodimensionato, si scende ulteriormente. L'efficienza degrada più la rete è congestionata.

- ✔ **Scenario #3.** Si può ottenere un blocco delle reti, quando i pacchetti consegnati sono pari a zero. La rete può essere consumata da pacchetti che non arrivano mai a destinazione.

Saturare una rete vuol dire congestionarla.
Il controllo del flusso è indispensabile per la comunicazione tra macchine, ma quello della congestione è indispensabile per la comunicazione tra qualsiasi due macchine del sistema. Per tenerne conto esistono 2 approcci:

- ✔ la rete da una mano a gestire il problema;
- ✔ (scelto) IP non fornisce nessun sistema di controllo della connessione, quindi si deve implementare a livello di rete. Questi sistemi fanno sì che i router, man mano che passano i pacchetti, si mandino informazioni simili a quelle delle finestre utilizzando certi dati negli header dei pacchetti. L'opzione minima comporta un bit che segnala o meno una congestione.

Nella realtà IP non da feedback a TCP, allora TCP deve avere un approccio investigativo degli effetti della congestione. La rete può segnalare la congestione, sennò TCP deve analizzarne il comportamento a seconda di cosa arriva e di come arriva

Caso di studio: controllo di congestione ABR di ATM. ATM non è un protocollo, è una architettura indipendente sviluppata per trasmettere fonia, essendo digitale è stata sfruttata poi per trasmissione di dati. ATM ha un sistema che consente il controllo della congestione: *Available Bit Rate* che fornisce un servizio di banda elastica. Il sistema indica eventualmente il sovraccarico della rete, usando entrambe le soluzioni con flag e con campo esplicito di quale sia la banda suggerita. ATM non funziona con datagram o segmenti ma ha la sua architettura con la **cella**. La cella ATM è di 53 byte! Tenta di collimare comunicazione dati con quella vocale. Ci sono celle speciali per trasporto dei dati (= Resource Management Cells) che comunicano come e quando modificare il rate di spedizione di dati segnalando la presenza o la scompara della congestione.

Controllo della congestione di TCP. Possiamo diminuire o aumentare la dimensione della nostra finestra per agevolare il traffico. Si deve tener conto dei limiti di flusso e congestione e posizionarsi sulla più piccola delle due.
Per diminuire il rate si può solo diminuire le dimensioni della finestra di invio dei pacchetti, perchè la banda è data dal numero di segmenti per le dimensioni massime del segmento diviso il tempo per farlo sparire dal buffer (=num. di sec. necessario per spedire il pacchetto e ricevere il riscontro, ovvero il RTT un pochino allargato).
Si applica il buonsenso tipico della vita reale.

Slow Start. Partenza in due fasi o **slow start**, dove esiste una fase in cui si cerca di schivare la congestione. La soglia è il valore che definisce il confine tra la fase di partenza lenta e la fase di regime della congestione. Si incomincia con la finestra di congestione pari a 1. il primo pacchetto mandato è sempre 1, se arriva il riscontro si raddoppia la finestra di congestione, e se si riscontrano si raddoppia di nuovo... avanti così. L'aumento non è lineare ma cresce molto velocemente in potenze di due. Se si perde un riscontro allora, a seconda della implementazione: si aspetta a decretare la congestione a seconda del ritardo di consegna o dello smarrimento del pacchetto.
Si va avanti a moltiplicare fino al raggiungimento della soglia (partenza *meno* andamento di crescita lineare) allora l'incremento diventa lineare. Andando avanti quando giunge l'evento di congestione si dimezza il valore della soglia e o si ricomincia da 1 oppure si ricomincia dall'incremento lineare (versione *Reno*).

AIMD. *Per essere cauto cresco aggiungendo e diminuisco dividendo.* Permette di raggiungere un certo equilibrio ma c'è bisogno che il sistema di fair (= equo).

Latenza. Quanto visto fin qui è strettamente legato alla latenza. Abbiamo dati da trasferire ed il liv. di trasporto per far in modo che succeda introduce un sacco di operazioni che provocano ritardi e casini.

Lezione 4 ~ Livello di Rete

Livello di rete. Tutto quello che riguarda la gestione del corretto smistamento dei pacchetti nella rete. Tralasciamo la parte della "necessità" dell'applicazione e guardiamo il servizio sempre più vicino all'hardware di trasmissione dei pacchetti. Tali servizi sono:

- ✔ **Routing.** Selezione del percorso che ogni pacchetto deve seguire nella rete. Dobbiamo tener conto della scala, ovvero delle dimensioni dell'infrastruttura. Guardiamo i router, il protocollo di rete Ipv4/IPv6.
- ✔ **Modelli di servizio.** Servizi che idealmente possono essere richiesti a questo livello. Tra questi gli algoritmi di instradamento.

Funzioni base. Sugli end node c'è tutta la pila protocollare. Spostare i pacchetti però è compito anche dei router intermedi tra i nodi terminali. Esistono 3 funzioni fondamentali:

- ✔ **Routing.** Trovare la strada per ogni pacchetto perchè possa arrivare a destinazione. Se ne occupano gli algoritmi di instradamento.
- ✔ **Forwarding/Switching.** Correlata ma diversa dal routing, consiste nel far sì che il pacchetto si muova da una porta di ingresso di un router alla porta di uscita dello stesso router.
- ✔ **Setup della chiama.** Non avviene sempre ma solo dove l'architettura prevede che ci siano chiamate che verranno utilizzate per instaurare una connessione che sia utilizzata sempre per ogni singolo trasferimento (equivalente di una connessione TCP) attiva fino alla fine della comunicazione.

Il servizio base deve prevedere almeno il forwarding.
Jitter: fluttuazione temporanea del segnale. Potremmo desiderare che la comunicazione sia isocrona mandando pacchetti in maniera regolare. Vorremmo che i pacchetti arrivino in ordine e che non vadano perduti.
Il servizio, per fornire tutti o parte dei servizi, funziona con *circuito virtuale* o con *datagram*?

- ✔ **Circuito virtuale.** È un modo di far funzionare la spedizione dei pacchetti che funzioni come se fosse una chiamata telefonica instaurando un collegamento permanente tra chiamante e ricevente. Tutti gli scambi avvengono lungo questa connessione. Dal punto di vista fisico non è banale, ma il circuito virtuale instaura un accordo tra mandante e ricevente in maniera univoca prima di far fluire i dati. Ogni pacchetto si porterà dietro un numero detto *numero di identificazione* che lo attribuisce al circuito con medesimo numero identificativo. Negoziare la chiamata vuol dire chiede ricorse specifiche (banda, buffer, cadenza, ecc), allocando così ricorse specifiche. Sul circuito girano solo i pacchetti appartenenti al circuito. Si deve poi mantenere lo *stato* della connessione.
- ✔ **Datagram.** È il modello su cui si basa Internet. Non esistono chiamate di setup e non si instaurano cammini privilegiati, quindi non si perde tempo e non si mantiene lo stato. Non esiste a livello di rete il principio di connessione. Ogni pacchetto si porta dietro l'informazione riguardante il proprio destinatario. Ogni pacchetto è immesso nella rete e lasciato a se stesso.

Best Effort. Modello che fa "il meglio che può". Internet si basa su questo modello. Non c'è banda, non garantisce ordine e arrivo dei pacchetti... perchè non è richiesto.

ATM. Modello a circuito virtuale.

- ✔ **CBR.** Servizio a rate costante e garantito. È un flusso perfetto con frequenza costante. All'uscita non ci sono intoppi. Fornisce tutte le garanzie possibili, non perde niente e non ha nessun disordine interno. Fornisce anche la corretta cadenza dei dati. Non fornisce nessuna gestione di congestione, perchè non la permette a prescindere.
- ✔ **VBR.** Bit rate variabile ma sempre garantito.
- ✔ **ABR.** Bit rate disponibile. Ho però la garanzia che sotto un certo minimo non posso andare (banda minima garantita). Perdo pacchetti perchè in teoria posso finire per saturare il buffer. I pacchetti arrivano in ordine. Nessuna garanzia di timing. Manda info per stabilire se c'è congestione oppure no.

Evoluzione di reti telefoniche, non può andare oltre un certo livello di qualità (in peggio) ed ha

implementato il livello più performante perchè ha livelli di temporizzazione stringenti.

Internet. Modello a datagram.
IP fornisce un servizio elastico senza limiti di deterioramento. Si basa sul fatto che gli host siamo "furbi", considerando che siano in grado di fare molto (mentre Internet di suo fa poco), possono prendersi carico delle attività d recupero dei problemi nella comunicazione di rete: mantengo semplice l'infrastruttura e delego le cose complicate agli host finali che sono utilizzati ad un piccolo numero di utenti mentre l'infrastruttura deve servire moltissime comunicazioni (all'epoca dello sviluppo il tutto era molto costoso).

Funzioni del livello di rete.

