Reti di Calcolatori - 25/09/2012

Lezione 1. Introduzione alle Reti di Calcolatori

Cenni storici.

Tra gli anni ’60 e ’80 ci fu un’apertura dei sistemi. Le architetture erano basate su specifiche note e c’erano standard che le regolamentavano. E’ stato possibile quindi sviluppare periferiche di collegamento, il che è avvenuto principalmente quando è emerso il Personal Computer. Si è passati da sistemi formati da un grande calcolatore potente e piccole macchine collegate (*stupide*) ad un’apertura. Prima c’era infatti il computing centralizzato, un grande mainframe. Già verso la fine degli anni ’60 nacque internet su calcolatori scientifici etc. Gioca un ruolo importante la scala geografica: i collegamenti possono esser fatti tra uffici, su scala nazionale o internazionale. Negli anni ’90 le metodologie di collegamento sono state unificate dal TCP/IP, una suite di protocolli, che realizza l’interconnessione a livello software in base ad interconnessioni eterogenee. Ultimamente c’è stata un’esplosione di dispositivi, i calcolatori special purpose.

Classificazione.

In ambito geografico:

- **LAN** (*Local Area Network*): Reti di calcolatori ad estensione locale
- **MAN** (*Metropolitan Area Network*): Reti di calcolatori ad estensione limitata
- **WAN** (*Wide Area Network*): Reti di calcolatori che lavorano su scala ampia, anche mondiale.

Le reti di calcolatori sono modelli architetturali a livelli (strati) secondo un approccio dividi et impera. Le tecnologie locali sono:

- Cablate (con fili)
- Wireless (senza fili, collegamenti radio)

In questi anni è nato il paradigma Client - Server. Il Client attinge al servizio e il server lo offre. Il client prende l’iniziativa e il server è in attesa.

Per quanto riguarda il cablaggio, prima c’era incompatibilità delle interconnessioni a livello fisico, c’erano cablaggi proprietari. Nascevano problemi di incompatibilità. Poi sono stati costruiti su standard aperti.

Una rete di calcolatori nasce per interconnettere dispositivi elaborativi, che sono:

- **Terminali**: vi girano gli applicativi, sono detti anche host. Possono essere client o server, sono fatti da dispositivi intermedi.
- **Intermedi**: sono elementi interni costitutivi di una rete di calcolatori. Una rete che collega vari dispositivi host è fatta da una serie di dispositivi intermedi che realizzano la connessione di rete.

I programmi applicativi possono essere progettati attraverso 2 modelli:

- **Client-Server.**

Il client è tipicamente dotato di interfaccia per l’utente. Il server risponde al client e offre servizi, è sempre in esecuzione su una macchina. E’ implementato ad un programma che gira in background

(es. daemon).

- **Peer-to-peer.**

Non si diversificano client e server perché a seconda delle situazioni una macchina gioca un determinato ruolo.

Infrastrutture di rete.

E' un sistema di dispositivi o collegamenti. L'infrastruttura di rete può essere divisa in due tipologie:

- **Reti di accesso**: servono per dare connettività ai terminali, consentono ad un singolo PC di collegarsi alla rete.
- **Reti di backbone**: servono per connettere apparati intermedi (router, ecc…). Sono le dorsali del sistema e sono collegate alle reti di accesso. La rete backbone non è utile se non è collegata a reti di accesso e si basa su molteplici flussi di informazione. Sui link viaggiano aggregati di traffico, non un solo flusso di informazioni end-to-end.

Commutazione a circuito.

E' usato nelle telecomunicazioni. C'era la necessità di consentire a più coppie di terminali una connessione per comunicare contemporaneamente, quindi è sorta la commutazione a circuito.
Esempio: In una centrale che gestisce più telefonate, si assegna la capacità trasmissiva su base chiamata e si riserva una certa quantità di capacità trasmissiva su ogni telefonata. Se un link porta 2000 telefonate, 1/2000 viene assegnata ad una chiamata.

C'è una fase iniziale di segnalazione. La rete capisce quali sono le entità e stabilisce la comunicazione riservando una capacità trasmissiva. Abbiamo quindi che un certo numero di bit sono riservati per una specifica telefonata. Quando si attacca, si libera. Ci sono un certo numero di bit/s che possono essere trasmessi. 65 kBit/s è lo standard per le telefonate. La capacità è sempre un suo multiplo.

Un vantaggio sta nel fatto che, per tutta la durata della telefonata, c'è una banda trasmissiva dedicata. L'assegnazione della banda avviene a valle di una scelta di percorso. La commutazione viene anche utilizzata perché le risorse di trasmissione della rete sono limitate.
Si parla di congestione quando un link è pieno.

La TDM (multiplazione a divisione di tempo) può portare ad uno spreco di capacità. Non è un danno per le telecomunicazioni ma lo è per le comunicazioni dati tra PC.

Commutazione a pacchetto.

Si può dividere l'informazione in porzioni di banda o porzioni di tempo. Sono forme statiche di assegnazione, perché una volta assegnata una banda ad una connessione, essa resterà la stessa per tutta la durata della connessione stessa.

Il meccanismo della commutazione a pacchetto è usato per le reti di calcolatori. Il flusso di informazioni si spezza in pacchetti ognuno dei quali contiene un'intestazione con l'indirizzo del mittente e del destinatario. I pacchetti sono indipendenti tra loro.

Multitexting statistico di risorse: sono assegnate in base a chi ne ha bisogno. Serve quindi più intelligenza negli apparati terminali.

Quando un pacchetto arriva ad un dispositivo intermedio, questo lo mette in una coda, gestita secondo una politica FIFO. Non c'è un'assegnazione statica delle risorse. La coda serve a compensare il fatto che a volte il flusso di pacchetti è elevato. Si utilizza una tecnica di **Store ad Forward**: ogni nodo memorizza i pacchetti iningresso, per poi instradarli verso il nodo successivo. A volte sono molto frequenti e a volte meno frequenti, quindi la coda può allungarsi o accorciarsi fino a non esserci.

In certi casi di congestione, un pacchetto può essere droppato o perso quando il dispositivo non ha memoria per mantenerlo. La dimensione dei pacchetti non può quindi essere troppo grande, altrimenti si potrebbero perdere molte informazioni, né troppo piccola, altrimenti ne sono troppi e l'header avrebbe troppo peso. Ci sono trade off da rispettare.

La tempistica per la trasmissione di un pacchetto si basa sul numero di bit/s che possono trasmettere i link (che determina la velocità) e il tempo di accodamento (tempo di attesa prima della trasmissione). Inoltre c'è il tempo di attraversamento del collegamento (supponendo sia a velocità costante).

Qual è il **tempo di elaborazione di un pacchetto** in tutto il suo viaggio?

- Tempo di **pacchettizzazione**: il terminale deve mettere le informazioni in un pacchetto.
- Tempo di **trasmissione** su ogni collegamento: la quantità di contenuto nel pacchetto determina il tempo di trasmissione.
- Tempo di **accodamento**: è il tempo di attesa nella coda, che anch'esso risulta variabile.
- Tempo di **attraversamento** del collegamento: è direttamente proporzionale alla lunghezza fisica del collegamento.

Avere un ritardo di attraversamento elevato è un problema nelle comunicazioni real-time. Generalmente il limite di accettabilità è 0.5 s.

Il **Jitter** è la variazione del ritardo, che non è sempre costante, poiché dipende anche dalla lunghezza della coda. Con un jitter elevato certe applicazioni possono entrare in crisi.

Un'altra classificazione di reti di calcolatori è data da:

- **Reti a datagrammi.**

E' il modello fondamentale che ha ispirato internet. Su ogni pacchetto il dispositivo intermedio lavora come se fosse un'entità atomica e quindi fa autonomamente la decisione di instradamento del pacchetto.
Decisione di instradamento: i pacchetti sono indipendenti tra loro, possono avere velocità differenti e prendere strade diverse. Ci possono essere sorpassi tra pacchetti. Non si viola la temporalità però, esiste una logica di riordino.

- **Reti a circuiti virtuali.**

Prima di trasmettere i pacchetti, i terminali fanno decidere ai dispositivi un circuito virtuale per il quale dovrà passare il pacchetto. I circuiti virtuali sono identificati da un numero che viene messo attraverso un codice nell'intestazione del pacchetto. Nelle trasmissioni spesso c'è l'identificativo del circuito non del destinatario.

Reti di calcolatori - 28/09/2012

Lezione 2. Modelli a strati delle reti di calcolatori

Caratteristiche fondamentali delle reti di calcolatori.

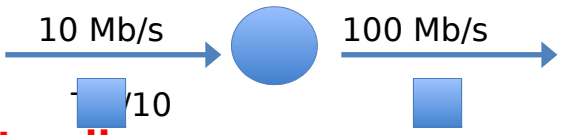
La commutazione di pacchetto si basa sulle reti a datagrammi e le reti di circuiti virtuali. La rete backbone allarga la portata geografica della rete. Copre fisicamente delle distanze.

L'informazione tra due terminali viaggia sottoforma di pacchetti. Il pacchetto è lo strumento attraverso il quale si realizza la suddivisione delle risorse trasmissive della rete.

- **Reti di accesso**: parti periferiche della rete
- **Reti di backbone**: collega i vari dipositivi intermedi e che ha come scopo quello di allargare la portata geografica della rete.

Reti di accesso e reti di backbone hanno una caratteristica che li distingue.
La **rete di accesso** ha una struttura di tipo stellare, cioè i percorsi che consentono a un singolo terminale di raggiungere gli altri terminali sono pochi, avendo quindi un basso grado di ridondanza. Se un collegamento è rotto il terminale non è connesso alla rete.
Nelle **reti di backbone**, invece, il grado di connessione della rete è maggiore, e quindi il fatto che un collegamento si guasti non pregiudica i collegamenti, perché ne sono di più. Sui collegamenti delle reti di backbone viaggiano flussi di traffico aggregati, non singoli.I nodi intermedi funzionano secondo una tecnica di ***Store & Forward***. I nodi intermedi funzionano attraverso un meccanismo di store and forward: un dispositivo intermedio ha una linea di uscita su cui si smista il traffico di più ingressi. Può accadere che la quantità di pacchetti in entrata ecceda la capacità di smistamento sul link. Occorre che il nodo possa gestire una coda (capacità di accumulo di pacchetti). E' essenziale. Questo compensa picchi di traffico. La capacità di assorbire picchi è basata sulla massima lunghezza della coda (necessariamente limitata). Può accadere che la coda si riempia e per congestione il pacchetto si perde.

Se un dispositivo intermedio ha un link a 10Mb/s a monte e un link a 100 Mb/s a valle, questo significa che i pacchetti sul primo collegamento arrivano spaziati nel tempo in una certa maniera e quando il pacchetto viene ritrasmesso nel secondo collegamento viene inviato in un decimo del tempo. Quindi la velocità di trasmissione a valle, chiamata quella a monte *t*, sarà *t/10*.
Un altro tempo significativo è il tempo di trasmissione del collegamento: maggiore è la distanza fisica del collegamento e più tempo ci metterà il pacchetto ad arrivare a destinazione.



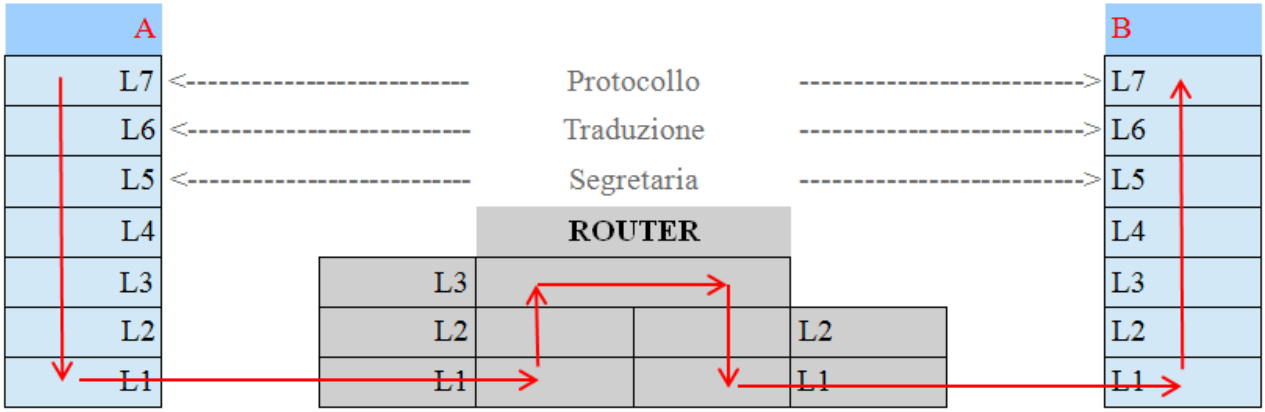
Livelli architetturali.

I progettisti di reti di calcolatori seguono un approccio divide et impera: si divide un problema in più sottoproblemi divisi in strati, dove ciascuno strato comprende le tecniche e i dispositivi che sono atti a risolvere uno specifico problema. Ogni strato è specifico della particolare rete. Le prime reti erano basate su modelli architetturali proprietarie, ma erano improntate ad un modello stratificato. Poi, si fece un tentativo di standardizzazione (ISO), e si creò un modello standardizzato di reti di calcolatori basato su 7 strati a cui fu dato un nome: **Modello OSI.**

L7	Applicazione
L6	Presentazione
L5	Sessione
L4	Trasporto
L3	Rete
L2	DataLink
L1	Fisico

Ha una valenza storica, è stato superato e viene usato raramente. Lo ritroviamo sulla rete internet con il modello semplificato a 5 strati. Le funzionalità dei 7 strati non sono necessariamente implementate dai dispositivi che costituiscono la rete (terminali e dispositivi intermedi). Ad esempio, i dispositivi intermedi si differenziano tra di loro a seconda degli strati su cui lavorano. Sui terminali, invece, possiamo immaginare che siano implementati tutti e 7 gli strati del modello.

Immaginiamo di avere due terminali che comunicano attraverso un dispositivo intermedio (router). A e B comunicano attraverso il router. Da un punto di vista di architettura a strati, i 2 terminali complicata. Il router ha più interfacce grafiche che comunicano con i terminali, impilate in uno stack protocollare implementano tutti e 7 gli strati, mentre il dispositivo intermedio presenta una situazione più (a 7 livelli). I modelli a strati sono detti pile o stack protocollari perché sono rappresentati con una pila.



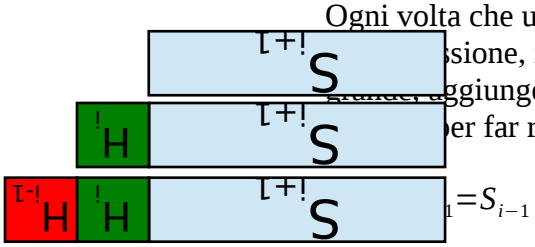
Il router lavora fino al livello 3, e i livelli dal 4 in poi li troviamo presenti solo nei dispositivi terminali (end-system, per questo si parla di servizio end-to-end, quando interessano solo i terminali e non gli altri intermedi).

In queste reti il colloquio avviene sempre tra entità paritetiche che parlano attraverso un insieme di regole, che costituisce un **protocollo** (insieme di regole che governa il colloquio tra due entità paritetiche).

Nel modello a strati, c'è un'interfaccia di servizio, attraverso la quale il livello inferiore offre un servizio al livello superiore. Questo servizio lo fa appoggiandosi del servizio dell'entità che gli sta sotto.

Quando A vuole inviare un messaggio a B, c'è un attraversamento della pila di livelli dall'alto verso il basso (il protocollo ha bisogno di traduttori, che ha bisogno di segretari, ecc...) fino al livello più basso, quello fisico, che trasmette fisicamente sul collegamento la sequenza di bit che viene ricevuta dall'altra estremità del collegamento senza avvalersi di nessuno strato inferiore per poter compiere la trasmissione. L'interfaccia definisce i servizi che ogni strato fornisce agli strati superiori. Avvien quindi che un certo messaggio inviato da A, alla fine diventa una sequenza di bit che vengono trasmessi su supporto. Affinché ogni strato possa comunicare con la sua controparte ci si avvale

della traduzione da parte dello strato inferiore. Lo strato fisico è l'unico che non ha strati sotto. Lo strato applicazione non ne ha sopra. La pila di livelli viene attraversata fino al livello fisico.



Ogni volta che uno strato passa un messaggio allo strato inferiore [illegible]ssione, il livello inferiore costruisce un messaggio più [illegible]ggiungendo all'entità un'intestazione. Le informazioni [illegible]er far recapitare correttamente il messaggio.