- ✔ **Forwarding.** Muove pacchetti dall'entrata all'uscita di un router. Consiste nello smistare pacchetti dalla porta di entrata a quella di uscita (buona per arrivare alla destinazione finale).
- ✔ **Routing.** Determina il percorso da seguire per arrivare a destinazione.
- ✔ **Algoritmi di instradamento.** Devono tenere conto della modifica dello stato ella rete per poter inviare i pacchetti a destinazione attraverso le porte giuste.

Architettura di un router. Un router è composto da parti essenziali, ottimizzati per compiere operazioni di calcolo di strade (protocolli di instradamento) e di forwarding. Consta di porte di ingresso ed uscite, di un oggetto fisico di collegamento (switching fabric e schede) e di un processore che elabora e fa di conto per fornire le info per lo smistamento dei pacchetti in maniera efficiente e veloce.
Porta di ingresso ed uscita dal punto di vista fisico non sono diverse, la porta è un connettore dove il cavo si collega.
Scheda di rete: la più ovvia è quella ethernet. Le porte dei router applicano a livello fisico la filosofia dello spostare la complessità alla periferia della rete. Le porte dei router sono in grado di compiere switching decentralizzato, con la possibilità di memorizzare entro le schede la tabella di forwarding.
Switching fabric: ne esistono di tre tipi, **memoria** (utilizzando un pc con due schede di rete che fa da router, la porta d'accesso è di basso livello), **bus** (bus per le porte, il datagram viene analizzato in ingresso e mandato su un bus verso la porta di uscita, il problema è la contesa del bus) e **crossbar** (più performante, usa il network di interconnessione, sono tanti bus a reticolo che interconnettono tutte le porte).

Porte di uscita. Anche loro bufferizzano quando il fabric switching è molto veloce e sul link in uscita la banda è più bassa del flusso dei dati sulla porta in uscita.
(che noia...)

Il livello di rete di Internet.

- ✔ **Protocollo IP.** Il pacchetto IP si chiama datagram. Consta di tante info, come la *lunghezza dell'header* (che è di dimensioni variabili), *tipo di servizio* (per gestire la qualità del servizio e distinguere le priorità del pacchetto), *lunghezza totale del datagram in byte*, *versione* del protocollo (Ipv4/IPv6), campo di informazione sulla *segmentazione del pacchetto* (è previsto che il datagram cambi di dimensioni, e porta via tempo), *time to live* (numero massimo di passaggi da un router a l'altro di ogni pacchetto), ecc...
  Overhead: quantità di "roba" che si aggiunge al pattuito.
  Indirizzi IP. L'indirizzo è proprio di una scheda di rete. Una parte identifica la sottorete di cui fa parte la macchina e l'altra parte identifica la macchina nella sottorete. Gli indirizzi di rete (o di broadcast) sono indirizzi speciali, da sottrarre al numero totale di indirizzi ottenibili. In genere sono due (uno per identificare tutta la rete con tutti i bit alti dalla parte degli host ed un altro che un host utilizza per identificare se stesso).
  Un host acquisisce un indirizzo IP con due metodi:
  - ✗ **Hard-Cored.** L'indirizzo si trova in un database, allora tale indirizzo è assegnato ad una macchina che collocherà nel database il suo nome accanto all'indirizzo assegnato.
  - ✗ **Dynamic Host Configuration Protocol (DHCP).** Sistema dinamico che permette ad una

macchina configurata di acquisire dinamicamente un indirizzo IP dal server DHCP che ne fornisce a seconda dell'area in cui si trova (e del parametro di tale server).

- ✔ **Protocollo ICMP.** (*Internet Control Message Protocol*) Protocollo di rete che funziona usando il datagram. Viene utilizzato dagli host e dai router per comunicare a livello di rete informazioni che possono tornare utili. È un protocollo molto semplice per controllare gli intoppi gravi. Usato molto in modo invisibile (Traceroute e Ping). Sta tra il protocollo di IP ed il protocollo di trasporto. I messaggi ICMP sono composti da pochissimi campi.
- ✔ **Protocollo di routing.** Gli algoritmi di routing sono molto importanti per l'instradamento dei pacchetti nella rete. È un servizio fornito dal livello di rete. Si tratta di trovare il percorso corretto lungo il quale instradare il pacchetto. L'algoritmo di instradamento calcola un percorso e compila delle tabelle di inoltro da comunicare ai router. I protocolli di routing servono a produrre le tabelle di inoltro che vanno lette per compiere l'operazione di inoltro.
  - ✗ **Astrazione dei grafi.** Il grafo è comporto da N nodi e N link, i nodi sono i router ed i link sono le connessioni tra tutti i possibili router. Assegniamo ad ogni arco un valore che definiamo *costo* del link (che si differenzia a seconda del protocollo e delle esigenze). Più il costo è alto più sono probabili la congestione, malfunzionamento, costi economici reali. È importante riuscire a minimizzare i costi. Esistono due famiglie di classificazione, algoritmi di tipo globale (*algoritmo che quando entra in funzione tiene conto di tutte le info di tutti i router interessati contemporaneamente in modo diretto, conosce lo stato dei link per ogni link*) ed algoritmi di tipo decentralizzato (*algoritmo che sfrutta le info sui link di ogni router che li connettono ai loro vicini e conoscono le info di altri non in maniera assoluta ma circa le conclusioni dei vicini riguardo i percorsi migliori per arrivare agli altri*) dei quali fa parte il tipico algoritmo **distance vector**. In ultimo, possiamo scegliere se calcolare le info in modo *statico* (le info sono inserite a mano a priori e si aggiornano con molta lentezza senza monitorare lo stato corrente della rete) o *dinamico* (gli algoritmi cambiano in fretta grazie ad aggiornamenti periodici basati anche sullo stato della rete in tempo reale).

DHCP. È un indirizzo IP a scadenza, quando scade è possibile rinnovarlo secondo procedura adeguata. Esiste un protocollo che permette di scambiare dei messaggi che consentano ad un host di chiedere e ricevere un indirizzo. Tale protocollo ha una caratteristica comune all'interno di IP: ogni dato viaggia all'interno di un datagram, nel corpo dati c'è un header e delle info che non sono del protocollo superiore ma dello stesso livello di rete. In pratica, è possibile viaggiare "orizzontalmente" all'interno del livello di rete. Non tutte le macchine possono ottenere un indirizzo in DHCP, poiché possiede un numero ristretto di indirizzi.
Come rivolgersi ad una macchina senza indirizzo? Si utilizza l'indirizzo di broadcast: la macchina DHCP manda un messaggio in broadast a *tutte* le macchine esistenti per vedere se ce ne è una che è in grado di rispondere sulla sua porta. L'host manda un messaggio per sapere se esiste un server DHCP, il pacchetto che viene inviato si porta dietro l'informazione della porta a cui rispondere e la posta a cui rivolgersi. Quando il server ed host si trovano, il server offre un indirizzo IP con le sue relative informazioni: quando l'host acconsente alla transizione manda una ricevuta positiva ed il server segna che ha assegnato quel tal indirizzo a quella tal macchina.
Il protocollo fa tutto da solo.

Indirizzi IP e ICANN. Una sottorete si rivolge al proprio provider (livello gerarchico superiore alla sottorete) per ottenere un pool di indirizzi IP. Il provider rilascerà allora una classe di indirizzi IP alla sottorete che ne ha fatto richiesta.

DOMANDA: quali sono gli indirizzi di rete o di broadcast di una sottorete?

Il provider ha un pacchetto di indirizzi che può suddividere per i suoi clienti. Il provider però riceve un pacchetto da qualcuno di più "importante": **ICANN** (*Internet Corporation for Assigned Names and Numbers*) è una istituzione internazionale che assegna nomi ed indirizzi. Funziona in modo gerarchico. I numeri sono allocati e si parla della gestione alta dei DNS, per prevenire controversie funge da tribunale per risolvere le dispute riguardo nomi e numeri di server e domini.

Traceroute e Ping. Come si fa a fare un Traceroute? ICPM può restituire al mittente un

pacchetto che comunica alla macchina la morte di un TTL. Potrei essere interessato a sapere quale è la strada che un pacchetto percorre per raggiungere la destinazione, specialmente quando c'è un intoppo. Tornano info riguardo a quale router ha raggiunto il pacchetto ed in quanto tempo l'ha raggiunto. Traceroute manda un pacchetto a destinazione con TTL uguale a 1, allora spira appena tocca il primo router, tornando le info di tale router. Poi ne manda un con TTL uguale a 2 e così via fino a raggiungere la destinazione. Se il datagram non arriva a destinazione vuol dire che c'è un intoppo, si potrebbe verificare una nuova strada alternativa. Purtroppo è molto rudimentale.