$=S_{i-1}$

C'è un progressivo imbustamento. Questa cosa avviene ogni volta che c'è un attraversamento di interfaccia verso il basso. Questo attraversmento di livelli avviene dal lato A. Quando la sequenza di bit arriva a B, c'è un attraversamento della pila di livelli dal basso verso l'alto, quindi ogni volta viene consumata l'intestazione del livello precedente e viene passata solo la parte utile del pacchetto (***payload***). Il pacchetto che arriva al livello applicazione B è la stessa cosa che è stata generata a livello applicazione A.
PDU (protocol data unit): Formato da Header+Payload
L'intestazione H_i, non interessa nulla all'entità del livello i-1, e l'entità i-1 ci aggiunge solo l'intestazione H_{i-1}.
L'idea fondamentale è che i due terminali parlano attraverso dei mediatori che sono in parte collocati dal loro lato e in parte anche in mezzo. I dati nell'intestazione servono all'entità i-1 per trattare il pacchetto, il payload non lo guarda. C'è una crescita del pacchetto in trasmissione e riduzione dell'header in ricezione.

Modelli a strati: perché?

Alcuni strati sono realizzati in software e altri in hardware. Questa stratificazione mi dà dei vantaggi, come ad esempio, mantenendo compatibili le interfacce, posso sostituire uno strato con un'altro differente che però avrà la stessa interfaccia.
Alcuni livelli aggiungono all'intestazione anche una coda chiamata ***tailer*** (informazione di controllo).

1. **Livello Fisico**: trasmette e riceve sequenze seriali di bit, spesso anche con conversione digitale/analogico.
2. **Livello Data Link**: il pacchetto viene chiamato frame. Inoltre questo livello serve quando la comunicazione tra vari dispositivi avviene attraverso un unico mezzo trasmissivo condiviso. Viene disegnato come se fosse un bus, dove quando un dispositivo occupa il bus, altri dispositivi non possono occuparli.
3. **Livello di Rete**: Decide l'instradamento dei pacchetti.
   - Nelle reti a circuiti virtuali c'è un'attività di segnalazione che fa scatenare una scelta di instaurazione del circuito, cioè mette un'informazione di stato in tutti i dispositivi intermedi.
   - Nelle reti a datagrammi non c'è un'informazione di stato per singola destinazione, ma ci sono scelte di instradamento che vanno prese ad ogni dispositivi intermedi.
   - Un altro compito particolare è il trattamento di eterogeneità dela rete, cioè c'è un protocollo pensato per essere implementato indipendentemente dal dispositivo su cui è applicato. (es.: Internet)

4. **Livello di Trasporto**: Ha come peculiarità quella di essere implementato solo negli end-system (ma non è detto, spesso anche gli intermedi sono terminali di un'applicazione!). Realizza funzioni di tipo multiplexing e demultiplexing del flusso: ad un singolo terminale arrivano diversi flussi informativi che quel singolo terminale deve elaborare. L'host parla con tanti diversi host. Una coppia di host spesso risolvono flussi informativi: in parallelo quindi il MUX è necessario per smistare le informazioni ai processi: bisogna smistare un flusso in un processo invece che in altro. Il livello trasporto offre un'interfaccia di servizio, la più comune è la socket API.
5. **Livello di Sessione e Presentazione**: Servono servizi non sempre necessari.
    - Quando due entità intendono scambiare delle informazioni, questo accesso al servizio passa attraverso molteplici connessioni di livello trasporto, e occorre mantenere un collegamento tra queste varie connessioni che dà luogo a un'informazione di stato che risiede nel livello di sessione.
    - Il problema a livello presentazione sta nel fatto che i dati possono essere codificati diversamente in macchine differenti (es.: big endian, little endian). Allora quando comunicano due macchine occorre che questi dati siano rappresentati in maniera tale da permettere il trasferimento dei dati.

L1. Livello Fisico

Codifica e decodifica dei bit, dice il segnale come deve essere fatto (che potenza deve avere, ecc...) e specifica le forme standard dei connettori e le caratteristiche degli stessi.

L2. Data Link

Opera vedendo pacchetti, che si chiamano frame. Può accadere che le tecnologie trasmissive sono diverse tra i vari collegamenti, e anche la frame sia fatta in maniera differente sui vari collegamenti. Quando l'entità A trasmette una frame e questa frame che arriva all'altra estremità del collegamento ha un errore, la frame viene scartata. Questa decisione viene presa a livello data link. Se al livello intermedio c'è un errore nel datalink viene scartato. Il recupero è ai livelli successivi. A volte il datalink chiede nuovamente il pacchetto.

L4. Livello Trasporto

Realizza il meccanismo multiplexing/demultiplexing.

Ci sono due attività importanti:

- **Controllo di flusso**: controllare la velocità con la quale sono trasmessi i pacchetti perchè il flusso di pacchetti che arriva al destinatario non ecceda la capacità di elaborazione del destinatario stesso. Si vuole evitare l'overflow della coda.
- **Controllo di congestioni**: una congestione è un evento che si verifica in uno dei dispositivi intermedi sempre per sovraccarico per il concorrere di varie comunicazioni contemporaneamente. Questo evento è deleterio perchè non danneggia una sola coppia di terminali, ma tante cppie di terminali contemporaneamente. Quando questo evento si verifica, la rete deve prendere delle contromisure: rallentare la trasmissione da parte dei terminali che emettono l'informazione.

L7. Livello Applicazione

I protocolli applicativi sono standard e sono implementati nelle applicazioni per poter offrire dei servizi.

Reti di Calcolatori - 02/10/2012

Lezione 3. Introduzione alla architettura di Internet ed ai protocolli TCP/IP

Per internet mancano gli strati di sessione e rappresentazione. I dispositivi intermedi si differenziano a seconda delle funzionalità e quindi degli strati che sono attivi. Alcuni strati sono realizzati in hardware (strati più bassi come livello fisico e datalink sono implementati negli hardware delle schede di rete) e altri in software.
Esistono link molto dipendenti dalla tecnologia realizzativa, altri basati su un supporto [*o controllo? ndR*] fisico e altri wireless. Un collegamento è qualcosa che consente la trasmissione seriale di bit da una macchina all'altra.

Dal livello 3 in poi si tratta di funzioni implementate in software (3, 4 almeno gli end system sono implementate nel SO della macchina e il livello applicazione è tipicamente software). Nei router (dispositivi intermedi) anche la logica di livello 3 è in hardware per massimizzare la logica del dispositivo (i livelli sono 3) ed aumentare le prestazioni. Invece negli end system la funzione livello 3 è implementata nel SO della macchina.

Protocollo TCP/IP.

IP è un protocollo di livello 3, TCP è un protocollo di livello 4.

L'idea base delle reti calcolatori è la commutazione di pacchetto, che nasce da Leonard Kleinrock, che ebbe quest'idea negli anni '60, perchè la considerazione di base è il fatto che la commutazione di circuito che stava nelle reti telefoniche si adattava male alla trasmissione di dati tra reti di calcolatori. Allora, Kleinrock parte con un modello matematico: vuole modellare il collegamento tra calcolatori attraverso una rete di code (*teoria delle code di Jackson*). L'esigenza di collegare calcolatori era evidente in ambito scientifico, allora l'idea di realizzare una rete fu finanziata come attività strategica dall'ARPA, agenzia per la difesa americana, che a differenza di altri decisero di finanziare questo progetto. Si passò da primi modelli concettuali a modelli implementativi e a realizzazioni prototipali. Già all'inizio fu coinvolta la BBN, che ebbe l'incarico di realizzare i primi prototipi di sistemi di tipo calcolatore, specializzati nel collegamento, attraverso collegamenti di tipo geografico. Basandosi sul modello architetturale sviluppato da un gruppo di lavoro con K e altri ingegneri realizzò in tempi rapidi i primi prototipi di sistemi collegati in rete. Il modello era diverso dall'attuale, gli equivalenti dei router erano gli IMP (calcolatori special purpose che fungevano da router che realizzavano la commutazione di pacchetto). A gli IMP si collegavano i calcolatori. Non esisteva ancora la rete locale. Stava appena nascendo questo concetto.
Ad un dispositivo IMP si collegava direttamente in maniera esclusiva un computer, un host. Il primo prototipo di rete realizzato (fine '69) prevedeva 4 IMP collegati tra loro rispetto una specifica topologia in quattro siti (IMB, UCLA e altre università). Questi collegamenti erano geografici a velocità di 56kbit/s.
In questo primo prototipo di internet, esistevano due diversi protocolli di livello rete, uno tra IMP e IMP, che serviva a stabilire la connessione tra gli IMP e uno tra host e host, che serviva a collegare due host. Fu sviluppato prima l'IMP-IMP. Successivamente nacque l'esistenza di realizzare protocolli di tipo trasporto e il protocollo di livello rete evolvette fino a diventare quel che conosciamo noi e ci fu un'aggiunta di altri siti appartenenti ad istituzioni pubbliche ed enti di ricerca americani.
La DARPA lo finanziò perché eravamo in epoca di guerra fredda e servivano gli scambi di informazioni che potavano resistere ad attacchi di tipo nucleare, cioè la rete deve essere robusta alla

caduta di link (con ridondanza e facendo in modo che non esista uno stato che se perso rende la rete ingestibile). Si consente alla rete di rigenerarsi automaticamente e riarrangiare l'invio di pacchetti consentendo comunicazione. Questo era realizzato dalla rete a datagrammi nelle comunicazioni a pacchetto. Questo prototipo di rete prese il nome di ARPANET.

All'inizio degli anni '70, nacquero le prime applicazioni, come quella della posta elettronica (scambio di messaggi in maniera asincrona). Le prime sperimentazioni mostrano l'inadeguatezza dei protocolli ARPANET, e nacque la necessità di non dover rifare i protocolli ogni volta che si aggiungevano collegamenti realizzati in tecnologia differente. Venne creato il TCP/IP nei sistema Unix a quel tempo.

PDP/VAX è un'architettura di minicomputer dell'epoca su cui girava unix. TCP/IP: consentiva di parlare attraverso la rete.
Nascita di reti locali, che non consentivano di collegare solo host remoti, ma servivano anche a collegare host che stanno nello stesso campus.
T1: collegamento digitale che porta collegamenti multipli di 56k.

Mano a mano che passa il tempo, la rete si potenzia come capacità di collegamenti tra i siti, che passano da 56 Kbps a T1 (1,544 Mbps). La rete comincia a svilupparsi secondo un sistema gerarchico, non paritetico (peer-to-peer). Si crea prima una dorsale, NSFNET. Verso la fine degli anni '80, si smantella la parte militare, la gestione passa da quella pubblica a quella privata e il backbone viene gestito da un'organizzazione no profit, l'ANS, che è un consorzio in cui rientrano grossi player delle telecomunicazioni.
La backbone diventa un territorio neutro su cui si agganciano le diverse compagnie, cioè degli operatori locali, attraverso dei punti di accesso. Il modello concettuale di internet è una rete di reti **(*slide 8*)**. Sono tante reti sia locali che geografiche. Questa connessione avviene attraverso reti di backbone, che sono anch'esse diverse tra di loro.

Esiste una gerarchia di operatori, che con le loro infrastrutture connesse realizzano internet. Queste infrastrutture offrono gli ISP provider, che formano una struttura gerarchica (locali, nazionali, backbone), e sono quelli che posseggono i collegamenti gerarchici su un'unica tratta. I collegamenti tra operatori di backbone avvengono attraverso punti di connessione chiamati Network Access Point. Spesso si parla di Tier, come sinonimo di backbone. Ad ogni livello ci sono problematiche di instradamento, che si risolvono con approcci differenti: ci sono problemi nella singola rete e problemi tra reti diverse. Dal punto di vista topologico, le reti periferiche tendono a essere più di forma stellata, mentre nelle reti di livello gerarchico superiore esiste un grado di connettività maggiore.
Un'organizzazione di infrastruttura efficiente ha bisogno di un intervento pubblico.

Le tecnologie che coinvolgono internet sono diverse, i protocolli che consentono la comunicazione tra host e router sono definiti da standard, la maggior parte dei quali sono prescritti dalla Internet Society (organizzazione no-profit, che si struttura in IETF e IRTF, la prima che standardizza i protocolli, la seconda che fa operazioni di ricerca). La IETF è un aggregato di persone che realizza in maniera peculiare, cioè agile, la standardizzazione.
Gli enti di standardizzazione sono sempre esistiti nel mondo delle telecomunicazioni per problematiche di interconnessioni nel mondo. CCITT standard di tipo telefonico importante al tempo. Un altro corpo di standardizzazione delle telecomunicazioni è l'ITU di Ginevra.
Quando si vuole definire un nuovo standard si organizzano gruppi di lavoro e si fanno delle call per proporre gli standard. Poi si fanno una serie di inciuci e mediazioni e la proposta più forte vince. E' un processo lento.

L'esigenza della velocità non è mai stata sentita troppo, ma nel mondo evoluto in maniera rapida, c'era la necessità di lavorare con procedure diverse. Gli standard venivano scelti in maniera molto lenta, ma poi il processo è diventato molto più rapido, grazie all'IETF, che utilizza proprio internet per poter definire lo standard. Il gruppo di lavoro è una definizione di un certo numero di documenti che rappresentano uno standard.

L'IETF opera grazie ad internet e la posta elettronica, è organizzata in aree e gruppi di lavoro specifici (gruppi di lavoro con milestone da raggiungere, muoiono dopo). Un gruppo di lavoro definisce una serie di documenti che rappresentano gli standard (architetturali o di protocollo). Questi documenti emergono attraverso la cooperazione di gruppi di lavoro e società pubbliche. Due volte l'anno l'IETF si unisce in meeting enormi, si fanno votazioni pubbliche e quando si raggiunge un livello sufficiente di consenso le proposte di standard progrediscono fino a diventare standard, definiti in RFC (request for comments) che sono dei documenti approvati.

Lo standard viene approvato dall'IETF e viene chiamato RFC (Request for Comments), scritti in formato testuale puro ASCII. E la caratteristica fondamentale è la snellezza di lettura.
Per tutto quello che riguarda i livelli più bassi della gerarchia (1,2) non interessano all'IETF, e questo è possibile perchè il protocollo di livello rete è molto snello, cioè si adatta facilmente ad essere implementato a valle di livelli data-link anche elaborati.

I collegamenti sono sia tra host, sia tra router, sia tra host e router, quindi il dispositivo fondamentale è il router, un oggetto che lavora a livello 3, che ha come compito fondamentale quello di istradare i pacchetti e consegnarli a rete.
Ogni router fa del suo meglio per consegnare un pacchetto alla destinazione, ma ciò non significa che ci sia una garanzia che il pacchetto arrivi a destinazione, perchè il pacchetto potrebbe andare perso nella rete, in quanto ci potrebbero essere più problemi per cui il pacchetto arriva al router e poi viene perso per vari motivi. Come conseguenza dell'adozione del modello datagram, è il fatto che i pacchetti possono viaggiare su percorsi diversi e quindi possono arrivare in un ordine diverso da come sono stati mandati.

Quando è stato possibile per le persone collegarsi ad internet, lo si fa attraverso una rete ISP, che offre connettività attraverso reti locali. Le compagnie che offrono questo servizio sono le reti telefoniche, che si sono travate glàle infrastrutture per la trasmissione dei dati.
Ci sono varie difficoltà, come il fatto che la rete telefonica agli inizi degli anni '90 era di tipo analogico, trasmesso su una banda da 0 a 8 kHz. Su questa banda, che viene utilizzata da una rete telefonica, si è tentato di trasmettere dati, che sono sequenze di bit. Allora, due sono state le tecnologie che si sono potute adottare:

l'adozione di modem analogici: dispositivi che, una volta creato un circuito attraverso una telefonata, su questo collegamento telefonico, trasmettono un segnale che si ottiene attraverso la modulazione di una portante con una sequenza di bit, quindi il segnale è di tipo fonico, ma trasmette un segnale digitale, e in questo modo c'è il collegamento tra due modem che collegano tra di loro. Questo tipo di soluzione consente trasmissioni digitali sono fino a 56 Kbps e impegnava tutta la linea telefonica.
Verso la metà degli anni '90 sono nate tecnologie che consentono di usare il doppino telefonico per trasmettere il segnale telefonico e i dati contemporaneamente, alla velocità fino a 1 Mbps downstream e 20 Mbps in downstream.
Nel mezzo, c'è l'ISDN, che si basa su un'evoluzione della rete telefonica, che consisteva nell'offrire un servizio digitale fino alla casa dell'utente, quindi chi aveva una rete a casa doveva collegare un telefono particolare, e la velocità arrivava fino a 128 Kbps. Questa transizione è avvenuta verso le

fine degli anni '90, quando non era ancora chiara l'utilità dell'ADSL.

Questa tecnologia funziona fino a quando il collegamento tra ISP e la rete di casa ha una distanza fisica limitata (1,2 Km al massimo), ma questo è abbastanza vero nelle grandi città, dove c'è una densità elevata di centrali, ma non è vero nelle zone rurali, dove si tirano collegamenti più lunghi di 2 Km, che degradano le caratteristiche del segnale ADSL. Questo è vero oggi in Italia, mentre negli Stati Uniti, c'era il fenomeno delle TV via cavo, cioè negli USA c'era un forte sviluppo di collegamenti su cavo coassiale, che era stata adattata anche come infrastruttura di accesso a internet.

Quando uno ha una rete aziendale, cioè una rete locale, e ha necessità di collegarlo ad internet, i collegamenti che abbiamo visto prima non vanno bene, perchè hanno un bitrate troppo basso, e allora occorrono dei collegamenti di maggiore capacità, e le soluzioni sono differenti, e la possibilità di creare questi collegamenti risente il peso della tradizione di tutte le leggi e le normative che derivano dal mondo delle telecomunicazioni.

Negli ultimi anni, il legislatore ha imposto all'operatore predominante, che possiede il doppin otelefonico, la possibilità di cedere questo doppino perchè possa essere utilizzato da altri operatori per fornire il collegamento (pratica dell'unbundling). Questo doppino arriva in centrale, dove viene diretto in una zona fisica, nella quale un operatore terzo può installare i propri apparati. Gli operatori dicono all'operatore che possiede il doppino di girare l'utenza di una persona al proprio apparato.

Uno degli ISP più importanti è la GARR, ente pubblico che gestisce il backbone della rete che connette le varie università italiane e ha interconnessioni con altri operatori (GRANT, GX..)
In America ci sono molti collegamenti coast-to-coast caratterizzati da elevata potenza.

Gli operatori stabiliscono collegamenti con operatori di gerarchia superiore, che si paga, e in alcuni casi questi operatori ritengono conveniente stabilire una connessione paritetica tra di loro, e questa connessione non si paga.

La standardizzazione dei protocolli è una delle problematiche per il corretto funzionamento di internet, ma a livello operativo è ancora più importante la gestione degli indirizzi: ogni host è identificato univocamente da un indirizzo (sequenza di 32 bit). Questo indirizzo viene assegnato dall'ISP, ma ancora più a monte, occore un'entità che ripartisca i blocchi di indirizzi agli ISP commerciali, senza possibilità di conflitto, e questa attività viene fatta dall'ICANN.

Lo stack protocollare nel modello di riferimento di internet.

Al di sopra del livello trasporto segue un livello applicativo. Il protocollo di livello rete internet è solo uno: IP; nel livello trasporto ne esistono 2 più importanti:TCP e UDP, implementati negli end system, cioè nel sistema operativo. Il livello rete non può essere modificato direttamente e questo comporta una difficoltà di manutenzione: infatti la transizione alla versione IPv6 è molto lunga. Quando tutte le funzionalità del TCP non sono ritenute necessarie si usa l'UDP che è minimale. Per il livello applicativo abbiamo più protocolli: HTTP, FTP, ecc…

Tanti protocolli a livello datalink e fisico, un unico protocollo a livello rete, due protocolli a livello trasporto e molti protocolli a livello applicativo. Questa numerosità dei layer è rappresentata come una clessidra. Grossa eterogeneità sopra e sotto, grossa omogeneità nel mezzo. L'IP è un protocollo unificante. IP assume un modello di servizio sottostante estremamente minimale, nel senso che assume la possibilità di trasmettere un pacchetto attraverso un link da un router ad un altro. Questo

IP, attraverso percorsi, dà la possibilità di consegnare un pacchetto da un host all'altro. Ma come avviene la trasmissione del pacchetto, su questo IP non fa assunzioni particolari.
IP è talmente minimale che si può implementare su qualsiasi tegnologia trasmissiva sottostante, e a limite sono sottosfruttate da IP. IP offre la possibilità maggiore di consegnare un pacchetto da un host ad un altro, ma questo a volte ancora non basta, e per questo, sono state poi inventate una serie di soluzioni particolari a vari problemi. L'IP nasce per un mondo in cui i computer erano fermi.
L'indirizzo è legato alla posizione dell'host a cui si collega. Ad esempio, un portatile avrà sempre un indirizzo IP diverso.

Protocollo applicativo **HTTP.**

HTTP è un protocollo di livello applicativo: cioè è implementato negli end-sistem e supporta uno specifico tipo di servizio: nasce a supporto del servizio WWW, un servizio di condivisione di documenti di natura ipertestuale che vede l'accesso a questi documenti attraverso un programma client che visualizza gli ipertesti (testi arricchiti con collegamenti e figure multimediali). Queste entità vengono fornite da un programma server, che fornisce questi documenti che sono singolarmente identificati da un'etichetta URL.
E' un protocollo nel quale è specificato il ruolo del client e del server. E' un protocollo di tipo richiesta risposta: il client fa una richiesta, il server fornisce una risposta. E' molto importante non solo perchè il servizio WWW è importante ma anche perchè nel corso del tempo, questo stesso protocollo è diventato una modalità di interazione per programmi che vogliono scambiare informazioni attraverso internet, anche per scopi che vanno al di là del servizio WWW. Alcuni documenti hanno come protocollo di trasmissione HTTP.
Il web così come nacque negli anni '80, in realtà era nato per fornire documenti statici, preregistrati sottoforma di file in un server, che devono essere trasmessi ad un client, ma il web è diventato poi una forma di accesso a servizi che hanno bisogno di un'iterazione dinamica (ad es.: una ricerca, una prenotazione).
La cosa che rende il protocollo semplice, è che il protocollo è di tipo testuale: i messaggi inviati tra le due parti sono sequenze di codici ASCII, che se catturiamo e traduciamo nel codice ASCII diventano delle righe di testo che possiamo decodificare.
Le tipologie di messaggi di richiesta sono in numero limitato. Il messaggio più usato è quello di **get**: dammi la risorsa, che contiene, nella sua prima riga, oltre al metodo (tipo di messaggio di richiesta), l'identificativo della risorsa richiesta, identificata da un URL.

URL: http://nomehostserver[:numeroporta]/percorso/pippo.htm

Il numeroporta tradizionale è :80
[…] → opzionale

L'URL punta al file pippo.htm, che si trova sul server su un certo percorso, che corrisponde ad un percorso nel file system. Il programma server ha una sua cartella root, e a partire da questa, arriva scendendo di cartella in cartella al file pippo.htm. Se il file non esiste, il server risponde con un messaggio di errore.
Il server capisce se l'URL è di un file statico o un programma in base ad una convenzione fatta, magari c'è una cartella virtuale. Il server è un application server, risponde in maniera diversa in base alla cartella di riferimento (statica o virtuale).
Esistono diversi programmi webserver (Microsoft o Apache, opensource...) che offrono servizi server.

Il numero di porta è identificativo del processo che gira sulla macchina del server. Una volta

identificata la macchina, il pacchetto di richiesta può essere consegnato a più programmi in esecuzione. In questo caso, stiamo consegnando il pacchetto ad un'applicazione che sta in ascolto sulla porta 80. Il numero 80 viene utilizzato di default per servizi web-server. In realtà, quando parliamo di protocolli applicativi, dobbiamo sempre specificare il protocollo operativo su quale protocollo applicativo opera. La scelta del protocollo di trasporto (TCP o UDP) da utilizzare non è libera dall'applicazione, ma è insita nel protocollo applicativo.
Se scelgo TCP, chi sta sopra può fare affidamento su una connessione end-to-end affidabile: una volta che invoco il servizio TCP, attraverso la rete si stabilisce una specie di circuito vituale che stabilisce tra A e B un tubo di comunicazione senza perdite, nel quale tutti i BIT sparati da A, vengono consegnati nello stesso ordine e nello stesso ordine a B. E' una connessione sicura e bidirezionale. Il servizio si dice anche byte-oriented: l'unità di misura di dato consegnato è l'ottetto, il byte. Non c'è nessuna garanzia su velocità di trasmissione e ritardo, ma c'è sull'affidabilità. Questa garanzia si offre usando dei meccanismi di controllo di ritrasmissione, che fanno si che ogni qualvolta che l'entità che riceve dei dati si accorge che c'è stata una perdita, implementa dei meccanismi di recupero di queste situazioni di errore, e quindi il dato viene ritrasmesso fino a quando non arriva a destinazione correttamente. Questo però non ci dà nessuna garanzia sul ritardo e sulla velocità di trasmissione. Questo servizio è utile per trasferire le informazioni in maniera affidabile. Posso evitare di implementare il recupero di errore negli strati successivi. Per alcune applicazioni questa necessità di affidabilità garantita non c'è quindi si usa l'UDP che è minimale, ricorda molto IP.

(*slide 10*: comunicazione attraverso un protocollo richiesta/risposta.)

Reti di Calcolatori - 04/10/2012

Lezione 4. Livello trasporto: protocollo HTTP

Quando si descrive un protocollo applicativo dobbiamo definire il protocollo di trasporto cui si appoggia (TCP o UDP). L'UDP consegna i messaggi in maniera non affidabile. Il protocollo TCP, pur essendo implementato su una rete di commutazione di pacchetto a datagrammi, realizza a livello 4 un protocollo di astrazione: c'è un canale di comunicazione bidirezionale affidabile nel quale tutte le informazioni trasmesse come sequenze di byte vengono consegnate esattamente, nella stessa sequenza e senza perdite, nell'altra estremità. Le estremità non sono macchine, ma processi. Questo modello di servizio che realizza quest'astrazione si realizza attraverso opportuni meccanismi di controllo di sequenze, perchè l'informazione trasmessa può subire perdite, cambi di sequenza, che non sono compatibili con i servizi. Perchè si possa stabilire una connessione tra i due end-point, occorre una fase preliminare di instaurazione di connessione che si esplica attraverso lo scambio di 3 pacchetti:

Three-Way-HandShake.

Fase iniziale (three way), seconda fase (hand shake) se mi metto ad osservare le due entità che comunicano, vedrò viaggiare prima da A a B, prima due pacchetti speciali di controllo, con l'arrivo del 3° pacchetto si considera instaurata la connessione e dal quarto pacchetto iniziano i dati.
Il processo che chiamo A prende l'iniziativa a instaurare la connessione, e rispetto al protocollo applicativo è quello che prende il ruolo di client, mentre il server (in questo caso B), deve essere sempre disposto ad accettare servizi dal client.
Un protocollo applicativo definisce come sono fatti i messaggi che si scambiano. Questi messaggi si scambiano su una connessione affidabile TCP che viene instaurata preliminarmente. Inizialmente possiamo immaginare che ci sia un'identificazione.

Requisiti di alcune applicazioni. (*slide 7*)

Le applicazioni non hanno tutte gli stessi requisiti rispetto al servizio di comunicazione richiesto dalla rete: esistono applicazioni che richiedono che l'informazione avvenga senza perdita, ma sono più elastiche rispetto al requisito di bandwidth (quantità di bit al secondo che mediamente riescono a trasferire le due parti che comunicano). Questo bit-rate che i due terminali riescono a trasferire dipende sia dalla caratteristica che dal carico della rete. Se un'applicazione è elastica per il bandwidth vuol dire che non cambia la riuscita dell'applicazione in base a questi parametri. Per i flussi multimediali, per esempio, ho invece dei requisiti in termini di throughtput del collegamento. Se non è sufficientemente elevato i dati arrivano ma non nel tempo richiesto alla riproduzione continua: il file viene trasferito in un tempo troppo lungo.

Ci sono applicazioni interattive particolarmente sensibili al parametro del ritardo, cioè il tempo di trasferimento della rete (dipende dallo stato della rete, dalla velocità e dalla lunghezza fisica del collegamento...) alcune richiedono un ritardo piccolo e altre vogliono che vari in un intervallo di tempo contenuto, cioè il jitter del ritardo deve avere solo piccole variazioni.
Le prime 3 applicazioni della tabella useranno il protocollo TCP, che garantisce la sicurezza, mentre le applicazioni multimediali tendono a utilizzare UDP, che, pure non garantendo l'affidabilità, garantisce la velocità di consegna. La messaggistica istantanea tende ad usare TCP.

Un altro protocollo applicativo importante usa UDP, ed è un esempio di protocollo che si basa su un'interazione richiesta-risposta di tipo semplice: se la richiesta non è mai troppo grande, allora a volte non si usa il TCP, perchè tutta l'informazione sta in un pacchetto e se c'è perdita si fa la ritrasmissione.

Port Number.

E' un concetto che serve a stabilire un SAP (Service Access Point - servizio definito nel modello OSI che permette di collegarsi e scambiare dati con altri utenti collegati al SAP) al quale serve agganciare un'applicazione per ricevere i messaggi ad essa diretti. Quando un messaggio attraverso la rete arriva all'Host destinazione, il protocollo di trasporto in esecuzione deve realizzare lo smistamento del pacchetto rispetto al programma a cui deve essere consegnato, e questo smistamento viene fatto attraverso il numero di porta. Per esempio **l'applicazione server web**, per default si aggancia al numero di porta **:80 TCP**. Il programma web-server si aspetta di ricevere messaggi di richiesta HTTP da un client che glieli invia. Il client HTTP è un programma browser, che a sua volta aggancia la macchina a un port number TCP, che è un numero abbastanza diverso ed è scelto casualmente. L'8080 è usata tipicamente per i server applicativi.

Un server, tipicamente, non può aprire più porte perchè non è necessario. Ma ci possono avere due web-server che possono essere contemporaneamente in connessione sulla stessa macchina su due porte differenti.

Il numero di porta usato dal client è casuale, e viene scelto in base alla disponibilità dei numeri di porta sulla macchina client. Il numero di porta è una risorsa che può essere esaurita, perchè è sempre un numero rappresentato su 16 bit. Allora, il sistema operativo, che gestisce il TCP, ogni volta che un client chiede che venga aperta una connessione, assegna all'end-point un numero di porta che ancora non era stato assegnato a nessun'applicazione, in maniera casuale o progressiva.

Una volta che si realizza il three-way handshake, la connessione è identificata univocamente da una quintupla:

- Protocollo (TCP)
- Indirizzo sorgente (A): indirizzo IP
- Indirizzo della controparte (B): indirizzo IP
- Port Number lato A
- Port Number lato B

Una volta stabilita la connessione, i due processi (browser e web server) possono inviarsi messaggi. Chi invia il primo messaggio è il client, che invia la richiesta. Ad ogni richesta il server risponde sempre con un messaggio di risposta. Per avere altre informazioni per altre richieste, o si fa una nuova connessione o viene utilizzata la precedente. Infatti, quello che può succedere, è che una volta realizzato questo scambio, la connessione viene abbattuta. Quindi se c'è bisogno di una nuova connessione, si fa una nuova richiesta-risposta.

Protocollo HTTP 1.0. (*slide 12*)

L'informazione fondamentale inviata nel messaggio di richiesta è una URL che serve a identificare in maniera univoca un singolo oggetto, cioè serve a chiedere che il server invii il contenuto di un oggetto al client (copiare il contenuto di un file). Questo protocollo ha la caratteristica di essere stateless: nel client non rimangono informazioni sui messaggi mandati precedentemente: non c'è un concetto di sessione.
L'URL è una stringa stringa (tutto il protocollo è testuale, i messaggi sono una sequenza di codice ASCII) che rispetta una data sintassi, dove le parti racchiuse da parentesi triangolari sono opzionali. I protocolli sono descritti dagli RFC (documenti in cui sono descritte delle regole di sintassi).

C'è:

- uno **specificatore di protocollo**;
- un **nome di host**, che serve a identificare a livello rete in maniera univoca la macchina sulla quale gira il programma web-server con cui vogliamo comunicare. A livello rete gli host sono rappresentati da un numero a 32 bit (indirizzo IP). L'associazione tra nomi simbolici e indirizzi IP avviene in internet attraverso il DNS. L'host di destinazione, a livello HTTP, è tipicamente indicata attraverso il nome simbolico e non l'IP;
- la **porta** (80 di default);
- un **percorso**, cioè una stringa che il server deve essere in grado di associare ad una risorsa, e questa associazione avviene attraverso il file system del sistema operativo, strutturato a forma di albero. Scrivere un percorso significa navigare nelle sotto directory del filesystem per identificare un singolo file. Il path col cancelletto (#) indica un frammento: il client dice al server che non vuole la risorsa dall'inizio ma a partire da un certo punto;
- **:string**, sta a significare una **query-string**: un modo che consente al client, attraverso un opportuno formato, di passare determinati dati. Questo accade quando il client ha inserito dei dati in una form, ad esempio, e questi dati devono essere inviati dal client al server. Non è usato per trasmettere dati sensibili come le password.

La versione più usata è la 1.1 del '99. La prima stabile è la 1.0 ('96) e prima ancora si usava la 0.9 che aveva un formato di messaggi un po' diverso. Nella 1.1 cambia l'associazione tra interazione HTTP e connessione e altre cose che riguardano la gestione del caching.