IPv6. Gli indirizzi IPv4 potrebbero finire, perchè 32 bit son pochi. Serve anche un header più snello per un inoltro più veloce. Da molti anni, con IP si mandano messaggi che contengono dati di applicazioni molto diverse tra loro, si potrebbe gestire meglio la *Quality Of Service* (tipologia di contenuto dei pacchetti IPv4). Così si è ridotta la lunghezza dell'header, eliminando qualsiasi servizio di frammentazione del pacchetto (passeranno solo i pacchetti della lunghezza giusta, gli altri verranno adattati dal sorgente). I campi presenti sono *versione*, *priorità*, *identificatore del flusso* (per identificare datagram che contengono dati dello stesso flusso, ma non si sa ancora che cosa è il flusso), *lunghezza del campo* (che è sempre fisso però, quindi fornisce la lunghezza della parte dati), *next header* (campo in cui si dice qual'è il protocollo al quale va consegnato il pacchetto di dati), *hop limit* (TTL), *indirizzo sorgente e destinazione* (128 bit ciascuno).
Mancano il *checksum* (inutile tempo di elaborazione), i campi *opzioni* che però non impedisce di aggiungere delle opzioni al pacchetto che vanno inserite in un altro modo. Esiste un *ICMPv6* al quale è passato il compito di gestire l'assenza del servizio di frammentazione; quando IP scarta un pacchetto troppo lungo lo butta ma informa l'host sorgente tramite ICMP. È meglio gestito l'indirizzo di multicast.
Passare ad IPv6 è traumatico. Usando IPv4 si è provato a risolvere il problema dell'esaurimento degli indirizzi e ci si è riusciti abbastanza bene. Però la gestione del cambiamento di un protocollo non è banale, se non è proprio necessario non si fa. Non è possibile fermare tutto il mondo per effettuare il cambio (danno economico) e poi le cose mai funzionano come dovrebbero ed il tempo di stop si prolungherebbe. Bisogna farlo pian piano, lavorando con un protocollo misto:

- ✔ **Tunneling.** Quando si è di fronte ad un IPv6 lo si inserisce in un pacchetto IPv4 con tutto l'header e lo spedisce. Ciò vuol dire che devono esistere tutti e due i protocolli in funzione. Fare *tunneling* vuol dire che un pacchetto IPv6 che deve passare su macchine IPv4 viaggia in maniera logica tra macchine IPv6 in maniera praticamente diretta vedendo IPv4 come un livello basso di trasmissione pura e semplice. Dal punto di vista fisico il pacchetto IPv6 viene inserito in un pacchetto IPv4 come parte di dati.
- ✔ **Dual Stack.** Nel router, il pacchetto IPv6 passa entrambe le pile protocollari inserito in un IPv4 con scarto dell'intero header dell'IPv6. C'è perdita di informazione e si modifica il pacchetto originale.

Algoritmo di tipo Link-State. Non ci sono motivi perchè un tipo sia dominante rispetto all'altro, spesso esistono anche ibridi di diversi algoritmi magari non funzionali se presi singolarmente.
L'algoritmo base di tipo Link-State è l'**algoritmo di Dijkstra**. Tale algoritmo presuppone che si conoscano tutti i nodi, dove si trovano e come si collegano tra loro. Allora ogni nodo manda agli altri in broadcast le info riguardanti i link che conoscono direttamente, per informarli, questo permette di calcolare i percorsi che minimizzano i costi indifferentemente dal funzionamento errato di qualche router. Allora si costruiscono le tabelle di inoltro per quei nodi. L'algoritmo è di tipo iterativo, dopo tot volte conosce il miglior costo per ogni destinazione (ci sono tante iterazioni per quanti sono i nodi). Dopo i conti effettuati dai router si crea una ramificazione nuova ed ottimale, la quale viene memorizzata in una tabella per ottimizzare gli scambi dei pacchetti.
Perchè questa cosa funzioni devono essere mandati tantissimi pacchetti. È tollerabile se i router non sono moltissimi e se sono abbastanza veloci. Col fatto che i link sono dinamici il cambio dei costi fanno oscillare i percorsi, portando un pacchetto a transitare magari anche numerose volte sullo stesso router degradando moltissimo il tempo di transito ed il percorso del pacchetto (*oscillazione dei pacchetti*). Per risolvere il problema è necessario dare un numero massimo di hop per non appesantire la rete... certo non è una soluzione che porta alla consegna corretta del pacchetto.

Distance Vector Algorithm. Il costo per arrivare a destinazione è il minimo valore che si calcola sommando i costi delle tratte che interconnettono il router destinazione a tutti i vicini più il costo che è calcolato come miglior costo per arrivare a destinazione.

- ✔ **Esempio di Bellman-Ford.** Se prendiamo il router di partenza, deve andare a sommare i migliori costi dei suoi router vicini più i costi che conosce per andare a destinazione. Nella tabella di inoltro serve l'informazione per arrivare dalla partenza all'arrivo, ovvero la porta opportuna assieme al costo relativo della transizione.

Si utilizzando le seguenti definizioni: stima del miglior costo, stima dei link fisici di un router con i suoi vicini; esiste un vettore delle distanze con tutti i costi per arrivare ovunque (appunto, il *distance vector*). Quando cambia qualcosa, poichè l'algoritmo è dinamico, si aggiorna anche il vettore. Ogni router ha un vettore delle distanze e può consultare quello dei suoi vicini.
Ogni tanto ogni nodo manda ai suoi vicini fisicamente collegati il proprio vettore, in maniera asincrona (tutti i nodi mandano quando vogliono e non tutti insieme). Se non ci sono variazioni non viene mandato nessun aggiornamento.
È un algoritmo iterativo, ogni volta che c'è un nodo va rifatto il calcolo; è asincrono perchè i messaggi vengono mandati solo se un vicino ha il vettore che cambia e ci si aspetta che i messaggi che circolano siano drasticamente di meno. È distribuito perchè in questo modo il risultato di ogni conto di ogni router dipende anche non solo da quello che ha calcolato lui ma anche dai calcoli degli altri router della rete.
Su ogni singolo nodo c'è attesa che arrivino altre informazioni (che non possiede) perchè son cambiati altri distance vector, vengono rifatte allora tutte le stime: se il proprio vettore cambia viene notificato ai vicini altrimenti il sistema torna in attesa.
Cosa succede quando cambiano i costi dei link?? Il nodo che si accorge del cambiamento prende atto della cosa è si ricalcola i propri corsi, se il risultato è modificato avvisa gli altri router vicini. Quando i link migliorano la convergenza ad un nuovo risultato è abbastanza veloce. Quando i link peggiorano le notizie del deterioramento viaggiano molto lentamente. **Ritorno avvelenato**: l'algoritmo funziona quasi sempre se ogni volta che un router non manda l'info dei costi giusta ma falsa (essendo comunque questa info indispensabile). Ogni volta che un router comunica propri costi ai vicini invece di mettere un costo vero notifica un costo infinito.

Confronto. Considerando la complessità del messaggio, il LS manda un numero di messaggi nell'ordine di grandezza del numero di nodi per il numero di link; il DV manda messaggi solo ai suoi vicini noti. I tempi di convergenza sono diversi: il LS è sincrono con tempo costante, nel DV il tempo varia perchè dipende da cosa è cambiato e solo se è cambiato. Il LS teme l'oscillazione, il DV teme il conteggio all'infinito (avanti e indietro di un pacchetto da un router all'altro). Il LS è più robusto perchè fa conti diretti con info di prima mano (se un nodo manda info sbagliate non si propagano e si correggono da sole col tempo), il DV si deve fidare e la possibilità che l'errore di un router si propaghi e deteriori i conti è alta. Il LS per ottimizzare lo smaltimento dei pacchetti ha bisogno di un'altissima quantità di dati occupando la banda.

Routing Gerarchico. *Quanti router abbiamo nella rete?* Se consideriamo tutti i router esistenti al mondo, utilizzando un LS avremmo una quantità di dati scambiati e di tempo sarebbero enormi. Anche il DV è deprecabile, perchè si porterebbe dietro tutte le info di tutti i router per arrivare a tutti i router. Il problema della scala è un problema insormontabile se non si applicano delle modifiche. Per ultimo, ogni router appartiene ad una rete di competenza diretta di una azienda o di uno stato (comunque di una organizzazione specifica). Se per i domini c'è una autonomia amministrativa, non esiste lo stesso per i router. Per questi motivi è necessario aggregare i router tra loro simili in regioni dette **Automonous Systems**, allora queste regioni useranno gli stessi protocolli detti **protocolli intra-AS**. Tra regioni diverse servono dei "ponti" con dei router, detti **gateway router**, all'interno dei quali esiste un protocollo che mette in accordo protocolli diversi tra regioni diverse (**protocollo inter-AS**).

Protocollo Inter-AS. Se avessimo più di un algoritmo di intercomunicazione sarebbe un guaio, perchè *questo unico algoritmo* deve essere un collante tra i protocolli interni che sono diversi tra loro e deve essere in grado di farli comunicare tra loro. Il protocollo in questione deve fare in modo di prendere il datagram prodotto da un dato AS (che il router deve instradare al router gateway giusto per esportare il datagram in un altro AS, visto che possono essere diversi i router gateway) e

tradurlo per il protocollo dell'AS di destinazione. Il protocollo inter-AS stabilisce il miglior percorso per raggiungere l'AS di destinazione, annientando i link interni di ogni AS (e considerandoli come enormi scatole). È detto **sistema della patata bollente**, poichè il datagram deve essere mandato a destinazione il più velocemente possibile.

Protocolli Intra-AS. Ne esistono diversi:

- ✔ **RIP.** *Routing Information Protocol:* è un DV, incluso nelle distribuzioni Linux dal 1982. Molto semplice, dovuto alla metrica delle distanze (meccanismo di valutazione del costo dei link): il costo è uguale al numero di hop, comoda per fare dei conti ma rude per l'utilizzo pratico del sistema. Il massimo numero di salti è fisso (si chiama diametro della rete) ed è pari a 15. Quando cambiano i costi (che comunque cambiano molto raramente) i vettori si scambiano tra i vicini ogni 30s attraverso un messaggio detto *Response Message*. Tale messaggio è inviato ad un massimo di 25 router vicini.
- ✔ **OSPF.** *Open Shortest Path First*: è un protocollo basato su LS che si basa sulla segnalazione agli altri pacchetti, allora ogni nodo deve avere una info topologica di tutti gli altri router riguardo anche a cosa loro sono collegati. I pacchetti hanno una riga per ogni router conoscitore dello stato del link a lui direttamente collegati, lo manda a tutti e non solo a quelli vicini. Si utilizza direttamente un datagram IP. Rispetto a RIP è molto avanzato: considera sistemi di sicurezza; esistono diramazioni multi-percorso; sono previsti possibili argomenti per valutare i costi e la metrica è più complessa (si tiene conto del tipo di link, ecc) e si basa anche sulla disponibilità della banda momentanea, congestione ecc; è possibile utilizzare un supporto per il multicast (MOSPF). È intrinsecamente gerarchico, parlando già di protocolli di instradamento che funzionano all'interno di un dato AS. Questo protocollo entro un AS è in grado di gestire una struttura gerarchica: se in un dominio grande usiamo questo sistema risparmiamo tempo nell'instradamento dei pacchetti. All'interno dell'area i pacchetti viaggiano con un modello base del LS, se i router sono moltissimi è utile suddividerli in sottoaree: all'intero di un AS si stabiliscono microaree con router di confine (ma non gateway di confine delle grandi AS), allora in questa configurazione è ammesso l'utilizzo del protocollo OSPF. Solo all'interno dell'area girano i pacchetti locali ed ogni nodo ha una info completa della topologia dell'area e non di tutto l'AS.
- ✔ **IGRP.** *Interior Gateway Routing Protocol* (diffusissimo, proprietario della CISCO). Ibrido tra DV e LS.

Protocolli Inter-AS. Sono:

- ✔ **BGP.** *Border Gateway Protocol.* È un protocollo che collega due AS differenti. Per stabilire le strade migliori utilizza info di tipo tecnologico (banda, percorsi, ecc) sia info di policy (ovvero regole, non solo politiche istituzionali ma di *politica interna*, e non sempre riguardano l'ottimizzazione del percorso). Permettono alla rete di una istituzione di rispettare l'istituzione e non la topografia della rete. Un AS dichiara la propria esistenza al mondo, oppure no: esiste ma non sempre è utilizzabile, dipende se lo permette il protocollo.
  I peer BGP pari-livello si scambiano info su connessioni TCP semi-permanenti (a lunga durata, dette *BGP-Session*). TCP è solo unicast allora tutte le connessioni di questo protocollo sono fatte a coppie. Le sessioni sono esterne quando i router sono di confine tra due AS differenti. Anche all'interno dell'AS però le tabelle di inoltro si basano su info calcolate dal protocollo inter-AS per comunicare come mandare pacchetti tra AS diversi.
  Quando mandiamo un prefisso con attributo (contengono gli identificatori degli AS attraverso i quali il prefisso della connessione può viaggiare) definiamo una **strada**. Il successivo AS tramite passare è un'altra info, che ci permette di dire che il prossimo passaggio lo si fa attraverso quell'AS. Quando il gateway riceve l'info può decidere se accettare o rifiutare il pacchetto.

Network Address Translation. Soluzione alla non-applicazione dell'IPv6. Permette di far comunicare sulla rete anche i nodi nascosti in reti interne. Nella rete locale le macchine si conoscono tra di loro e non c'è nessun problema; nel momento in cui devono farsi vedere dal

mondo esterno lavorare con i nodi nascosti non è più possibile. Usiamo un proxy web (ok.. ma non basta). C'è necessità che una macchina interna possa ricevere la risposta della domanda mandata preventivamente, ma è difficile perchè non si sa come poter raggiungere quella macchina (che sulla rete generale è nascosta). Il NAT non è esattamente un protocollo di funzionamento di rete, ma una specie di gioco sporco che vuole risolvere un problema in maniera temporanea fino all'avvento del IPv6. Il router che lavora con in NAT deve:

- ✔ lavorare sui datagram in uscita ed in ingresso;
- ✔ trasformare l'indirizzo interno più il numero di porta della macchina della rete nascosta in un indirizzo esterno (con indirizzo ufficiale del NAT ed il numero di porta del router, che è una porta TCP/UDP);
- ✔ il router deve costruire una tabella (NAT Translation Table) contenente la lista dei numeri di porta delle macchine interne.

Il router manda allora la richiesta ed attende la risposta, quando arriva prende il pacchetto e sostituisce il suo indirizzo IP con l'indirizzo IP ed il numero di porta di chi aveva fatto la domanda originale all'interno della rete interna.
Si utilizza un numero di porta a 16bit e possiamo avere al più 60.000 connessioni che passano attraverso lo stesso NAT (servendo domini locali abbastanza grandi). Dal punto di vista architetturale sono violate le politiche end-to-end ed il modello a strati della pila protocollare.
Quando arriverà IPv6 non se ne parla più.

Multicast. Regola che non è per forza implementata. Fare multicast implica che una macchina manda lo stesso pacchetto a molte altre macchine e non solo ad una sola macchina in attesa. Il multicast è una versione complicata del broadcast (che invece è una connessione uno a molti selezionati come una parte completa di un dominio). Il multicast permette un indirizzo di multicast che racchiude un gruppo di destinatati che hanno un proprio indirizzo (tipo lista abbonati), mentre il broadcast consente di mandare ad un gruppo, senza indirizzi esplicitati.
Esistono due modi:

- ✔ **Modo implicito.** Non è un vero multicast, ma un automatismo che instaura N connessioni unicast. Per evitare i mandare in giro pacchetti inutili, si spedisce un singolo datagram per ogni tratta e quando va “splittato” lo si moltiplica.
- ✔ **Modo esplicito.** Si manda un singolo pacchetto unicast agli interlocutori che hanno un software applicativo che capisce chi sono i destinatari e fa partire comunicazioni unicast solo per le destinazioni che necessitano quel datagram.

Broadcast. Abbiamo gli stessi problemi del multicast. Duplicare i datagram a livello di partenza è inefficiente perchè crea troppo traffico. Con il sistema di **flooding** potremmo pensare di mandare a tutti i vicini (a tutti i link) e poi ogni router manda ai suoi vicini e così via, anche se purtroppo esistono delle maglie tra i router che possono ciclare il pacchetto all'infinito, o alla fonte. Con il **flooding controllato** si può verificare se un pacchetto è già passato per quel router. Con l'**albero di copertura**, invece, i pacchetti non sono mai ridondanti; prima di mandare il broadcast dei pacchetti si deve costruire l'albero di copertura che permette di raggiungere tutti i nodi ma attraverso una strada sola. A seconda di dove parte il pacchetto si creano alberi diversi. Si parte da un nodo centrale, quello a cui tutti gli altri mandano messaggi unicast (strada a minor costo) e questi messaggi sono inoltrati da un nodo all'altro fino a destinazione. Tutti i router devono appartenere all'albero.

Indirizzamento multicast. Si scelgono macchine con indirizzi indipendenti tra loro. Utilizzo IGMP (Internet Group Management Protocol) per far comunicare host collegati a router per la comunicazione multicast. Serve anche un protocollo di instradamento multicast. Senza di loro si lavora senza un multicast esplicito.
Anche in multicast serve trovare un albero di copertura per l'instradamento dei datagram. Se un router ha tra i propri nodi dei membri di un gruppo multicast dovrà essere in grado di costruire l'albero di copertura. Non è necessario che tutti i router appartengano all'albero (se ce ne sono che non servono al gruppo multicast). Si parla di *alberi condivisi* quando più gruppi multicast si intersecano.

Lezione 5 ~ Livello Data-Link

Introduzione. Il livello di data-link è implementato sempre sulle schede di rete. Scendendo nella pila protocollare si raggiunge sempre più lo strato hardware abbandonando man mano quello software. Che cosa deve fare il data-link?