Trasferimento di pagine web.

La pagina web, nella versione tradizionale, è un sistema di ipertesti che sono costituiti da informazioni scritte in formato testuale, codificato in **HTML (*HyperText Markup Language*)**, arricchite da ulteriori contenuti multimediali, che sono visualizzati dal browser insieme al testo, ma non sono memorizzati insieme al testo, sono memorizzati in file diversi, dove ogni file corrisponde ad un contenuto. L'HTML dice come comporre graficamente una pagina. Dice anche che in certe parti del documento devono essere introdotte delle imamgini, dove ogni immagine è associata ad un proprio URL. Le immagini dovranno essere recuperate dal browser attraverso tante connessioni HTTP. Queste interazioni HTTP possono avvenire con lo stesso server o con server diversi (pagine che contengono banner pubblicitari), e quindi si richiede al client di stabilire tante connessioni HTTP con tanti server diversi.
Per stabilire la connessione occorre che le due parti stabiliscano una connessione TCP, e questo avviene con il **three-way hand shake**.
Questo modo di procedere è **non persistente**. Ma con questo modo di procedere si spreca tempo. L'invio della risposta del server al client non determina l'abbattimento della connessione, ma c'è un timeout scaduto il quale, se la connessione non è stata usata, viene abbattuta. Questo modo di procedere, invece, è **persistente**.
Il tempo che passa tra l'invio di un messaggio e la ricezione del messaggio di risposta viene chiamato **Round Trip Time**. Il messaggio di risposta è tanto più grande quanto più è grande l'oggetto che deve essere trasmesso. Se il file è grande non si riesce a trasmetterlo in un unico pacchetto, ma occorre una sequenza di pacchetti che devono essere trasmessi dal server al client.

Non tutto il messaggio è sempre testuale, perchè il messaggio di risposta può contenere una sequenza di byte che devono essere incapsulati nel messaggio di risposta che possono significare qualsiasi cosa. L’intestazione del messaggio HTTP, invece, è sempre testuale. Nella risposta c’è una descrizione del tipo del messaggio, perché il client che riceve l'informazione deve sapere come trattarla, perchè deve sapere come decodificarla opportunamente. C'è un header che contiene il

content type.

Esempio: **(slide 21)**

`GET /pat/pagename.html HTTP/1.0`	*Serve a definire il tipo di messaggio.*
`User-agent: Mozilla/4.0`	*Ci dà informazioni sull'utente: dice che browser sta usando.*
`Accept: text/htm, image/gif, image/jpeg`	*preferenzialmente ci si aspetta una cosa del genere.*
`Accept-language: it`	*Il server preferisce la versione italiana*
`---riga vuota---`	*Indica la fine del messaggio*

Messaggio di richiesta.

Il messaggio di richiesta è strutturato in questo modo. Dopo la prima riga, ci sono altre righe che forniscono al server informazioni aggiuntive. L'**accept** serve a dire che in default il browser accetta in risposta vari tipi di decodifica. Inoltre, il browser si aspetta di ricevere una risposta in italiano. Il messaggio di richiesta termina con una **riga vuota**, che indica la fine del messaggio, e ogni volta che andiamo a capo, troviamo la sequenza di due codici ASCII particolari, che, in coppia, vengono identificati come **fine riga**.
Il campo **method** contiene uno dei messaggi di richiesta (es.: get, put, post, end). **SP** rappresenta uno spazio vuoto e **CR/LF** sono i due codici ASCII particolari che vengono identificati come fine riga. Poi, c'è una serie di righe di intestazione, col formato nome del campo:valore. Infine c'è il **payload**.
Nel messaggio di tipo GET non c'è il payload, ma ci sono altri tipi di messaggi che hanno bisogno di payload (ad esempio, l'upload di un documento). Nel server c'è qualcosa che recupera il file e lo elabora.

Messaggio di risposta.

Il messaggio di risposta è simile al messaggio di richiesta. La prima riga ha un **codice numerico** che identifica il tipo di messaggio. La stringa di testo che viene subito dopo è una stringa testuale che sta a identificare che **tipo di risposta** è: una risposta positiva oppure c'è stato un errore e se l'errore è da attribuire al client o al server. La prima cifra (*centinaia*) indica la classe del messaggio di risposta. Il messaggio può essere ulteriormente specificato dalle due stringhe successive (es. HTTP/1.0 200 OK) poi l'OK indica la risposta positiva o un eventuale errore attribuibile al client o al server.
Ci sono una serie di linee di intestazione: la **data corrente per il server**, la **versione di web-server** che sta in esecuzione, la **data di ultima modifica del file** (che indica quand'è l'ultima volta che è stato modificato l'oggetto). Quest'informazione è utile perchè il client in questa maniera può fare a meno di andare a chiedere sempre gli stessi file: i programmi browser hanno una cache locale in cui mantengono una copia di tutti gli oggetti che raccolgono sul web. Oggetti spesso recuperati non vengono sempre richiesti al server, si vede se la copia in cache è aggiornata o no. Ci sono alcune parti di un sito che cambiano raramente e non devono essere sempre trasmesse. L'header **content-lenght** sta a identificare la lunghezza del payload e l'header **content-type** sta a identificare che l'oggetto dentro al payload è un testo in html. Il browser che riceve questo messaggio, estrae dal paylod il file e sa che deve essere interpretato come file html.

Esempio di codici di stato.

- 200 OK: risposta positiva.
- 301: oggetto richiesto è stato spostato. Poi in un header differente è specificato dove.
- 400: richiesta sbagliata.
- 404: file non trovato.
    - Le risposte di tipo 400 sono errori del client.
- 505: La versione del protocollo HTTP non è supportata dal server.
    - I messaggi di errore di tipo 500 indicano che il problema è del server (es. Server sovraccarico).

La connessione HTTP.

HTTP 1.0.
Nel caso di connessione non persistente, ogni richiesta-risposta corrisponde all'apertura e alla chiusura di una connessione. Prima di ogni interazione c'è il three way handshake. C'è un'inutile perdita di tempo.

HTTP 1.1.
Si può risparmiare tempo se si evita di abbattere e ricreare la connessione ogni volta che c'è una richiesta-risposta. In HTTP 1.1 la connessione viene mantenuta persistente. C'è un timeout sia lato client che server allo scadere del quale la connessione è abbattuta. Altrimenti viene usata la stessa connessione. E' usata per gestire una sequenza di transazioni consecutive.

HTTP 1.1 con pipelining.
Esiste una modalità di interazione con pipelining: il client può inviare due messaggi di richiesta consecutivi, sapendo che il server risponderà con due risposte sicuramente nello stesso ordine, e ordinate arrivano anche le risposte. Grazie all'informazione content-lenght, sempre inviata nel messaggio, il client, quando si vede arrivare sul tubo un messaggio sa quanti byte deve contare per il payload. Poi tira una riga e sa che finisce il primo messaggio

Lo scambio di messaggi.

Messaggi di richiesta (metodi):

- **GET**: Il client chiede al server di ricevere una richiesta identificata da un URL. Lo stesso invio di informazioni con piccoli campi può avvenire con la GET attraverso la query string. Se devo trasmettere dati poco pesanti, posso appendere a destra una sequenza opportunamente codificata di nomi di campo = valore.
  Si possono usare messaggi di **get condizionali**: ci sono alcuni header che servono a specificare delle condizioni che, se soddisfatte, devono richiedere la trasmissione dell'oggetto da parte del server. Il server può rispondere in due maniere: se l'oggetto è stato modificato, risponde con una nuova versione dell'oggetto e la nuova versione della modifica, se invece l'oggetto non è stato modificato risponde con un codice speciale che sta a significare che l'oggetto non è stato modificato.
  Può essere assoluto, condizionale o parziale. Ci sono degli header che specificate delle condizioni richiedono l'oggetto da parte del server. Es: if-modified-since. E' una specie di head con la get.
- **HEAD**: Una specie di ***metodo get fasullo***: serve solo a far finta di chiedere una risorsa: col

metodo head chiede al server: che cosa faresti se ti inviassi questa richiesta? Serve a vedere se una certa URL esiste o no e a sapere quante volte è stato modificato il file. Il browser, se quest'oggetto già cel'ha nella cache locale, chiede l'oggetto al server con HEAD e non con GET, e in questa maniera valuta se l'oggetto già esiste in cache e se la versione che sta in cache è ancora valida.

- **POST**: serve a trasferire informazioni dal client al server, ma l'URL nominato nel messaggio di richiesta non è un nuovo URL che identificherà la risposta successivamente, ma sta a identificare un programma che deve farsi carico di elaborare l'informazione così richiesta. Generalmente si usa il metodo POST invece del PUT, per postare dei contenuti. L'URL indica un programma che deve essere eseguito dal lato server perché deve creare l'operazione. Le informazioni sono trasmesse nel body del messaggio, si compila una form. Il programma che gira lato server estrae i dati dal payload. Come vengono trattati i dati non dipende dal protocollo, l'elaborazione dipende dai programmi lato server. Non viene creata una nuova risorsa.
- **PUT**: duale di GET, ma è quello mai utilizzato nella realtà. Con GET il client chiede al server di ricevere una certa risorsa. Con put, il client chiede al server di creare o sovrascrivere una risorsa associata a un dato URL. Se produce successo fa si che la risorsa sia disponibile e richiedibile con una GET successivamente. Consente di uploadare nuovi contenuti sul server web. Ma non si usa il protocollo HTTP generalmente.

Il protocollo specifica solo come avviene la trasmissione, l'elaborazione delle informazioni è responsabilità delle applicazioni lato-server.

Wireshark.

Consente di ispezionare il traffico che viaggia attraverso le schede di rete. Il programma cattura il pacchetto e conoscendo il formato dei protocolli standard interpreta il pacchetto. E' utile per protocolli testuali. C'è anche una vista strutturata del pacchetto, diciamo.

[H4 = TCP o UDP]

Reti di Calcolatori - 05/10/2012

Lezione 5. Il protocollo HTTP

Wireshark consente di catturare il traffico che passa attraverso una scheda di rete.
Io: interfaccia di rete virtuale attraverso il quale si possono controllare le connessoni tra un'applicazione e un'altra sulla stessa macchina. Questa interfaccia di rete è assegnata ad un determinato indirizzo: 127.0.0.0. E in qualunque sistema operativo c'è un'interfaccia di questo tipo.

I primi tre pacchetti registrati sono quelli relativi al three way hand-shake.
La prima riga contiene il messaggio che il programma wireshark interpreta come protocollo HTTP. C'è l'indirizzo di sorgente e destinazione, e tra i vari campi che troviamo nell'intestazione di livello 3, c'è scritto che il protocollo HTTP di livello 4 che è contenuto nel payload P3. Questo è importante perchè a livello destinazione, quando il pacchetto arriva a IP,quest'ultimo deve decidere se mandare il pacchetto all'entità TCP o all'entità UDP. C'è scritto, quindi, che il protocollo 4 è TCP. Quando poi vado ad espandere l'header TCP, vedo che il destination port è :80. Al livello 4 c'è una prima riga di intestazione e i due caratteri (/r/n) che identificano la fine riga. Poi ci sono una serie di header che si possono classificare in header generici di richiesta, header generici del protocollo... Alcuni sono tipici della richiesta, es. User Agent (Mozilla, Windows NT 5.1 – cioè associata a Firefox su WinXP).

Payload: C'è una prima riga di intestazione, poi ci sono una serie di header, che si possono classificare in header generici di richiesta, header generici di protocollo e altri. Alcuni di questi header sono specifici per la richiesta. Poi ci sono altri header che danno informazioni su cosa si aspetta di ricevere il server. L'header **connection: keep-alive** attraverso il quale il client dice il server che vuole che la connessione TCP non deve essere abbattuta ma deve rimanere attiva per un po' di tempo per usare, quindi, una connessione persistente.
L'header **referer** sta a indicare qual'era l'URL che appariva nel browser nel momento in cui precedentemente è stata fatta la richiesta (magari la pagina è stata raggiunta attraverso un link).Anche se l'informazione è inutile ai fini del protocollo, in quanto l'HTTP è stateless, in questo modo il server può tracciare come il client sta viaggiando all'interno del sito, che è un'informazione che viene loggata nel server per un'analisi a posteriori e per riferire al developer del web-server alcune informazioni.
Un campo significativo è quello **host**: consente ad un unico programma web-server che è in esecuzione su una macchina di gestire vari siti, situati in due cartelle disgiunte. Si gestiscono contemporaneamente questi due siti grazie al campo host. Il messaggio di richhiesta contiene lo specificatore di host e, usando questo campo, il web-server può discriminare quale file deve essere preso. E' una specie di demultiplexing che si realizza a livello applicativo. In alternativa, si possono anche far girare due processi diversi che ascoltano su due porte diverse, però la differenziazione non è più trasparente all'utente finale, perchè l'utente finale deve specificare la porta a cui vuole accedere.

Un file HTTP viene diviso in più pacchetti, e come vengono divisi questi pacchetti è una cosa fuori dal controllo dell'applicazione che ha creato quel file. Il TCP prende questa quantità di informazioni dall'HTTP, e sulla base di meccanismi interni decide quanta di questa informazione deve entrare in un pacchetto, quanta in un altra e così via. Il client, in realtà non deve fare solo una ricezione, ma deve fare tante receive finchè non riceve tutto il documento che era stato spacchettato.
A livello TCP si incollano il payload dei vari pacchetti.

I metodi POST e PUT. (*slide 30*)

Il messaggio **POST**, a differenza del GET, oltre all'header HTTP ha anche un corpo, che contiene le informazioni che il client sta inviando al server. Il content type è PDF e nel body c'è il contenuto del file. Il metodo **PUT**, invece, consente di creare una nuova richiesta, ma non viene quasi mai usato.
Il codice di risposta è un numero su 3 cifre decimali, dove la 1° cifra decimale sta a indicare la classe di risposta. Ad esempio, i messaggi di tipo 300 sono messaggi di ridirezione: il server non può dare le informazioni al client, ma gli dice in quale altro server può trovare quell'informazione.

Header.

Ci sono header generali, ma esistono anche header specifici di risposta o di richiesta. C'è, ad esempio, l' Header **Server**, usato nei messaggi di risposta, che serve a codificare il tipo di server. Gli header generali li possiamo trovare sia nella richiesta che nella risposta (data, ecc...). Altri header sono presenti quando nel messaggio c'è un body e servono a dire il main-type dell'informazione trasmessa nel messaggio, oppure la lunghezza dell'informazione. Tra questi c'è l'header **Last-Modified**, che dice quando quel documento trasmesso è stato modificato per l'ultima volta, e serve per decidere se la copia posseduta in cache dal client è ancora valida o no. Il client mantiene in una cache locale il contenuto dell'oggetto e in una specie di metadato scrive anche quando è stato modificato per l'ultima volta. C'è poi l'header **Expires**, che indica la data dopo la quale la copia non è più considerata valida. Serve per evitare di mantenere nella cache oggetti troppo vecchi. Quando il client trova una versione scaduta controlla per vedere se è ancora valida oppure la richiede al server (di solito).

Cookies.

I cookie sono un meccanismo che è stato concepito da Netscape per superare i problemi che derivano dal fatto che HTTP è un protocollo stateless: il server non è tenuto a mantenere informazioni su connessioni precedenti. Questo è un problema quando si deve creare una sessione di lavoro e il server deve fornire delle risposte al client condizionatamente a risposte che aveva dato precedentemente al client (ad esempio, la selezione di un linguaggio quando apriamo un sito).
Il server ci dà la versione perchè c'è uno stato che non è mantenuto lato server, ma lato client, attraverso un oggetto (cookie), che è una stringa di testo che codifica una certa informazione di stato. Quando il server richiede che venga mantenuto uno stato, genera questa stringa di testo e invia questa stringa al client. Il client deve mantenere questa stringa e ogni volta che sono delle interazioni tra client e server deve inviarla al server. Il client deve mantenere quella stringa e ritrasmetterla ogni volta. Il client non sa cosa vuol dire, il server sì.
Se nei cookie sono immagazzinate informazioni pr[...]

Quando si usano i cookie, il server web, che di per sé è stateless, si applica una logica applicativa che gira esternamente al server stesso. Il client invia la richiesta, il server invia la richiesta al programma che genera il cookie specifico, il cookie viene inviato al server, che nel messaggio di risposta invia anche il Set-Cookie. Il client, poi, in ogni successiva iterazione invia la richiesta e il cookie al server.
Questa stringa viene analizzata e il server fa una risposta customizzata per lo specifico client. Come vengono usati i cookie dal server dipende dalla sua logica applicativa, non è standardizzato. La cosa standardizzata è la trasmissione inalterata del cookie, che però non è strettamente obbligatoria: il browser decide se mantenere i cookie o no, ma alcuni siti ne hanno bisogno per f11unzionare.