- ✔ il minor numero di errori possibili che arrivano sul canale (trovare e correggere l'eventuale errore);
- ✔ siccome molti pacchetti sono condivisi, ci vuole qualche cosa che permetta di regolare la condivisione del canale fisico;
- ✔ passando da IP a data-link togliamo dal pacchetto l'intestazione IP trasformandoli in *frame* senza indirizzo. Se i link sono end-to-end non c'è problema, che sorge se invece i collegamenti sono più ampi. Va gestito il sistema di indirizzamento di questi frame, tramite i MAC address;
- ✔ trasferimento affidabile dei dati, senza problemi di congestione (visto che è un problema della rete e non della comunicazione di due macchine fisicamente interconnesse), ma di controllo di flusso;
- ✔ gestione di protocolli che vadano d'accordo con l'hardware che si utilizza.

Ci si interessa ai singoli link, divisi in categorie (wired e wireless) entro le quali cadono tutte le tecnologie LAN e WAN. Il data-link ha la responsabilità di spedire dati in maniera corretta sul link condiviso.
I servizi del data-link sono:

- ✔ **Framing.** Si deve prendere il datagram ed incapsularlo nel frame, si aggiunge una intestazione ed un segnale di fine messaggio.
- ✔ Se i link sono mezzi condivisi, vogliamo accederci in sicurezza.
- ✔ Utilizzo dell'indirizzo MAC.
- ✔ Consegna affidabile tra nodi adiacenti, che non dovrebbero arrivare sfalsati (wired), non è vero in un sistema wireless.
- ✔ Mantenimento il controllo di flusso come al livello di trasporto.
- ✔ Correzione degli errori di trasmissioni.
- ✔ Half-duplex e full-duplex.

La comunicazione prende il datagram e lo incapsula nel frame.

Error Detection. *In che modo possiamo accorgerci di un errore sui bit?* Prima si parlava solo di checksum, ma non è il sistema più potente che abbiamo a disposizione. Si deve fare un rilevamento più affidabile (e nessun sistema è affidabile al 100%), si utilizzano informazioni ridondanti che dovremmo ridondare almeno tre volte (ma tre volte è troppo costoso). Utilizzando le parti ridondanti si equiparano i dati originali con quelli ridondanti e si controlla la presenza di deterioramenti.

- ✔ **Controllo di ridondanza ciclica.** Ovvero l'utilizzo del checksum. Implementato in tutte le tecnologie più utilizzate.

Multiple Access Links and Protocols. Abbiamo due tipi di link, uno punto-a-punto e quello broadcast. Il mezzo può essere cablato o meno, ma comunque condiviso. In questo sistema condiviso possono essere immessi nella rete due segnali contemporaneamente, e possono fare interferenza perdendo il significato di entrambi (= Collisione). Servono protocolli che devono basarsi su un algoritmo che non può essere implementato su una singola macchina; il canale stesso non deve provocare la collisione.
Idealmente, avendo un canale di una certa banda se lo volessi usare in maniera condivisa, quando un nodo trasmette lo fa ad una certa intensità ma quando vogliono trasmettere più nodi suddividono la banda tra loro in parti uguali. Non serve sincronia, ma non è banale. Il problema non è dividerlo in parti uguali, ma è usarlo tutto quando non ci sono contendenti.

- ✔ **Partizionamento del canale.** Suddivido il canale oppure il tempo in parti uguali ed ogni segnale usa una di queste partizioni. Quando il canale è libero però non è possibile usare tutto il canale nella trasmissione.
  - ✗ *TDMA.* Ogni messaggio ha diritto ad uno slot di tempo.

- ✗ *FDMA.* Come TDMA, solo che è il canale in frequenza ad essere suddiviso, e non il tempo. Ogni messaggio utilizza la propria banda di frequenza.

✔ **Accesso casuale.** Il canale non viene diviso a priori e si usa una stima probabilistica. Le collisioni avvengono sicuramente. Il primo che arriva ipotizza di avere tutta la banda libera per tutto il tempo che gli serve.

- ✗ *Slotted ALOHA.* Ho frame tutti della stessa dimensione. Partono tutti allo stesso momento. I nodi possono cominciare la trasmissione solo quando parte la trasmissione temporale e non quando vogliono loro. Se due nodi vogliono trasmettere nello stesso slot c'è collisione e viene riconosciuta (i pacchetti sono corrotti e devono essere ritrasmessi in uno slot successivo scelto in un tempo random probabilistico tanto ampio quanto è grande la probabilità di collisione). Serve la sincronizzazione di macchine diverse, così difficile da ottenere.
- ✗ *ALOHA puro.* Se mandiamo frame fissi può capitare che i frame si sovrappongano con quelli precedenti, se i frame sono mandati a caso senza nessun controllo. Se le code si sovrappongono le collisioni sono sempre più probabili. È ancora peggio del sistema degli slot temporali.
- ✗ *CSMA.* Detto anche **Carrier Sense Multiple Access**, sulla scheda di rete esiste un dispositivo puramente hardware. Il nodo che vuole trasmettere lo fa subito, però prima ascolta il canale. Ascoltando si può rendere conto se il canale è già occupato, allora si rimanda la trasmissione di un tempo random. Non si eliminano del tutto le collisioni, dovute a tempistica ed a dimensioni di frame.
- ✗ *CSMA/CD (Collision Detection).* Come CSMA, ma se avvengono delle collisioni la trasmissione salta (tanto ormai i dati sono già corrotti). Nelle reti cablate riduce al minimo le collisioni, pur non eliminandole del tutto. Nel wireless è più difficile, a seconda di cosa c'è attorno all'access point.

✔ **Turnazione.** Si fa un po' per uno. Chi ne ha bisogno usa la banda, poi quando non serve più la rilascia per fare in modo che la prenda chi abbia bisogno.

- ✗ *Token Ring.* Suddivideva il canale in maniera efficiente e corretta. L'anello chiudeva tra loro un tot di macchine. Era l'idea di base del protocollo di data-link della IBM. Il *token* era un messaggio mandato da una macchina all'altra e quando lo si riceveva la macchina poteva mandare il suo frame. Una volta finita rilasciava il token per la macchina successiva. Se le macchine sono tante l'unica macchina che trasmette deve attendere che tutte le altre mandassero i loro frame per poter trasmettere ancora. Se una macchina saltava, si perdeva il token. Bastava staccare il cavo ed aprire l'anello per mandare tutto a puttane. Inefficiente proprio a basso carico.
- ✗ *Polling.* Sistema master-slave che controlla le richieste e passa il controllo e l'accesso alla rete a chi ha fatto la richiesta per primo. Funzionava in periodi in cui i device erano stupidi e primitivi e non si voleva sovraccaricarli.

Bisogna essere equi (per non dare vantaggio a nessuno), semplici e decentralizzati (sennò serve un coordinatore esterno).

MAC Address end ARP. Gli indirizzi IP sono a 32 bit strettamente legati alla sottorete ed al dominio all'interno del quale stanno le macchine (indirizzano a destinazione i pacchetti). Gli indirizzi MAC (o LAN, o Ethernet) hanno la stessa funzione ma tenendo conto che è fatto per connette macchine già collegate fisicamente l'una all'altra. Sono indirizzi a 48 bit che non hanno struttura gerarchica e sono virtualmente *hardcoded* sulle schede (formalmente ogni interfaccia di rete ha un MAC address con indirizzo immodificabile). Ogni macchina ha una o più schede, ognuna delle quali ha un indirizzo fisico indipendente (i numeri esadecimali non dipendono dalla rete in cui si trovano). Esistono gli indirizzi di broadcast, concettualmente uguale allo stesso di IP. Gli indirizzi MAC sono tradotti in esadecimale e non in decimale come IP.
Gli indirizzi sono associati alle schede da un'associazione esterna, allora ogni costruttore possiede un pacchetto di indirizzi che lui stesso assegna ad ogni scheda. Devono essere unici (e nessun MAC address deve essere pubblicato). Anche se si può incorrere in esaurimento di indirizzi, di solito si riutilizzano i vecchi indirizzi dismessi. Non esiste nessuna gerarchia tra indirizzi, permettendo così la massima portabilità.

Ogni indirizzo IP ha un MAC di corrispondenza (che appartiene alla sua scheda di rete). All'interno di ogni host esistono **tabelle ARP** *(Address Resolution Protocol)*, versione semplice a livello data-link di quello che può essere il DNS. Queste tabelle associano indirizzi IP ai corrispettivi MAC con un TTL (tipicamente basso) che indica la durata di tale connubio, dopotutto è necessario che le attribuzioni IP-MAC siano velocemente revocabili. Tutto è dinamico e viene memorizzato in maniera completamente volatile.
Nel *protocollo ARP*, un frame viene mandato in broadcast a tutte le macchine, alla ricerca della macchina giusta che, quando riceve il frame broadcast risponde dicendo che il messaggio mandato a tutti è indirizzato a lei. È una cosa che succede solo quando non esiste corrispondenza nella tabella ARP tra IP e MAC.