Web caching.

Non tutti gli oggetti si possono mantenere i cache: esistono degli header che scrivono quali informazioni si è autorizzati a mantenere i cache, di solito vengono mantenute solo informazioni

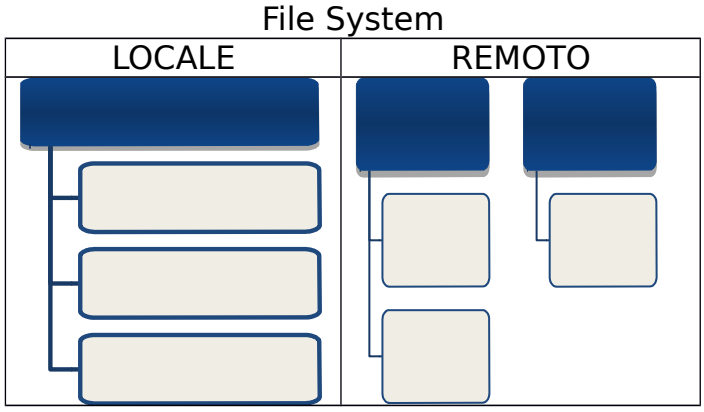
come immagini, testo, etc.. per velocizzare la navigazione. Quest'azione di caching, oltre ad essere esplicata nel client stesso, può essere intercettata da un'altra cache. In questo caso, si parla di **proxy**. Un proxy riceve una get dal client, comportandosi come un server, e la invia al server, comportandosi come un client, per poi ricevere la risposta dal server e inviarla al client. È un oggetto che si comporta come un client rispetto al server e un server rispetto al client. Fa, insomma, da mediatore rispetto a client e server. Ci sono proxy che sono piazzati per filtrare contenuti, come meccanismo di protezione. In altri casi, il proxy può modificare il contenuto o può, semplicemente, fare da cache. Se ho, ad esempio, tre client che chiedono lo stesso contenuto al server, una volta ottenuto per primo, la cache lo mantiene. Quando successivamente si richiede la stessa risorsa questa viene fornita direttamente dalla cache invece di ricolloquiare col server. Questo può comportare meno traffico e una velocizzazione dell'iterazione, perchè la cache sarà fisicamente più vicina del server. Spesso l'uso di questi oggetti avviene in reti aziendali.
A volte, il proxy può anche alterare i contenuti.
Il problema che si pone quando si mette una cache in mezzo è come fare affinché le richieste del client vi arrivino e non vadano al server. I proxy lavorano di default non sulla porta 80 ma su un'altra. Dico al browser che tutte le richiste HTTP che escono devono essere dirette al proxy specificato, che smisterà le richieste ai server corrispondenti.
Quando invece il proxy è messo dall'ISP, quest'azione di redirezione del traffico verso i proxy deve essere trasparente al client, e questa cosa richiede l'intercettazione dei pacchetti che escono verso internet e l'inoltro trasparente verso i proxy.
Quando abbiamo superato la data di scadenza del pacchetto la cache non lo rende disponibile (sia la locale che quella in mezzo). Però non è detto che un oggetto scaduto sia effettivamente stato moltiplicato. Il client richiede l'oggetto con un header condizionale. Se l'oggetto non è cambiato manda l’errore 304: non modified. Anzi può anche essere modificata la data di scadenza. Se è cambiato viene inviata la nuova versione del documento.

Reti di Calcolatori - 09/10/2012

Lezione 6. Protocolli applicativi: FTP e SMTP

Protocollo FTP.

Supporta il trasferimento di file. Oggigiorno è un protocollo molto meno utilizzato. Una volta si aveva l'esigenza di accedere a sessioni di lavoro tra macchine connesse in maniera remota. Invece di navigare nel file system del computer remoto è stato sviluppato il FTP, che consente di trasferire file dal file system della macchina remota a quello della macchina locale o viceversa (con operazioni di get e put). A volte quest'esigenza si sviluppa all'interno di reti locali, e anche se i sistemi operativi hanno messo altri meccanismi (cartelle condivise), mentre questo protocollo supporta l'attività di trasferimento di file attraverso una sessione di lavoro che viene instaurata dall'utente verso la macchina remota e che viene preceduta da una fase di autenticazione, per definire i diritti di accesso di un utente ad una macchina. Questo protocollo è definito da un apposito RFC, e la sua particolarità è che vengono instaurate tra client e server due connessioni TCP, una che consente un dialogo di controllo tra client e server, un'altra su cui viaggiano i dati veri e propri. Questo protocollo si instaura tra due entità client e server.
C'è un server FTP che si mette in ascolto sulla porta 21 e aspetta un comando. Esistono sia client FTP a linea di comando e sia client FTP con interfaccia grafica.



Il protocollo già è stato utilizzato per avere la vista di cartelle e sottocartelle del file system remoto.

Client FTP da linea di comando.

I programmi client di tipo command line ricordano una shell di comandi di un vecchio sistema operativo. Possiamo dare una serie di comandi per avere informazioni e per effettuare il trasferimento tra vari computer.
La vista del file system remoto può essere parziale, e ciò dipende dai privilegi dati ad un certo utente.

Ci sono 4 comandi fondamentali:

- **dir**: ci dà tutte le sottocartelle di una determinata cartella
- **cd**: per spostarmi tra le cartelle
- **get**: preleva il file dalla macchina remota
- **put**: invia il file alla macchina remota

A questi 4 comandi fondamentali corrispondono 4 messaggi di richiesta da client a server. Con questo protocollo non possiamo lanciare programmi sulla macchina server.
Ci sono altri comandi che non danno interazione diretta tra client e server, perchè come posso cambiare la directory corrente della macchina remota, posso cambiare la directory corrente della macchina locale, e quindi non ho interazione tra client e server. Questa cosa si fa con il comando di **lcd**, che serve a cambiare la directory locale. In generale, nella command line interface, i comandi dati nel formato ***>!ls***,vengono dati alla macchina locale. Con questo comando vedo il contenuto della directory locale.

Controllo e dati viaggiano su connessioni separate. La connessione TCP che serve al controllo viene aperta e rimane sempre aperta. La connessione TCP che serve all'invio di dati viene stabilita ogni volta che si fa un trasferimento di file: se il client chiede un trasferimento, i dati vengono inviati con una connessione dati che, una volta utilizzata, viene abbattuta. In realtà, questo protocollo ha un concetto di stato che deve essere mantenuto dal server, perchè c'è un concetto di **sessione di lavoro**. La connessione di controllo viene sempre instaurata dal client, invece, la connessione dei dati può essere di due tipi: dal client al server e dal server al client.

Il protocollo è testuale, cioè i comandi e le risposte sono visti come testo ASCII.

Comandi e codici del protocollo:

- **User e Pass**: il client richiede di autenticarsi. Il server risponde con un messaggio di risposta che ha un formato simile alle risposte HTTP.
- **List**: comando che serve a restituire la lista dei file presenti nella macchina remota. E' quello che un client fa automaticamente quando c'è l'interfaccia grafica.
- **Get** e **Put**: sono i due messaggi del protocollo, che hanno i corrispondenti get e put per il protocollo a comand shell.

Con il web ho un sistema di ipertesti tra di loro collegati, mentre con FTP innanzitutto devo poter avere accesso alla macchina remota e, se la macchina remota non mi conosce, non posso accedere ai file della macchina remota. Questo è vero anche se molti server consentono l'accesso anonimo, cioè sono configurati per consentire l'accesso al file system di un computer locale a un qualsiasi utente.
Il client grafico tipicamente si installa esplicitamente.
[ftp://ftp.unina.it] [ftp://user:password@ftp.unina.it]

[Esempio slide 9]

Non si possono aprire sessioni di lavoro FTP su una certa macchina se su quella macchina non è in esecuzione FTP. Se faccio quest'errore, il collegamento FTP fallisce, perchè dà un comando di errore di connessione fallita.
Ci sono due tipi di connessioni:

- **Versione Attiva**: Il client apre una connessione di controllo utilizzando un ***ephemeral port*** (*una porta effimera è una porta che viene scelto casualmente da porte tra 1024 e 65535*), mentre il server apre una connessione dati dalla porta 20 verso il client. Effettivamente, il client stabilisce la connessione di controllo 21 del server, e questo numero di porta è casuale. Quando parte il client, si mette in ascolto sempre sulla porta 20. Il server apre una connessione dati verso la porta 20 del client.
- **Versione Passiva**: La prima parte è uguale, ma il server sceglie un porto effimero della connessione dati e la comunica al client sulla connessione di controllo dal server al client.

Così il client sa che deve aprire una connessione dati sul server su quella porta. E' necessaria la comunicazione perchè il numero di porta non è sempre lo stesso, ma cambia di volta in volta.

Tra le due versioni, quello che cambia in maniera sostanziale è chi prende l'iniziativa per stabilire la connessione TCP per i dati. La prima interazione avviene più facilmente, perchè non c'è la comunicazione della porta, ma a volte non funziona, soprattutto quando tra client e server c'è un'entità di tipo **firewall**, che consente di farsi attraversare dal traffico in maniera normale solo in un verso, mentre il traffico nel verso opposto può attraversare il firewall solo se c'è stata una connessione nel verso giusto. Quindi, non si riesce a stabilire la connessione dati da Server a Client. Quindi, tipicamente il client è configurato per supportare per default la modalità attiva e poi, dando un opportuno comando, si può forzare il client a lavorare con la modalità passiva.

Protocollo DNS: consente di collegare la macchina in internet usando un nome simbolico.

Protocollo SMTP

È il protocollo che utilizziamo quando inviamo dei messaggi di posta elettronica. La posta elettronica è stata la prima applicazione che ha determinato un notevole interesse all'utilizzo di internet, ed è anch'essa supportato da protocolli standard. In realtà, il servizio di posta elettronica coinvolge due protocolli differenti:

- **SMTP**: invio del messaggio. Il server SMTP si trova non tanto distante dalla macchina da cui stiamo inviando il messaggio e interagisce con altri server per mandarlo al destinatario.
- **POP3:** quello usato dall'utente quando vuole vedere il contenuto della propria casella elettronica ed, eventualmente, vuole leggere il contenuto dei messaggi.

L'utilità della posta elettronica è che consente una comunicazione asincrona.
Le entità in gioco sono:

- User Agents
- Mail Servers
- Protocollo SMTP

Gli utenti compongono il messaggio, e questo messaggio, opportunamente formattato, viene inviato dal client al server più vicino, che lo recapiterà al server del destinatario attraverso l'indirizzo e-mail del destinatario. Ogni utente che ha una casella di posta ha una mailbox: un contenitore virtuale di messaggi inseriti in questa mail. Quando un messaggio viene inserito nella mailbox è finito il compito del protocollo SMTP e poi agisce un altro protocollo (IMAP o POP3). Lo User Agent serve per inviare e leggere il messaggio, è un programma di posta elettronica. Ora ci si connette tramite browser e ci si autentica all'inbox via HTTP. In questo caso l'interazione dell'utente finale avviene in HTTP e il web server fa le varie interazioni con altri server per interagire con la casella di posta.

Il Mail Server funge da client quando c'è da recapitare, mentre da server quando c'è da ricevere. Quando lavora da server, il server SMTP sta in ascolto sulla porta TCP 25, quindi il client che vuole inviare un messaggio si collega col server TCP, stabilisce una connessione sulla porta 25 e invia il messaggio. Ci sono le fasi di apertura della connessione (hand shaking), invio del messaggio e chiusura della connessione.

(*slide 18*)

La sequenza di caratteri *CRLF.CRLF* è il delimitatore di messaggio, e viene inviato da client a server quando il messaggio è terminato.

Esempio di interazione client-server. (*slide 19*)

RCPT TO: invia il messaggio a un dato destinatario.
. = un punto sulla riga da sola termina.

Formato del messaggio.

C’è un header e un body. L'interazione SMTP prevede un'intestazione iniziale incapsulata nel messaggio inviato al destinatario e il messaggio del corpo, che contiene solo caratteri ASCII. Se vogliamo inviare un contenuto multimediale c'è una codifica particolare. I messaggi che hanno degli allegati vengono codificati tutti attraverso una tecnica particolare, base 64. Quando ci sono allegati il content type del messaggio diventa multipart/mixed. C'è una parte testuale (content type text) e l'immagine jpg inviata da un'altra parte.
Usiamo WireShark (sniffer etheral): si tenta di sniffare il traffico sulla rete. Si trova data, user agent, subject, etc…

Prelievo della posta.

Occorre che siano attivi due server. Ci sono due macchine con un indirizzo condiviso: un server riceve il messaggio e lo scrive dentro l'inbox, quando l'utente vuole leggerlo parla con il protocollo POP3 con un altro server e va a leggere la cartella. Ci può essere una stessa macchina che ha in esecuzione sia SMPT sia POP3. Bisogna dare il nome della macchina per i due server, i nomi sono diversi, ma possono mapparsi con lo stesso DNS sulla stessa macchina. Usare nomi differenti quindi non impedisce il fatto che stiano sulla stessa macchina.
Questo permette all'utente di vedere la lista dei messaggi nella inbox, identificati da un ID numerico, e leggere il contenuto di messaggi. C'è una specie di LIST di messaggi e una GET per leggere il contenuto di un messaggio specifico.

Il messaggio è inviato tramite SMTP al server del mittente: ogni utente conosce il proprio server SMTP. Questo si occupa tramite il DNS di scoprire l'indirizzo server SMTP del destinatario. Tramite il protocollo POP3 (o IMAP) si accede e si legge il messaggio.
L'iterazione tra utente e server SMTP a volte può essere mediata da un server web. L'utente finale usa il browser, parla col web server (sulla porta 80) e c'è un programma che gira nel web server che parla con l'SMTP che si preoccupa di consegnare la mail al destinatario. Si interagisce con un'applicazione web. POP3 prevede un'autenticazione con username e password e poi con un messaggio LIST ottiene dal server una lista di messaggi, con un numero progressivo e una lista di messaggi. Con RETR ID legge il messaggio e con DELE ID elimina il messaggio.

Reti di Calcolatori - 11/10/2012

Lezione 7. DNS

Il protocollo DNS (Domain Name System) è un servizio di risoluzione di nomi, cioè una mappatura di nomi simbolici associati a macchine su indirizzi che identificano gli host a livello rete. Gli indirizzi sono dati da 4 numeri interi da 0 a 255 nella forma a.b.c.d che è una rappresentazione alternativa che in binario si rappresenta in 32 bit. Ogni interfaccia di cui è dotato un host è identificata da un suo indirizzo. Questa situazione è detta di **host multi-homed**: un host che si collega alla rete attraverso più interfacce.
La normalità, per i router, è di due interfacce di rete, o anche 3. Tipicamente, quante interfacce di rete e quale tecnologia viene utilizzata dai link collegati ai link dipendono anche dai router che utilizziamo. I router da una parte hanno l'interfaccia di rete di tipo Ethernet che consente loro di collegarsi ad una rete locale, e poi hanno un'interfaccia di tipo ADSL, che consente loro di collegarsi alla rete internet. Le reti locali (tipo ethernet) hanno, da un punto di vista tecnologico, una tecnologia bus: c'è un bus che collega le varie stazioni. Da un punto di vista di cablaggio fisico, il router ha una presa che si collega ad un doppino telefonico e altre prese che si collegano ai vari computer, dove le varie prese sono collegate con una tecnologia Ethernet. Il router ha un suo proprio indirizzo che gli consente di collegarsi alle altre macchine locali. Un router presente in un servers ISP ha più tecnologie per collegamenti seriali geografici e ha vari collegamenti che collega il router ad altre reti della rete ISP.

Questo servizio viene utilizzato quando un client vuole collegarsi al server. L'applicazione riferisce la macchina su cui si trova il server attivo con un nome simbolico (www.unina.it), dove c'è una macchina su cui gira il web-server. È così per tutti i siti web. Anche per il servizio FTP agisce il protocollo DNS: ad esempio scriviamo “ftp://ftp.unina.it”. In questo caso sto dicendo che voglio usare l'ftp e il server su cui voglio collegarmi è l'host usato dal nome simbolico ftp.unina.it (il nome è scelto così ma potevo anche chiamarlo pippo.unina.it e potevo chiedere a questo server di accedere ad unina.it. A volte si trova anche web.domain.com). Mettere l'ftp è una convenzione, serve a ricordare facilmente.

C'è un altro uso del servizio DNS: quello che viene fatto dalla posta elettronica. Quando diamo un indirizzo email, il server SMTP che riceve la mail sa che dovrà contattare il server SMTP al dominio unina.it. Questo tipo di richiesta ci sta per 2 motivi:

- Occorre conoscere qual è la macchina che ha in carico il servizio di consegna della posta elettronica nel dominio unina.it
- Supponiamo che questa richiesta fornirà a chi l'ha fatta l'informazione che la mail inviata ad un utente debba essere consegnata al server il cui nome simbolico è mail.unina.it, quindi, attraverso una prima interrogazione si saprà che se voglio consegnare un messaggio di posta ad un utente di unina.it devo collegarlo a quel server. Poi si indentifica l'IP.

L'intervento del DNS avviene sempre a livello applicativo: è bene tenere sempre separati i servizi. La caratteristica fondamentale degli indirizzi IP è che devono identificare in maniera univoca l'interfaccia di un host nella rete, cioè non ci possono essere nella rete due macchine che hanno lo stesso indirizzo IP. Il servizio DNS si interroga usando un apposito protocollo che si chiama DNS. Il protocollo DNS utilizza per lo scambio dei messaggi a livello trasporto il protocollo UDP. L'intestazione UDP a ciò che è scritto nel pacchetto aggiunge solo il concetto di port number, che serve soltanto a fare il demultiplexing del pacchetto al destinatario. Usando il port number, il protocollo UDP consegna questo pacchetto al processo server DNS che è in esecuzione su quella macchina.

Questo è un servizio fondamentale e complesso. Consente di fare varie cose, come scoprire ad una macchina della quale conosciamo un nome simobolico se sono associati o meno altri nomi simbolici. Più nomi simbolici possono essere associati allo stesso IP. Tipicamente dà informazioni a chi usa l'infrastruttura, cioè la localizzazione del server.

Un'altra funzione è quella di **supporto al servizio di posta elettronica**: si può conoscere qual'è il server di posta elettronica che ha la responsabilità di prendere messaggi che sono destinati a certi domini.
Un'altra funzione importante è quella di **distribuzione del carico**, cioè si può fare in modo tale che attraverso il servizio, lo stesso nome simbolico sia mappato non solo su un indirizzo IP, ma su un pool di indirizzi IP. Questo accade quando ci si aspetta che il flusso di connessioni sia così elevato che una sola macchina non è necessaria. In questa maniera, se gli indirizzi IP associati al sito sono 5, allora ogni server avrà un quinto del carico che arriverebbe alla singola macchina se fosse una sola. Questo avviene in modo trasparente rispetto agli utenti.

Si potrebbe pensare di costruire un **DNS centralizzato** che realizzi la risoluzione di tutti i nomi, ma questa cosa darebbe troppi problemi. Il DNS entra per quasi tutti i servizi applicativi (non accediamo mai senza usare nomi simbolici ma direttamente con l'IP, è raro). Laddove il DNS non è disponibile verrebbero meno molti servizi fondamentali: single point of failure.
Il tempo che occorre affinché l'operazione sia conclusa inficia le operazioni restanti: è un tempo che si associa a tutti gli altri (volume di traffico). E' conveniente mettere i DNS il più vicino possibili agli utenti finali. Le uniche soluzioni realistiche da prendere in considerazione sono di tipo distribuito. Quindi, si fa un partizionamento delle informazioni mantenute dal sistema DNS, tenendo presente che gli stessi nomi simbolici che utilizziamo posseggono già una gerarchia. La notazione con i punti che usiamo quando scriviamo i nomi di host sta ad indicare che c'è una gerarchia nella localizzazione logica nello spazio dei nomi che si riflette in una gerarchia di server che fanno le risoluzioni. Quando richiedo un'operazione di risoluzione sono coinvolte varie entità dove ognuna delle quali ha […]

I server **DNS** sono **organizzati gerarchicamente**. Il primo livello è il dominio più a destra dei nomi.
I domini sono identificati dalla stringa che viene dopo il punto più a destra (.it, .com, .gov etc.).
L'aggiunta di nuovi domini di primo livello (top level) è una decisione politica gestita dall'ICANN.

Al vertice di questa gerarchia ci sono **13 Root Server**: 13 server, opportunamente posizionati nel mondo, che conoscono i server che gestiscono i domini di livello top (**top-level domain**). I domini di livello top sono in numero di livello limitato. Gestire un dominio significa l'autorità di dare una risposta ad una query data a quel dominio. Il server autoritativo per il dominio unina.it, o è in grado di dare con certezza la risoluzione di ogni nome simbolico o conoscere un server di livello gerarchico inferiore che è in grado di effettuare la risoluzione. (www.unina.it → www.dis.unina.it). Chi mi da la risposta autoritativa può essere o il server autoritativo per unina.it oppure un server che sta sotto che è autoritativo solo per i sottodomini dis.unina.it. Nel dominio dis.unina.it ci sono poi più host. I server che stanno più in alto non devono conoscere tutto, ma possono conoscere anche il nome di un host di livello inferiore che è in grado di effettuare la risoluzione. Le informazioni di risoluzione sono mantenute in una cache locale.

Occorre contattare uno dei top domain per conoscere il nome simbolico di un server autoritativo per il dominio .it, poi bisogna andare dal server autoritativo per il dominio .it e andare a chiedere qual'è il server per il dominio autoritativo unina.it, e poi chiedere l'indirizzo IP della macchina server www.unina.it.

Il server DNS locale è una macchina che si trova il più possibile vicino agli host e si prende il carico di fare l'interrogazione al DNS che può richiedere più interazioni. Allora, per alleviare l'onere di fare queste varie interrogazioni e ottenere il risultato finale ogni volta che si deve fare una risoluzione DNS, in ogni rete locale si colloca un server **DNS** che **non** è **autoritativo** per nessun dominio,ma ha solo il compito di mettersi in mezzo alle richieste che avvengono tra vari client (tipo *proxy*) e prendersi l'onere di prendersi le interrogazioni tra i vari server e dare al client che aveva richiesto il servizio solo la risposta finale. Per questo server passano tutte le mie richieste di risoluzione.

I vantaggi sono:

- Liberare dall'onere della risoluzione il client che gira sull'end system
- Tutti questi server DNS che fanno richieste possono utilizzare delle teniche di caching: quando un server si vede arrivare una richiesta e interroga un altro server, la risposta che ottiene, oltre che mandarla al client, la mantiene in una cache locale, perchè gli può fare comodo.

Il server DNS locale è quello che configuriamo nella macchina con la risoluzione di rete. È quello cui il client rivolge tutte le richieste di risoluzione.
I server DNS non fanno nessun controllo sull'indirizzo da cui proviene la richiesta, sono libero di mettere qualunque IP, però se faccio richieste a DNS lontani queste sono più lente, e quindi devo prendere un server DNS che sia il più vicino possibile. La scelta dell'IP del DNS è guidata dalla rete internet (es.: *Telecom, li suggerisce*).

I DNS Root conoscono i server autoritativi per i top level domain e hanno degli indirizzi IP ben noti che sono cablati nella configurazione delle macchine. I top level domain gestiscono tutti i domini di primo livello. Sono etichettati da A a M. I top level domain gestiscono in maniera autoritativa tutti gli indirizzi di primo livello. Il DNS autoritativo è capace di risolvere tutti i nomi all'interno di un dato dominio.

(***slide 14***)

Schema iterativo: a partire da un'unica richiesta scatena interrogazioni successive. Quando si fa una richiesta, le iterazioni sono sempre iterative, però il DNS locale potrebbe (se non vuole gestire le iterazioni) fare una richiesta di tipo iterativo mettendo un flag: chiede al top level domain di avere direttamente da lui la risposta. Allora è lui che si occupa di richiedere le risoluzioni. Questo è un tipo di gestione delle query, detto ricorsivo, che sposta le richieste dei nomi al top level domain. Però questo genererebbe troppo carico sui server radice.
È chi fa la richiesta che decide se mettere il flag, è chi invia la richiesta che decide come ottenere la risposta. Così è più difficile adottare strategie di caching: ritorneremmo allo schema centralizzato. Non esiste un meccanismo esplicito di invalidazione delle cache, nonostante esista un'informazione di scadenza: se la risoluzione cambia prima della scadenza non si può dire che deve essere cambiata prima. C'è una latenza governata da un meccanismo di time out. Anche i client mantengono una **cache** locale, di livello **resolver** (programma che si occupa di fare le richieste al DNS locale).

Nslookup: indica qual è il server a cui si fanno richieste, è l'indirizzo dell'interfaccia di router con cui si parla dal portatile. Il router manda le richieste di risoluzione ad un indirizzo locale.

Cosa memorizza un DNS?
Ogni server DNS mantiene le informazioni dette **Resource Record**: è una quadrupla di elementi

che contiene Nome, Valore, Tipo e TTL. Il significato di Nome e Valore dipende dal campo tipo, e i valori di tipo sono 4:

- **A**: consente di sapere che un nome è collegato ad un IP.
- **NS**: contengono, invece, l'informazione dell'indirizzo IP che è autoritativo per un certo dominio, quindi nel campo nome c'è il dominio, mentre nel campo IP c'è il server autoritativo.
- **CNAME**: Nel campo valore c'è il nome canonico che identifica l'host e nel campo nome ci sono dei nomi alternativi che si possono usare per identificare in maniera alternativa l'host identificato preliminarmente dal nome canonico. Attraverso questi tipi di campi, il DNS può mantenere l'associazione tra nome canonico e nomi alternativi
- **MX**: Mantengono l'informazione del server che fornisce la posta per un certo dominio. Nel campo nome c'è il dominio di posta e nel campo valore c'è il nome dell'host del server associato al nome.

I messaggi DNS sono inviati all'interno di singoli messaggi DNS. Il TCP offre un'astrazione a livello superiore di tipo tubo. Il DNS offre a livello superiore un servizio di consegna di messaggi. Il messaggio UDP viene messo in un pacchetto e viene consegnato (meno affidabile).
Occorre un meccanismo che mi consenta, ad ogni messaggio di richiesta inviato, di associare un messaggio di risposta. Quando arrivano le risposte, devono associargli alle relative richieste e allora, a questo scopo, c'è un campo di identificazione che serve ad identificare ogni coppia richiesta-risposta. Ci sono dei bit attraverso i quali si posso arricchire sia le richieste che le risposte, e nei messaggi di risposta si apprende se quella risposta è stata prodotta. C’è un campo di identificazione, 16 bit, che identifica la coppia richiesta-risposta. Nei flags ci sono dei bit attraverso i quali si arricchiscono sia richieste che risposte: si chiede un meccanismo di ricorsione iterativo o ricorsivo. C'è un flag per sapere se la risposta è autoritativa o no (cioè si sono prelevate info su una cache o no). Attraverso questi campi, si può costruire un messaggio a struttura variabile: il numero di richieste e risposte contenute nel messaggio è variabile.

Questo pacchetto **BIND** è un'implementazione dei protocolli DNS. Questo pacchetto comprende la parte server, che ha un eseguibile che implementa un server DNS, e la parte client, dove c'è una libreria, attraverso la quale un programma applicativo può richiedere la risoluzione di un nome. Questa oprerazione si fa attraverso una chiamata di funzione.

Attraverso il servizio DNS si possono anche fare query di risoluzione inversa, cioè è possibile conoscere l'indirizzo IP associato ad un certo nome simbolico, e quest'informazione viene sempre mantenuta in un altro file di configurazione che è fatto secondo un altro tipo di struttura (**reverse DNS**). Nei vari file di configurazione usati da BIND c'è il file che contiene gli indirizzi IP dei 13 server ROOT.

Reti di Calcolatori - 12/10/2012

Lezione 8. Tecniche e modelli che servono per superare il modello Client-Server

CDN: Content Delivery Networks

I protocolli, da un punto di vista applicativo, assumono quasi sempre un modello client-server. Cosa si fa quando una singola macchina calcolatore non è in grado di gestire una molteplicità molto grande di client? Una prima soluzione può stare nell'implementazione di una architettura di server distribuita organizzata gerarchicamente, come il modello DNS, che però è un modello architetturale che è stato concepito per gestire il servizio dei nomi anch'essi organizzati gerarchicamente.
Immaginiamo servizi più tradizionali (***accesso al web***), ed è chiaro che in una situazione di questo tipo abbiamo bisogno di una macchina server su cui gira il processo server che abbia sufficienti risorse computazionali, e poi abbiamo anche un problema di opportuno dimensionamento di canale di collegamento che connette il server alla rete internet.
Ci vuole un canale di comunicazione a banda sufficientemente larga. Questo componente qui può rappresentare il collo di bottiglia, che limita le prestazioni del sistema.
Quando abbiamo situazioni di questo tipo, si possono fare due operazioni, combinandole. Se ci si accorge che il collo di bottiglia sta proprio nella macchina server, perchè la macchina server sta lavorando al 100%, possiamo immaginare di splittare il carico su più server, usando, ad esempio, il servizio DNS, che allo stesso nome www.unina.it farà corrispondere più server. In questa maniera c'è un bilanciamento del carico tra queste macchine, che dal punto di vista applicativo sono viste tutte quante con lo stesso livello, ma dal punto di vista degli indirizzi sono macchine differenti.
Queste macchine sono dei server che sono montati in un **REC**, un armadio su cui sono sono montati uno sopra l'altro. Se pensiamo al servizio web, allora c'è il problema di come mantenere coerenti le versioni di sito possedute e fornite dai server. Questo problema può essere risolto, o dando ad ogni server un disco server, ma abbiamo problemi di aggiornamento, oppure spesso si usano delle reti dedicate per la gestione dello storage (**SAN**), che viene messo in condivisione con le macchine del REC attraverso una rete dedicata a larga banda. In alcuni casi, si pouò fare in modo che il bilanciamento del carico tra le varie istanze di server fisico avvenga non solo attraverso il DNS, ma attraverso delle macchine che vengono messe a monte e che fanno da ***load-bags***, che sono una specie di proxy trasparente che intercetta il carico e lo suddivide tra le varie macchine. Quando non posso andare oltre un certo limite nell'aumentare la capacità di collegamento, oppure quando voglio ottenere altri benefici, chi gestisce il servizio web può pensare di mettere nella rete diversi **data-center**, ciascuno dei quali è posto nella rete in un punto diverso. Questo tipo di logica porta alcuni benefici: oltre ad aumentare la capacità di servizio del sistema, ho anche una maggiore robustezza del sistema, e posso ottenere dei benefici di localizzazione del traffico, perchè la rete in realtà è essa stessa è un agglomerato di router e link che hanno una capacità limitata, quindi distribuendo il carico, tutto il carico di domanda proveniente dai client si smisterà secondo certe logiche che tendono a mantenere questo traffico il più possibile locale nella rete.
Questo tipo di smistamento del carico a livello geografico comporta una localizzazione del traffico e comporta anche dei benefici per gli utenti finali, perchè ridurre la distanza sulla rete tra client e server, significa avere **migliore latenza** nella comunicazione. Una migliore latenza, in realtà, si traduce anche in un **maggiore throughtput**.
Questa linea di soluzione è quella che viene usata nelle **CDN** (Content Delivery Network), che sono dei servizi commerciali offerti da un numero di operatori limitato, che si propongono di offrire un servizio a degli utenti commerciali, dove gli utenti sono i fornitori di contenuti, cioè sono coloro che devono mettere in piedi un servizio di tipo web che sia popolare, che non si vogliono preoccupare della gestione della rete. Chi opera la CDN, possiede dei data-center e su questi ospita dei server

che contengono i contenuti che vanno distribuiti, e si occupano dei meccanismi che devono bilanciare il carico sui server. Tipicamente c'è un server master che ha una copia originale e dei server che ne sono collegati per aggiornare i contentuti periodicamente. La coerenza dei dati è mantenuta attraverso meccanismi di gestione. Tipicamente, nelle CDN commerciali, non si tratta di sistemi di cache, ma di […].

Uno degli aspetti fondamentali è come fare in modo che il client venga rediretto su un server invece che su un altro. Questi si chiamano **meccanismi di redirezione**.