Ethernet. Per interconnettere le macchine, a livello di protocollo, si usa la tecnologia Ethernet. Si tratta di una tecnologia con costi bassissimi (economico e funzionale), semplice più di altre tecnologie coeve. È in grado di mantenere lo standard di economicità e semplicità pur evolvendo con le richieste di aumento velocità e banda. La forza dell'Ethernet è un protocollo che si adatta perfettamente a tutti i cambiamenti.
Si è partiti con un cavo coassiale al quale si attaccavano altri cavi tramite transciever (e si "infilzavano" nel cavo coassiale), allora tutti i nodi condividono lo stesso mezzo con segnale che gira sullo stesso cavo utilizzando così lo stesso dominio di collisione. Adesso ogni macchina è collegata a stella ad uno switch, ogni "raggio" isola il proprio dominio di collisione.
Il frame Ethernet è composto da *header*, *trailer* (per controllo ridondanza ciclica), *indirizzo destinazione e sorgente*, campo *tipo* e frame incapsulato. Il preambolo (header) è il primo che arriva. È permessa la sincronizzazione con controllo del flusso: se le due schede che parlano hanno versioni del protocollo differenti, lo switch riceve i bit dal sender e li interpreta per il receiver basandosi sulle info contenute nel preambolo.
Ethernet non è connection-oriented (quindi connection-less), è inaffidabile (i pacchetti che non gli sconfinferano vengono cestinati). Il protocollo di Ethernet per accedere al mezzo condiviso è CSMA/CD: dal punto di vista implementativo, l'algoritmo prende il datagram dal livello IP, fa il Carrier Sense; se la portante è idle trasmette, se invece è busy aspetta fino alla liberazione. Se quando parte il segnale rimane libero per tutto il tempo della trasmissione il pacchetto arriva, sennò la trasmissione abortisce e manda un segnale di collisione. Per ritrasmettere il pacchetto si sceglie un ritardo scelto a caso proporzionale alla probabilità di collisioni.

Hubs. Comunicare con Ethernet è comodissimo. Ma cosa c'è "dentro" alle reti Ethernet? C'è un sistema fisico di interconnessione.

- ✔ **Cavo condiviso.** Ha i suoi difetti, se si stacca il cavo va tutto a puttane. Per ovviare basta usare un mezzo più continuo.
- ✔ **Hub.** Concentratore di cavo. È un oggetto fisico che contrariamente al cavo steso mantiene la continuità ed eventualmente "ripeta" il segnale amplificandolo per coprire distanze maggiori di un singolo cavo. Quello che succede nell'hub, però, è lo stesso che succede dentro un singolo cavo. Non separa i domini di collisione. Non c'è buffer per frame accodati, non sono schede intelligenti ma solo connessioni fisiche.

Lo si usa solo in casi di reti semplici e contenute.
Switch. Sono device a tutti gli effetti che implementano i protocolli di data-link. Fanno store-and-forward dei pacchetti (con buffer dedicati), interpreta selettivamente il destinatario spedendo il pacchetto sulla porta giusta per la consegna. Non ha bisogno di essere configurato. Autorizza le trasmissione simultanee: sono possibili connessioni simultanee tra diverse coppie di macchine che passano tutte per lo stesso switch. Ogni link è full-duplex.
Uno switch sa a quale porta inviare selettivamente un frame in arrivo consultando la sua **switch table**. Simile ad una tabella di inoltro di un router, mantengono la coppia tra indirizzo MAC di destinazione e numero della porta. Imparano le coppie leggendo man mano i dati di provenienza di ogni pacchetto, memorizzando interfacce e TTL. Le tabelle hanno una breve vita media.
Hanno una topologia virtuale a stella. In strutture grandi non si possono usare tipologie pure a stella, allora si interconnettono a cascata gli switch (stelle a cascata). A livello istituzionale le macchine sono interconnesse a cascata per motivi tecnici ma anche in modo gerarchico.

Differenze tra switch e router. Entrambi sono strumenti che instradano i pacchetti, i router a

livello di rete guardando indirizzi IP, gli switch con gli indirizzi MAC. I router hanno tabelle di inoltro costruiti da algoritmi di instradamento, gli switch hanno tabelle che imparano le strade dei pacchetti. La differenza sostanziale è che gli switch lo fanno autonomamente, i router vanno gestiti ed amministrati. Su entrambi però sono implementabili policy di sicurezza. Lo **switch** utilizza un solo protocollo, i **bridge** sono apparecchiature con poche porte che fanno da *ponte* tra protocolli diversi (entrambi separano i domini di collisione).

Protocollo punto a punto. Il data-link punto a punto ha un sender, un receiver ed un "qualcosa" in mezzo, non usa un link condiviso. È un protocollo di data-link che utilizza i frame (con regole ad hoc di incapsulamento); il livello di rete non è necessariamente IP ma anche qualsiasi (la parte fisica è poco legata ad uno specifico protocollo di rete); la connessione è viva (ed a lunga scadenza) e va verificata; deve essere possibile la negoziazione del livello di rete (devono essere in grado di configurare gli eventuali indirizzo di rete, ma solo all'inizio).
Non viene richiesta controllo di flusso e correzione degli errori, non serve nessun controllo di ordine di arrivo pacchetti (sono in un cavo!!) e non necessitano di supporto multipunto. Se servono, ci pensano i livelli superiori.

- ✔ **Byte Stuffing.** Come distinguere un byte di flag da un byte non di flag? Se nel corpo dei dati ho un byte uguale a quello di flag, lo duplico. Quando il receiver trova un byte singolo capisce che è quello di flag, se ne trova uno doppio scarta il duplicato e capisce che non è di flag.

ATM. Il protocollo PPP gestisce la trasmissione di dati a livello di link tra link geografici. ATM non è un protocollo, ma una intera infrastruttura di rete indipendente. I router ATM si chiamano switch, ma non sono la stessa cosa degli switch Ethernet. Le linee ATM forniscono la possibilità di connettere entità che stanno in locazioni diverse, utilizzandolo come protocollo di data-link mostrandolo tale alla nostra architettura. **Asyncronous Transfer Mode**, creata dalle compagnie telefoniche per digitalizzare le comunicazioni telefoniche, fornisce infrastruttura di banda larga. Si commutano pacchetti anche se la filosofia è quello del circuito virtuale (i pacchetti sono *celle*). L'architettura ATM è a tre layer (minimi obbligatori), fatta un po' come TCP/IP.

- ✔ **AAL (ATM Adaptation Layer).** Adatta l'architettura ATM all'applicazione sovrastante (nel nostro caso, al protocollo IP). È presente solo negli end-node (anche router), il router deve avere la capacità di gestire questo layer ATM e non tutti i router possono farlo (ne serve uno apposta). Prende il pacchetto, lo mette nel pacchetto ATM, lo frammenta e diventa un tot di *celle*. AAL5 adatta i pacchetti IP (è quello che serve a noi), ma ce ne sono diversi. È diviso in:
  - ✗ *Sottolayer di convergenza.* Prende il datagram e lo infila in un pacchetto che non è già di 53byte ma è costituito dal datagram più l'header ed il trailer del sublayer di convergenza. Il pacchetto ingloba il datagram di partenza.
  - ✗ *Sottolayer di divisione ed assemblaggio.* Spezza il pacchetto di convergenza in blocchi da 53byte ognuno con header e trailer personalizzati.
- ✔ **ATM.** Paragonato al protocollo IP del livello di rete. In ATM il livello di rete sta sotto al data-link. Effettua il trasporto tramite circuito virtuale. Ogni cella si muove su circuito virtuale, circuito che fa settato all'inizio tramite chiamate (concordando caratteristiche di connessione) e che va chiuso con chiamate finali che concordano la chiusura del circuito. Ogni pacchetto si porta dietro un *identificatore di circuito virtuale*, legato al percorso stabilito dopo la chiamata di setup. Comunicazione totalmente state-full (ogni connessione ed ogni passaggio è registrato nello switch). È possibile richiedere banda e buffer a sufficienza per comunicazioni fatte ad hoc. Si dividono in:
  - ✗ **PVC (Permanent Virtual Circuit).** Circuiti permanenti che durano finchè si paga il servizio, passano per uno o più switch e forniscono il link virtuale (equivalente ad un cavo fisico e reale) tra due nodi.
  - ✗ **SVC (Switched Virtual Circuit).** Circuiti che si usano quando si gestiscono reti ATM a livello locale, ogni chiamata tra macchine costruisce il circuito virtuale e via di seguito. È una cosa dinamica, ondemand.
- ✔ **Fisico.** Cavi e cavetti. Ha due sublayer:
  - ✗ TCS (Transmission Convergense Sublayer).

- ✗ PMD (Phisical Medium Dependent). Come SONET/SDH (bande larghe usate dalla telefonia implementate su rame ma soprattutto su fibra).