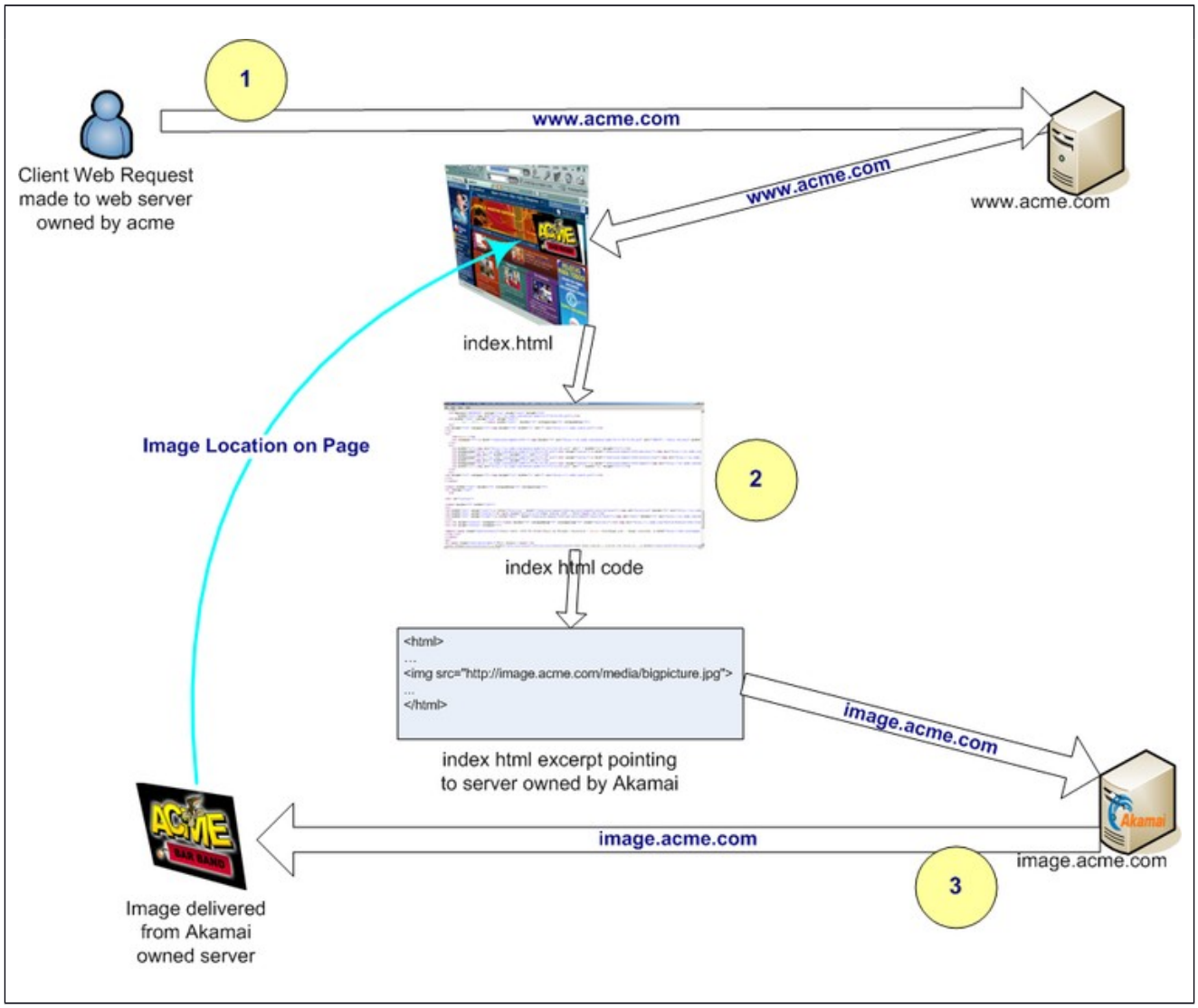
Gli utenti finali, non sono consapevoli dell'intermediazione della CDN, perchè le CDN hanno come requisito di essere trasparenti agli utenti finali. Tipicamente queste repliche si trovano nelle reti di accesso nella rete dell'ISP. Una rete CDN, purchè possa operare facilmente, deve avere dei rapporti commerciali con gli ISP, perchè il posizionamento delle repliche determina l'efficacia del servizio di CDN: i CDN devono basarsi su un insieme di data-center opportunamente costruiti.
A volte, questi servizi CDN sono anche **offerti dagli ISP stessi**, anche se di fatto non è un business per gli ISP perchè gli operatori si basano solo su comunicazioni locali, non su comunicazioni mondiali. Quando interviene una CDN, fa una favore agli ISP, perchè lo allevia da possibili problemi di congestione.
Il **server Master** è il server origine, ed è quello che fa da riferimento nella distribuzione dei contenuti. In realtà il server origine viene gestito da chi produce i contenuti, che poi vengono offerti al gestore del CDN.

I meccanismi di redirezione non sono tantissimi. Lo strumento primo è quello del DNS: occorre utilizzare dei CDN particolari che siano in grado di dare una risposta rispetto ad un altra a seconda dell'indirizzo IP. Per un server di contenuti che utilizza una CDN la risposta di risoluzione sarà differente a seconda anche di chi fa la richiesta. Dobbiamo essere in grado di mappare un indirizzo IP in una data posizione geografica. Ma, in internet, un certo indirizzo a priori non è in grado di esprimere la posizione geografica della macchina, a meno che non si abbiamo informazioni che possiede l'autorità che gestisce gli indirizzi. Esiste un registro in cui sono contenuti gli indirizzi, quindi ogni blocco di indirizzi è registrato e quindi si sa chi ha fatto quella domanda. Sono stati fatti dei database che consentono un posizionamento più dettagliato. Esistono dei database che mi consentono a localizzare un indirizzo IP.

Akamai Server.

È uno dei principali CDN. Questo meccanismo era basato anche sul server originale. In questo caso non è il DNS che fa la correzione, ma restituisce all'user l'indirizzo IP del web server. Il client contatta il server di origine che, o fa una redirezione corretta usando un messaggio di HTTP, o risponde con una pagina html che deve dare, e quindi parte del servizio la fa al client, solo che, poiché la pagina html contiene dei link, che hanno più peso rispetto ai vari componenti della pagina html stessa, e quindi vengono modificati contenendo il nome del server replica che il client dovrà successivamente interrogare per ottenere le immagini, ad esempio. C'è un'operazione di riscrittura dinamica della pagina web.

Akamai effettua un mirror dei contenuti situati sul server del cliente (in genere audio, grafica o video) sui propri server, in maniera del tutto trasparente per l'utente. Sebbene il dominio sia lo stesso, l'indirizzo IP punta a un server Akamai, anziché a quello del cliente. Il server viene scelto automaticamente in base al tipo di file e alla provenienza dell'utente.



Architetture P2P (Peer To Peer)

Le **applicazioni P2P** sono un'evoluzione di questo modo di offrire servizi che decentra in maniera esasperata il ruolo del server: stavolta, il servizio non è più offerto da un'unica entità, ma la fornitura del servizio è demandata ad una molteplicità di macchine, che sono le stesse macchine che gli utenti finali usano per accedere al servizio, quindi non c'è più la distinzione assoluta tra il ruolo di client e il ruolo di server, ma l'applicazione è una sola, che è in grado di svolgere sia il ruolo di interfaccia per l'utente, sia come fornitura di servizio verso altri utenti. Questo modello nasce per il servizio di distribuzione dei file, ma in realtà è stato anche utilizzato per altre tipologie di servizi applicativi (Skype,...). La trasmissione dei flussi vocali tra utenti avviene attraverso una **cooperazione di tipo Peer-To-Peer**. Lo stesso modello è stato usato per la distribuzione, in tempo reale, di stream video, che consentono di accedere ad uno stream di un flusso video.

Si possono conferire caratteristiche come:

- **Scalabilità:** il flusso di informazioni viaggia nella rete e non si concentra in un unico punto, ma si distribuisce tra vari host.
- **Robustezza:** se ho un servizio offerto da un unico server, e questo server va giù, non ho più

il servizio, ma nel caso in cui ci sono più server con tecnologia P2P non ho questo problema.

E' un'esasperazione del concetto di server distribuito. Questa responsabilità viene anche diminuita, perchè quando, è accettato di default che un server nella rete vada giù in un qualsiasi momento. In realtà, siccome i peer hanno un senso di responsabilità che è quello degli utenti che lo usano, si può assistere ad altri fenomeni, come comportamenti maliziosi o egoistici.

Mano a mano che i peer entrano, aumentano la velocità di trasferimento di file, quindi il server può essere dotato di un collegamento che non è molto grande. Il P2P funziona quando il fattore limitante diventa la capacità di upload del server.

BitTorrent

L'idea fondamentale è quella che il file non viene mai trasferito da un peer ad un altro peer nella sua interezza, ma viene trasferito a pezzi, detti **chunk**, e i peer che dialogano tra di loro, quando aprono una interazione tra di loro, si scambiano questi chunk. Questo è utile perchè si può aggiustare la conformazione della rete al variare della situazione della rete stessa: se entrano nuovi peer si può riorganizzare la rete tenendo conto dei nuovi peer che sono appena entrati nella rete.

Prima di bitTorrent, le applicazioni di scambio di file P2P, tendevano a scambiare tutto il file. Spezzettando i file si può organizzare la rete in maniera più efficiente, se i file sono grandi. Un peer, inoltre, può uploadare il contenuto del file non solo quando il file l'ha ottenuto nella sua interezza, ma potrà fare l'upload di pezzi di file quando avrà assemblato un chunk. Un chunk ricostruito completamente, quindi, potrà essere immesso nella rete a sua volta.

I collegamenti logici tra peer creano una rete P2P, perchè questi peer sono organizzati secondo una certa tipologia secondo la quale non ci sono collegamenti fisici, ma ci sono solo collegamenti logici, e questo tipo di rete viene detta **overlay**: una rete logica creata da una serie di entità applicative che stabiliscono collegamenti tra di loro fatti tipicamente attraverso connessioni TCP. Si crea una rete applicativa che si costruisce sopra la rete fisica, sovrapponendosi ad essa.

Un'altra entità fondamentale è il **tracker**: un'entità di tipo server che tiene traccia degli indirizzi IP dei peer che formano un overlay. I peer che formano uno **swarm** (*sciame*), sono quelli che sono interessati a condividere lo stesso file. Ciascuna rete deve essere organizzata: i peer devono sapere quali altri peer devono contattare, e prendono quest'informazione dal tracker. Quindi, quando un utente vuole entrare in uno swarm, contatta il tracker, prende gli IP che stanno nello swarm, che sono quelli che eventualmente possono farmi entrare nella rete: stabilire un numero di collegamenti e chiedere pezzi di file attraverso questi collegamenti. Ancora prima di contattare il tracker occorre avere un'informazione preliminare: il file quanto è grande, in quanti pezzi è diviso, e l'indirizzo del tracker. Questa informazione è tipicamente registrata in un file **.torrent** che viene ottenuto con altri metodi. Quella mappa è utile perchè, per esempio, nella mappa ci sono registrate informazioni di controllo che consentono al peer di capire se i pezzi del file sono originali o meno, perchè peer maliziosi potrebbero corrompere il file e quindi alterare lo swarm. Per proteggersi, nel file torrent ci sono informazioni di controllo che consentono di capire se quel chunk è come deve essere o meno. Un peer che possiede l'intero file si chiama **seed**, mentre un peer che non contiene tutto il file interamente si chiama **leeck**.

Algoritmo di Chunk Selection.

È l’algoritmo che esprime con quale criterio vengono scaricati i chunk. Ogni peer usa un criterio di

selezione di tipo **rarest first**: i peer dialogano tra di loro per sapere gli altri peer quali file posseggono, allora un peer vede qual'è il pezzo meno distribuito tra i peer e inizia a chiedere quello, con la paura che scompaia dalla rete. Nella scelta delle connessioni che si attivano tra i peer gioca un ruolo l'**algoritmo di tit-for-tat**: un peer invia un chunk ai 4 più veloci, che è un approccio che premia chi è più buono verso di me, e periodicamente queste scelte vengono aggiornate. Ma ogni peer, ogni 30 secondi, seleziona un altro peer in maniera casuale e gli invia dei chunk. E' un comportamento altruistico.

Napster

La prima rete P2P fu **napster**: sistema che nacque per la condivisione di file mp3. C'era un server centralizzato che possedeva i peer e i file che i vari peer possedevano. Quando un utente chiedeva un file, l'utente sapeva quel file da chi era posseduto e iniziava la condivisione. La rete morì per motivi di copyright. La soluzione di unico punto di collegamento è stata deleteria per la rete.
Il server di napster non possedeva il file mp3, ma possedeva una conoscenza che serviva a fare un'azione illegale, quindi il server faceva da complice.

Nella rete g**Nutella**,invece, c'è un server che serve nella fase di **bootstrap**. Il server, però, non possiede informazioni sui contenuti, conosce soltanto gli indirizzi dei peer. Nasce il problema del discovery delle risorse: come si fa a sapere chi ha le risorse che interessano? La rete Gnutella è una grande rete che collega tutti e su questa rete ci sono i contenuti più disparati.
Su questo tipo di reti si possono seguire vari approcci: si può fare un **flooding**, cioè un peer si collega ad un altro peer se ha un file, e se il peer non cel'ha inoltra la richiesta ad un'altro peer. In questa maniera, posso raggiungere tutti i peer della rete, quindi se il file c'è lo trovo, se non c'è sono sicuro che non lo trovo.
Bisogna, però, mettere **meccanismi di limitazione**: si fa in modo tale che il messaggio di richiesta di ricerca possa viaggiare nella rete facendo un numero di salti limitato, e in questa maniera il messaggio di richiesta viaggia nella rete in un intorno limitato. Questo approccio non allaga la rete di ricerca, e posso non trovare il file che mi serve. Si potrebbe far allargare la rete di ricerca fino a quando non trovo il file. In questo modo, se la risorsa è più vicina la trovo prima. Questa propagazione dei messaggi di richiesta e le relative risposte avviene sulla rete overlay.
Normalmente, il trasferimento del file avviene secondo una logica HTTP.

Ci sono messaggi di **ping-and-pong**: vengono inviati da un peer quando vogliono entrare nella rete. Il server ha una lista di peer, dove ogni peer non può accettare ogni altro peer. Un peer, quindi, o accetta una richiesta di aggancio alla rete, oppure può girare la richiesta ad un altro peer, fino a quando il peer appena entrato non trova un peer che è in grado di accettarlo.

Ogni peer invia periodicamente il messaggio di Ping per sondare la rete alla ricerca di altri peer.

- Un peer che riceve un Ping risponde inviando al mittente un Pong.
  - Il messaggio di Pong contiene: l'IP e la porta su cui il peer accetta connessioni, il numero di file condivisi e il numeri di Kb totali condivisi.
- Possono essere inviati più messaggi di pong per comunicare il contenuto della propria peer cache
- Il messaggio di Ping viene inoltrato ai vicini fino a che il TTL non si annulla.

Spesso, c'è il meccanismo di un'organizzazione comunque gerarchica di peer: non tutti i peer sono

uguali. Allora, conviene tenere conto di questo fatto e assegnare dei compiti di maggiore responsabilità ai peer migliori, che vengono detti **superpeer** o **hub**, che hanno il ruolo di aggregare una serie di peer di livello inferiore che gli si collegano a stella, e poi c'è una rete di livello superiore che collega i superpeer tra di loro. Questi superpeer si prendono anche l'onere di immagazzinare le informazioni di connessione dei peer che gli stanno sotto. Così, l'informazione può viaggiare localmente. I superpeer hanno una specie di cache di informazioni sui file.

Skype

Organizzata secondo una rete overlay.
Ci sono 3 tipi di host:

- I **server di login**, per l'autenticazione,
- I **supernodi**, che tendono ad aggregare i vari peer,
- Gli **host ordinari**, che sono i peer che non partecipano attivamente al ruolo di distribuzione dei flussi locali.

In realtà, c'è un problema di distribuzione delle informazioni di presenza: per sapere quali utenti della contact list sono disponibili o meno, quest'informazione è condivisa in modalità peer-to-peer.

Esistono altre applicazioni, che nascono per la distribuzione di flussi multimediali attraverso l'overlay.

I peer di un'applicazione si organizzano secondo un'overlay. Il volume di traffico che un'applicazione P2P è in grado di gestire può essere ingente. Gli ISP guardano alle applicazioni P2P in maniera preoccupata, perchè mentre un'applicazione client-server è abbastanza osservabile, un'applicazione P2P genera un traffico che viaggia secondo direzioni difficilmente prevedibili da parte degli ISP. Quindi, ci sono problemi di gestione del traffico. Può anche accadere che alcuni ISP limitino il traffico dei dati delle applicazioni P2P. In realtà, un modo per controllare la formazione di un overlay c'è, e in bitTorrent sta nell'andare a lavorare sul tracker. Normalmente il tracker dà una lista di peer casuale, ma agli ISP conviene mantenere il traffico locale. Allora, se il tracker gestisce in maniera più efficiente l'associazione della lista, per dare IP locali, ho vantaggi sia dal punto di vista del client, sia dell'ISP.

Reti di Calcolatori - 16/10/2012

Lezione 9. Livello Rete: Protocollo IP

Stack OSI.

Il livello rete si appoggia sul livello inferiore data link che offre un servizio a livello frame tra host connessi. Il livello rete offre al livello superiore un servizio di consegna di pacchetti end-to-end. Il compito del livello rete è quello di creare un'infrastruttura di router collegati attraverso link e decidere l'instradamento dei pacchetti che dal mittente vengono consegnati al destinatario. Questo strato può essere progettato attraverso vari requisiti: si può scegliere di avere una consegna affidabile. Questo non avviene in internet, dove c'è un servizio di tipo **best effort**: senza garanzia di consegna di pacchetti al destinatario. La funzione a livello rete è quella di realizzare una interconnessione di dispositivi quella di interconnettere sottoreti. Il protocollo IP viene anche deto di **internet warming**, cioè connette reti che possono funzionare autonomamente l'una dalle altre per avere una rete internet di scala mondiale.