ATM viene vista allora come una tecnologia di data-link (anche se è di rete). Il traffico di datagram a cui siamo abituati è imponente, ATM è sconveniente. Se vogliamo costruire una rete tutta ATM scala linearmente con il numero di macchine. C'è un grande tempo di latenza per connessioni molto brevi.

IP su ATM. Gli switch ATM sono router. Quello sta sotto i router è una sola rete (ATM), le reti Ethernet non hanno problemi di differenziazione tra indirizzamento IP o ATM. Quando un router IP si interfaccia ad una rete ATM deve implementare l'architettura sotto IP ed anche quella ATM (quindi sul router c'è un doppio layer parallelo IP e ATM). Quando arriva tutto a destinazione, sembrerà non essere cambiato niente.

Lezione 6 ~ Wireless

Introduzione. In un cavo c'è un segnale fisico che viaggia in maniera guidata, nel wireless no e non si può schermare il segnale: c'è un diverso problema di gestione di un segnale fisicamente differente. La differenza nasce dal livello fisico. La grande diffusione delle reti wireless è una cosa recente, che ha introdotto nuovi problemi prima assenti. Ii vantaggi sono di non avere cavi vaganti e di poter essere "mobili" mantenendo comunque la connessione (nonostante il raggio d'azione di un access point diminuisca all'aumentare della distanza).

Elementi di una rete wireless. Esistono *stazioni base (access point)*, che permettono di connettere tra loro in maniera centralizzata i vari *host* dotati di una scheda di rete wireless. Possono essere completamente indipendenti oppure sovrapposti (ed in questo caso un host passa da un raggio d'azione all'altro in maniera totalmente pulita). Il protocollo che regola la comunicazione wireless è *802.11(a,b,g,n)*. I *link* esistono ma non sono fisici, il segnale dopotutto è un onda che si espande. Siamo automaticamente in una situazione di accesso multiplo al mezzo, che è per definizione condiviso (a meno di una situazione estrema) quindi c'è bisogno di una gestione della condivisione. Gli *access point* sono collegati con cavo al backbone, e loro stessi possono costituire un backbone.
Si possono creare reti wireless senza access point, dette *reti ad hoc* composte solo a host che mettono a disposizione la loro connettività per amplificare il raggio d'azione della rete. È possibile configurare degnamente la scheda di rete in modo da farla funzionare come gateway per altri host (allora riceve il segnale e lo ritrasmette ai suoi vicini). Per far ciò bisogna che gli host siano abbastanza vicini per sovrapporre i propri raggi d'azione.

Caratteristiche delle reti wireless. La diminuzione dell'intensità del segnale viene eliminata dando delle lunghezze standard massime oltre le quali il segnale sarebbe troppo basso, così succede per i cavi. Con i segnali wireless non si sa se saranno certamente più deboli dopo una certa distanza, perchè si ignorano gli eventuali ostacoli frapposti tra access point ed host. Esistono poi potenziali interferenze del segnale elettromagnetico con altre onde (con i cavi bastava schermarli, le onde non possono essere schermate). In wireless il segnale viaggia nello spazio tridimensionale, allora il segnale non arriva solo a destinazione, ma anche altrove propagandosi dappertutto fino ad esaurimento del segnale. Il segnale allora può anche riflettersi, come un raggio di luce.

CDMA (Code Division Multiple Access). Si utilizza un codice assegnato ad ogni utente (dalla componente numerica non infinita) che permette una sorta di partizionamento della frequenza wireless. Ogni host possiede il suo codice privato ed una chiave (=sequenza di chipping) con la quale decodificare il messaggio (codificato moltiplicando i dati originali per il codice di codifica). I segnali allora possono viaggiare senza danneggiarsi. Funziona se non si utilizza il wireless come il wired (trattando le frequenze e le codifiche come in wired).

Accesso multiplo. Bisogna evitare le collisioni quando più nodi trasmettono, perchè sono difficili da identificare. Allora si ascolta in segnale prima di trasmettere (per evitare di entrare nella rete "a piedi uniti" ed incappare in un altra trasmissione). Se non abbiamo collisioni potrebbe

essere difficile riceve re la trasmissione per via di terminali lontani o con ostacoli frapposti. Con il *carrier sense* riusciamo ad identificarle, ma non ad evitarle.
Il protocollo **CSMA/CA** inizializza una trasmissione "gentile". Se il canale risulta libero, per un tempo fissato ascolta il canale e poi trasmette, senza utilizzo del *collision detection*. Trasmette per tutto il tempo. Dopo la trasmissione aspetta un tempo fissato e manda una ricevuta (l'eventuale nodo nascosto viene così informato della trasmissione).
Il sistema può comunque collidere. Se più macchine vogliono trasmettere tutte allo stesso access point senza sentirsi tra loro, allora non è più un problema di tempo ma di intensità del segnale. Non si possono evitare le collisioni. Si pensa allora di ***prenotare il canale***, per essere sicuri che nessun altro comunichi contemporaneamente. Prima di trasmettere dati si scambiano dei segnali per prenotare il canale. Utilizzando piccoli pacchetti per la prenotazione della banda si riesce ad evitare la collisione di grandi frame. Il sender prima di trasmettere manda un pacchetto **RTS** (request to send) alla stazione base con carrier sense. Il pacchetto può arrivare da solo oppure collidere con un altra richiesta di un nodo remoto o nascosto. I pacchetti sono piccoli e brevi ed il problema di collisione si risolve in fretta. Quando una stazione riceve un RTS corretto, risponde con un **CTS** (clear to send) al mittente permettendo la trasmissione. Il messaggio è tecnicamente recepito broadcast, così tutti lo ricevono e si informano del fatto che c'è già qualcuno che trasmette, attendendo il loro turno. Non previene le collisioni, ma riduce al minimo il tempo effettivo di collisione (perchè possono collidere solo pacchettini molto piccini).

Mobilità. L'host deve poter continuare a lavorare passando da un access point all'altro. Il passaggio è automatico perchè lo switch si ricorda del MAC address della comunicazione.

Lezione 7 ~ Sicurezza delle Reti

Introduzione. Sicurezza intesa non solo come prevenzione alle intrusioni, ma sono legati anche alla confidenzialità (posso mandare messaggi diretti che possono leggere tutti, oppure che possono leggere in pochi perchè contengono dati sensibili), i messaggi devono essere integri perchè non è possibile mandare messaggi importanti che perdono informazioni importanti, deve essere possibile l'autenticità. La sicurezza non si applica solo al livello applicativo, ma anche ai livelli sottostanti (per esempio protocollo TCP e livello di rete per il protocollo IP).

- ✔ **Confidenzialità.** Solo sendere e receiver devono capire il messaggio, con critorafia.
- ✔ **Autenticazione.** Assicurandosi dell'identità degli interlocutori (con crittografia).
- ✔ **Integrità del messaggio.** Chi riceve deve essere certo che il messaggio che legge sia lo stesso medesimo che è stato mandato.
- ✔ **Sistema accessibile e disponibile.** Che non sono la stessa cosa. Una macchina è accessibile quando mi permette di usarla (quando è accesa), una macchina disponibile è una macchina che supporta il mio lavoro senza che mi pianti in asso.

Per esempio, esistono due attori di una conversazione ed un terzo si vuole intromettere. Gli attori comunicanti sono host, server, sono due punti qualsiasi impegnati in una qualsiasi comunicazione (server DNS, tabelle di inoltro, ecc). Chi si intromette può intercettare i messaggi; inserirsi attivamente nella comunicazione inserendo messaggi diversi nella connessione (mandare messaggi a nome di altre persone, *impersonandolo* clandestinamente); sostituirsi in una vera comunicazione tra due persone; infilarsi nella conversazione e lentamente sostituirsi ad una delle due persone; una o più macchine che esistono diventano di fatto inaccessibili ed inutilizzate (denial of service).

Crittografia. Non esistono macchine sicure, le uniche sono quelle fuori rete e spente. La crittografia ci permette di camuffare il contenuto di un messaggio. Ci sono filosofie diverse:

- ✔ **Cifratura a chiave simmetrica.** Due interlocutori si mettono d'accordo su una stessa chiave e la si usa per criptare e decodificare un messaggio. La chiave è condivisa. La più tradizionale è la cifratura di Cesare, che decide uno *shift* nell'alfabeto di caratteri usati.
- ✔ **Cifratura a chiave pubblica.** Esistono una chiave pubblica ed una provata, diverse tra loro. La chiave pubblica va scambiata liberamente, e solo chi possiede la chiave provata conosce la chiave segreta. Il gioco è: una chiave segreta è personale, ma altri usano una chiave "pubblica" che permette di cifrare e decifrare con chiavi diverse. Cosa non immediata.

Il baco fondamentale è dovuto al fatto che gli interlocutori debbano mettersi d'accordo e

scambiarsi la chiave (se simmetrica); deve essere un algoritmo di cifratura semplice che lavori su un computer e che lavori con i bit. **DES** (*Data Encryption Standard*): algoritmo che usa una chiave a 56 bit elaborando parole a 64 bit, rimescolando tutto. Per forzare brutalmente l'algoritmo servivano 4 mesi. Ma col tempo è diventato fallace, allora si è iniziato ad usare un algoritmo con 3 chiavi in successione.
AES (*Advanced Encryption Standard*): le parole di 128 bit sono cifrate da chiavi fino a 256 bit. Con la forza bruta servono 149 trilioni di anni (il DES si risolve in 1 secondo). Il problema della simmetria è la sicurezza dello scambio della chiave.
Nella cifratura a chiave pubblica, dev'essere impossibile computazionalmente derivare la chiave privata.
RSA: per la cifratura (rendere illeggibile il messaggio in circolo) in chiave pubblica chi spedisce cifra con la chiave pubblica del destinatario in modo che quest'ultimo, unico che conosce la chiave privata, può decifrarlo. Se applichiamo la chiave privata al risultato dell'applicazione della chiave pubblica su un messaggio, otteniamo un risultato che si ottiene anche applicando la chiave pubblica ad un messaggio cifrato con la chiave privata. Non cambia il concetto, ma cambia la funzione della decifrazione del messaggio. Anche se il risultato è lo stesso, non è che esiste un difetto, ma un'opportunità per utilizzare le chiavi di cifratura per sfruttare a nostro vantaggio questa caratteristica per autenticare un messaggio.
Quando autentichiamo vogliamo che chi riceve il messaggio abbia la garanzia che chi manda il messaggio sia veramente chi si aspetta che sia:

- ✔ Io ti dico che sono io. Io dichiaro la mia identità.
- ✔ Qualcuno si spaccia per me. L'intruso manda un messaggio al destinatario usando nome ed indirizzo IP del vero mittente.
- ✔ Dichiaro una prova della mia identità, ma crittografata. L'indirizzo IP è una informazione della corrispondenza tra persona e macchina mittente. Mandiamo un pacchetto con indirizzo IP e password della persona: questa è la prova di autenticità del mittente. Con l'**attacco di playback** l'intruso intercetta il messaggio con password e la utilizza per spacciarsi per il vero mittente. La password allora non si manda mai in chiaro (comunque l'intruso può non tradurre la password, perchè è il destinatario che deve tradurla per l'identificazione, allora all'intruso basta prendere la password cifrata e mandare un messaggio suo senza problema).
  Il playback funziona solo dopo l'intercettazione del messaggio. Si fa in modo che il codice cifrato di identificazione cambi tutte le volte: si usano i **nonce** (numeri usati statisticamente una volta e poi mai più). Bisogna contattare in modo aperto in destinatario, che restituisce un numero che dovrà esser usato come password cifrata (cifrata con la chiave simmetrica).
  Si può fare anche con chiave pubblica, ma fatto in modo che sia garantito che il mittente sia sempre chi dice di essere, così il numero verrà cifrato con la chiave privata del mittente. La chiave pubblica viene spedita in chiaro.

Il sistema è abbastanza sicuro, ma esiste sempre un **security hole**: tutto va bene a meno che l'intruso non riesca ad insinuarsi subito e sempre nella conversazione (*men in the middle*). In questo modo i due interlocutori non se ne rendono conto, ma l'intruso si insinua nella loro conversazione impersonando a turno mittente e destinatario. Non è risolvibile a meno di una chiave anche sulla chiave pubblica (si ottiene solo con l'intervento di una terza parte che garantisca).
La cifratura è sempre un lavoro dispendioso.

Integrità del messaggio. Si possono mandare messaggi confidenziali, messaggi autentici (non necessariamente confidenziali) e messaggi integri. È desiderabile che queste qualità valgano assieme. La cosa più importante comunque è l'integrità del messaggio, che non siano state apportate modifiche rispetto al messaggi originale. Non deve però risultare più pesante del messaggio di partenza.

- ✔ **Cryotigraphic Hash.** Produco un messaggio di lunghezza fissata (dato un messaggio di partenza) tale che sia computazionalmente impossibile trovare due messaggi di poco diversi che abbiano lo stesso hash. Deve essere computazionalmente impossibile anche ricostruire all'indietro il messaggio originale. Il checksum non rientra nella categoria.
  In pratica esistono funzioni si hash strandard, la più diffusa è **MD5** che produce messaggi di autenticazione di 128bit e che fa sia che sia difficile costruire messaggi che abbiamo lo stesso MAC del messaggio originale. Il **SHA1** è uno standard americano che usa 160bit (più

bit mettiamo più il messaggio è sicuro e più tempo serve a sbrogliarlo).

Firma digitale. Corrispettive delle firme reali, garantiscono l'autenticità di ciò che è stato firmato e sottoscritto. Per firmare un messaggio si usa la cifratura a chiave pubblica. Il problema è che vogliamo garantire le firme (essere in grado che un messaggio sia autentico e firmato dal vero titolare), il messaggio non potrà essere ripudiato dopo essere stato firmato (ed avrà valore legale).
Le Certification Authority sono entità ufficiali che siano in grado di fornire un legame sicuro, certificato e garantito tra l'entità in rete e la persona che la usa.

SSL (Secure Sockets Layer). Non tutto avviene a livello di applicazione, così anche a livello di trasporto si possono applicare sistemi di sicurezza per quelle applicazione che usano TCP e vogliono fare trasporto sicuro (ma in questo contingente non per forza affidabile) dei dati. Si deve fare l'autenticazione del server, la cifratura dei dati ed eventualmente l'autenticazione del client. La sicurezza qui è applicata alla socket e non a livello superiore. Autenticare il client? È più facile avere la garanzia che il server sia sicuro piuttosto che server che chiede la sicurezza al client.

- ✔ **Handshake.** Si fa in tre fasi. Viene fatta una autenticazione a livello di TCP (con SYN, SYNACK e ACK). Una volta stabilita la connessione TCP parte una analoga connessione SSL. Il sender (client) manda un HALLO SSL ed il server manda un certificato, allora il server si autentica attraverso il certificato e cifrato dalla chiave pubblica della *Certification Authority*. Il client allora manda al server una chiave di cifratura spedita con la chiave pubblica (non è il client che deve autenticarsi, ma vuole mandare la password segreta). Il server autentica con la chiave privata in suo possesso.
- ✔ **Key Derivation.** La chiave master server a generare altre chiavi (per client e server). L'algoritmo di cifratura è negoziabile.
- ✔ **Data Transfer.** Il pacchetto con i dati viene scambiato tra client e server. I bit vengono presi a blocchi alla volta e passano attraverso la funzione di hash. L'hash viene aggiunta all'insieme dei dati e si cifra con la chiave l'oggetto composto da dati+hash. I dati cifrati assieme all'hash vengono infilati nel "segmento" (record SSL) che permette la trasmissione del messaggio. Tutto questo costituisce il corpo dei dati del pacchetto.

I sistemi SSL garantiscono gli scambi di dati sul web.

IPsec. Per certi messaggi il livello di trasporto è troppo alto, e la sua sicurezza è insufficiente. Serve sicurezza a livello di rete. Non serve cifrare il contenuto di un messaggio o di un segmento, ma il contenuto di un datagram (che contiene un frame UDP/TCP o altro) per garantire le solite segretezza, integrità ed autenticità. Ci sono diversi protocolli:

- ✔ **Authentication Header.** Garantisce l'autenticità dell'header del datagram. Non fornisce confidenzialità. Viene inserito dopo l'header e prima del campo dati (che comunque gira in chiaro). Nel campo identificatore c'è il dato 51. non c'è necessità da parte dei router intermedi di gestire il contenuto dell'identification header (processano solo l'header IP). Tutte queste info sono usate dal livello IP di sorgente e destinatario. È un *imbastardimento* del sistema a layer.
- ✔ **Encapsulation Security Payload.** Incapsula nel datagram un datagram cifrato ed autentico. Deve fornire anche segretezza. Bisogna cifrare oltre che autenticare. Il pacchetto mantiene il suo header IP, poi inserisce un header ESP ed un trailer ESP, seguiti da un segmento di autenticazione ESP.

Entrambi i protocolli si basano su un handshake e forniscono a livello di rete (quindi IP) l'equivalente delle connessioni unidirezionali (andata e ritorno su connessioni differenti).

Disclaimer. Questi appunti riguardano il corso di **Architetture di Reti** tenuto dalla prof. Luppi per il *corso di laurea in Informatica dell'Università degli Studi di Ferrara*. Sono appunti presi durante la lezione, quindi in certe parti incompleti e probabilmente pieni di errori ed imprecisioni. Non sostituiscono a nessun titolo le lezioni ed il materiale fornito dal docente (ma possono essere usati per un ripasso rapido o per orientarsi nel vasto programma del corso).
Agnese Farinelli è l'autore di questi appunti. Scaricabili... dal QRcode!