I Router svolgono una duplice funzione:

- **Forwarding** (***inoltro***), che consiste nel ricevere un pacchetto e copiare questo pacchetto nella coda di uscita di un altri link, in modo tale da far avanzare il pacchetto verso la destinazione. La velocità in cui i pacchetti vengono trasmessi sulla coda dipende dai link e dalla loro capacità di trasmissione.
- **Routing** (***instradamento***), che è alla base della scelta che opera un router, che ha una molteplicità di interfacce, di quale deve essere l'interfaccia verso la quale il pacchetto deve essere instradato. Questa scelta determina il percorso sulla rete che il pacchetto compie per arrivare a destinazione. Il percorso è fatto da link e router. I percorsi che consentono il pacchetto ad arrivare a destinazione sono molteplici, e quindi bisogna operare delle scelte sulla base di una tabella, che si trova all'interno del router, che dice per ogni possibile destinazione qual è il link verso il quale il pacchetto deve essere trasmesso.

Nelle reti a commutazione di pacchetto, il pacchetto contiene un'intestazione, dove c'è un identificativo del destinatario, attraverso il quale il router va a fare una ricerca nella tabella di instradamento e decide su quale link il pacchetto deve essere messo. Questa è una scelta locale: ogni singolo router non ha visione di ogni singolo percorso, ma fa una scelta che instrada il pacchetto un po' più avanti. Dall'azione congiunta dei router, il pacchetto arriva al destinatario.
Può accadere che i router sono configurati in maniera tale che il pacchetto faccia una strada circolare, e così il pacchetto non arriverà mai a destinazione e consuma un numero di risorse elevato inutilmente. I protocolli IP hanno meccanismi che evitano che i pacchetti entrati in un percorso chiuso vivano all'infinito. Ovviamente non si hanno tutte le destinazioni in ogni router, ci sono aggregazioni: un entry va bene per determinte destinazioni.
Inoltre, la tabella di forwarding locale è scritta dall'azione di routing, cioè in ogni router viene eseguito un algoritmo che ha come scopo ultimo quello di andare a compilare le entry della tabella, cioè decidere il percorso del pacchetto.

Le funzioni del livello rete.

Le reti a commutazione di pacchetto possono essere progettate secondo due modelli: reti a datagrammi e reti a circuiti virtuali.
Nelle **reti a datagrammi**, i pacchetti possono seguire, verso la stessa destinazione, un percorso differente e quindi possono arrivare in tempi diversi.
Nelle **reti a circuiti virtuali**, il percorso che devono eseguire i pacchetti viene decisa una sola volta quando vengono inviati e, in questo modo, arrivano sempre nello stesso ordine. La rete stabilisce un

percorso che deve essere attraversato dai pacchetti che viaggiano tutti uno dietro l'altro seguendo lo stesso percorso: non sono più possibili i sorpassi (alcuni possono comunque andare persi a meno di meccanismi di recupero).
L'azione di routing avviene differentemente nei due modelli: nei secondi l'attività di routing viene invocata quando viene richiesta l'istaurazione di un circuito, nei primi l'attività di routing avviene sempre in background. Le due attività di routing e forwarding si differenziano in scale di tempi: l'attività di forwarding deve essere il più veloce possibile, perchè ad un router arrivano anche milioni di pachetti al secondo. Quando arriva un pacchetto al router, deve essere solo in grado di fare l'operazione di ricerca nella tabella e procedere alla copia del pacchetto verso la destinazione. Per questo motivo, le attività di forwarding vengono operate a livello hardware, mentre l'attività di routing può operare su scale più lunghe, perchè tipicamente la topologia della rete non cambia se non lentamente: eventi topologici nella rete sono più rari. Può accadere che un link sia particolarmente congestionato o che un link sia danneggiato ma le dinamiche degli algoritmi di routing sono più lente del forwarding (frazioni di secondo, piccolissime). Quindi, le dinamiche di algoritmi di routing sono più lente.

Lo strato di rete di Internet si basa sul **protocollo IP**, che **stabilisce** una serie di cose**:**

- il **formato dei pacchetti**: com'è fatta l'intestazione del pacchetto IP,
- le **condizioni di indirizzamento**: come gli host sono identificati attraverso degli indirizzi a livello rete,
- le **convenzioni di gestione di pacchetti**: quando un pacchetto arriva come deve essere trasmesso e come deve essere trattato.

Il protocollo IP prende le informazioni necessarie dalla tabella di inoltro. Questa tabella è costruita dagli **algoritmi di routing** che, siccome operano attraverso una cooperazione tra tutti i router, richiedono i router della rete siano in grado di dialogare tra di loro per decidere i percorsi. Allora, esistono dei protocolli di comunicazione che vengono detti **protocolli di routing** (come RIP, OSPF e BGP), che operano con criteri differenti per la scelta dei percorsi. Non è vero che tutti i router parlano con tutti gli altri router nello stesso linguaggio e non è vero che l'interazione avviene tra un router e tutti gli altri. Esiste un'organizzazione della rete per cui **la rete è una rete di reti**: è costruita interconnettendo tante reti, dove all'interno di ogni singola rete le scelte di instradamento sono fatte attraverso alcuni criteri. Perchè tutto funzioni, c'è bisogno di una connessione tra le reti. Sempre a livello di protocolli, collochiamo un altro protocollo **ICMP** che ha funzione di controllo e supporto.

Protocollo IP.

Il protocollo IP ha come obiettivo quello di consegnare i pacchetti dall'host mittente all'host destinazione. Tra le sue funzioni c'è qualla di gestire l'indirizzamento, le problematiche di gestione dei pacchetti, ad esempio dovute all'eccessiva grandezza dei pacchetti, la frammentazione (se la dimensione di un pacchetto eccede la massima dimensione sul link di uscita) e il riassemblaggio. Sempre in IP deve essere possibile supportare un'azione di multiplexing dei protocolli, che consente ad una stessa coppia di host di scambiarsi dei pacchetti a livello protocollo protocollo differenti (TCP, UDP, ecc...). Un pacchetto è costituito da un header e da un'area dati, indentificata col nome di datagramma (header+payload, al max di 2^{16}-1 byte). A differenza di quello che avviene in tecnologie di livello inferiore, la dimensione dei datagrammi può essere variabile, e per quanto riguarda l'header, anche la sua dimensione è variabile, anche se nella maggior parte dei casi è fissa ed è 20 byte. Questo perchè il protocollo può prevedere informazioni aggiuntive e quindi ha necessità di allungare l'header con campi eccezionali. Il datagramma può essere grande fino a 65535

byte.

Mittente e destinatario sono scritti nel pacchetto attraverso il loro indirizzo IP, che identifica sia la destinazione che la sorgente, e queste due informazioni sono quelle più importanti. Le altre informazioni servono a condizionare il modo con il quale i router trattano i pacchetti. Quando i router ricevono un datagramma in ingresso esaminano l'intestazione e decidono presso quale link di uscita il pacchetto va ritrasmesso. L'ultimo passo è quando arriva al destinatario, dove l'entità IP che si trova nel destinatario controlla che il destinatario del pacchetto è proprio un suo indirizzo, e capisce che non deve essere ritrasmesso ma consegnato al livello di trasporto superiore per consegnarlo all'applicazione.

Formato del pacchetto IP.

La parte di intestazione di un pacchetto è tale che su ogni riga ci siano 32bit, cioè 4 byte. La parte di opzioni spesso non è presente, generalmente ci sono 5 righe (in tutto 20byte).
L'header è un record di campi a dimensione variabile. Il primo campo serve a identificare la **versione** del protocollo IP (0100 per la quarta). Il campo **IHL** è il campo di header length, e mi dice quanto è grande l'header. C'è il campo **Type Of Service**, 8 bit, che contiene un'informazione che può condizionare la priorità di trattamento del pacchetto nel router. Il campo **Total Length** codifica la lunghezza totale del pacchetto. Questo campo fa capire perchè il payload deve avere massimo quella dimensione. I campi della seconda riga servono per la frammentazione. Il campo **TTL** (Time-To-Live) serve a evitare che un pacchetto che sia entrato in un pacchetto circolare ci rimanga all'infinito. Ogni router che si presta a ritrasmettere un pacchetto, decrementa il valore del TTL. Quando il TTL arriva a 0, il pacchetto viene cancellato, e quindi esiste un numero massimo di hop che il pacchetto può attraversare nella rete determinato dal TTL iniziale. Il campo **Protocol** serve a identificare il livello trasporto contenuto nel payload, e l'entità IP che riceve il pacchetto deve fare l'azione di demultiplexing, per capire se il pacchetto deve essere consegnato a TCP, UDP o altre entità. Il TCP è codificato con un opportuno numero di protocollo in quel campo. Ogni protocollo a livello superiore è identificato dal numero di protocollo standard. Il campo **Header Checksum** serve al controllo: ogni router che riceve il pacchetto deve verificare se l'intestazione del pacchetto non ha subito alterazioni, perchè se questo dovesse avvenire il pacchetto è inutile trasferirlo nella rete.
In questo campo c'è un'informazione di controllo sulla base dell'informazione memorizzata nell'intestazione. Siccome ogni router modifica l'intestazione del pacchetto perchè decrementa il TTL, deve ricalcolare la checksum. Poi ci stanno le informazioni di **indirizzo IP sorgente** e **indirizzo IP destinatario**, anche se solo ils econdo è fondamentale, in quanto il mittente serve al destinatario ma non ai router di trasmissione.

Frammentazione e riassemblaggio IP. (slide 18)

Se un pacchetto arriva a un router che deve trasmetterlo su un link di uscita che ha massima dimensione del pacchetto minore di quella del pacchetto, questo deve essere frammentato. Questi pacchetti proseguiranno il percorso verso la destinazione, e chi si occupa di riassemblare i pacchetti è l'host di destinazione. Quando arriva un frammento si tiene da parte finché il pacchetto non è completo. Poi c'è un timeout per aspettare i frammenti: finché non arriva l'ultimo pezzo il destinatario non sa quanti sono i frammenti. A valle di quest'azione di ricostruzione cosegna il pacchetto al protocollo di livello superiore.
Il riassemblaggio non avviene nei router intermedi. Il destinatario dovrà trovare nell'intestazione dei singoli frammenti come poterli riassemblare, e lo fa conoscendo l'ordine dei frammenti. Siccome i pacchetti viaggiano indipendentemente, può anche accadere che un pacchetto vada perso. In questo caso, l'entità destinataria aspetterà un po' di tempo, dopodichè, se questo pacchetto non è stato

ricostruito completamente, i vari frammenti vengono buttati via. Le informazioni per l'assemblaggio del pacchetto sono contenute nell'header. Sulla base del valore di **offset**, il destinatario ricolloca i vari frammenti per ricostruire il pacchetto.
Quando arrivano i frammenti c'è un campo **ID identification**, campo di 16 bit che serve a capire a quale pacchetto appartengono. Poi c'è un **fragment offset**: numero di 13 bit che identifica la posizione del frammento all'interno del pacchetto in multipli di 8 bit. Quindi il primo frammento avrà offset 0, il secondo frammento avrà un valore a che moltiplicato per 8 mi serve a dire di quanto è spostato il frammento nel pacchetto. Il bit **M** (***more fragments***) è a 1 se il frammento non è l'ultimo, a 0 se è l'ultimo. Non è scritto il numero totale di frammenti perché potrebbe esserci un ulteriore frammentazione dei pacchetti in caso di link che accetta una dimensione ancora minore. Il flag **D** (***don't fragment***) si dice a un router di non frammentare un pacchetto.
Nell'*esempio*, il pacchetto deve essere frammentato in 3 pacchetti più piccoli. Il pacchetto nasce di 4000 Byte, però ha un header IP e un payload. L'header occupa 20 Byte. Questo pacchetto genera 3 frammenti, dove ognuno sarà un pacchetto con un header proprio. Ogni singolo frammento può essere al massimo di 1500, e se il primo frammento è di 1500, il suo payload è di 1480. Per il secondo frammento vale lo stesso. Il payload 3 avrà dimensione 1020. Negli header di ogni frammento ci sono le informazioni sul riassemblaggio del pacchetto.

Opzioni.

Le opzioni nell'intestazione servono per avere informazioni ulteriori. Nelle reti IP l'unico modo affidabile per sapere un pacchetto quale sequenza di routing attraversa è usare il **route recording** nel campo opzioni. In questa maniera, quando il pacchetto arriva al destinatario, capisce questo pacchetto da quale router è stato elaborato ed inoltrato. Questo è l'unico modo affidabile al 100%. L'informazione che si ricava in altre maniere non è corretta al 100%, nel senso che, siccome le scelte di instradamento possono variare nel tempo, il fatto che un pacchetto ha fatto un certo percorso non mi autorizza che un altro pacchetto seguirà o stesso percorso. A causa delle opzioni, l'header può essere di lunghezza variabile, ed è per questo che c'è il campo header length.
Fino a quando i pacchetti non hanno opzioni, la loro elaborazione avviene mediante circuiti dedicati in hardware. Si utilizzano circuiti dedicati per velocizzare le operazioni allo scopo di massimizzare la quantità di pacchetti che il router è in grado di elaborare. Il fatto che possa essere richiesto di fare azioni ulteriori, e queste azioni sono a volte alcune e a volte altre a seconda del campo opzioni, rende più difficile le operazioni, e ciò rende difficile lavorare in hardware. Quindi se un pacchetto non ha opzioni viene elaborato in hardware (**fast path**); altrimenti esegue un altro percorso (**slow path**) che fa il pacchetto dentro al router, nel quale il pacchetto viene elaborato dalla CPU del router, quindi via software. Questo fa l'analisi dei campi opzionali ed elabora le azioni di opzione. La capacità di troughput di router sullo slow path è minore di quella che ha sul fast path: quindi non si guarda in modo benevolo ai campi opzionali nell'header. Per questo certe opzioni sono sconsigliate e spesso neanche utilizzate. Il source recording è indicato nel protocollo ma spesso non viene fatto perché questo lavoro rallenterebbe i router.
Il modello di servizio è best-effort e qualunque forma di affidabilità che voglio ottenere devo implementarli nei protocolli di livello superiore. Non mi devo aspettare dai router nessun'altro supporto che non sia la consegna nel miglior modo possibile la consegna.

Indirizzi IP.

Hanno dimensione 32 bit. Un indirizzo IP identifica univocamente un'interfaccia di dispositivo, ed è un'informazione strutturata in almeno due campi:

- **Rete**: Serve a collocare l'indirizzo all'interno della rete.
- **Host**: All'interno della rete specifica l'interfaccia del dispositivo.

Si parla di interfacce perché ci può essere un router con più interfacce, e queste hanno diverso IP.

Quindi l'IP è diviso in una parte rete e una host. La linea di demarcazione tra i due campi è variabile perchè se la mettessi fissa avrei un'eccessiva rigidità. L'assunzione fondamentale è che tutti gli host che si trovano all'interno della stessa rete devono poter comunicare direttamente attraverso una tecnologia di livello datalink. Ci sono dispositivi che hanno lo stesso indirizzo rete che devono poter comunicare a livello locale senza l'utilizzo del router. Il router entra in gioco solo quando un dispositivo vuole parlare con una rete non locale.
Gli indirizzi IP si rappresentano con stringhe di 32 bit, ma in generale con una notazione dot decimal. Sono quattro numeri decimali separati da un punto. Il numero deve essere compreso tra 0 e 255. L'**ICANN** assegna gli indirizzi IP (fa anche altro, gestisce i domini DNS, risolve dispute eccetera) a delle reti. Tutti gli host all'interno di una stessa rete hanno uno stesso valore. Bisogna dire dove si pone la divisione tra rete e host: quanti costituiscono la rete e quanti gli host.
Si mette un indirizzo: x.y.z.w/24 vuol dire che i primi 24 bit costituiscono la parte rete e i restati 8 bit rappresentano la parte host. Ogni volta che due host parlano tra loro in una stessa rete il router non entra in gioco. Anche i collegamenti punto-punto tra router costituiscono una rete ma bisogna stare attenti a non far confusione di assegnazione indirizzi.

Reti di Calcolatori - 18/10/2012

Lezione 10. Classi di indirizzi e subnetting

Tutti gli host che si trovano sulla stessa rete possono comunicare direttamente senza l'ausilio di un router, cioè attraverso tecnologie di livello data-link. Sono legati secondo una logica a bus. Anche il router, da questo punto di vista, è un host, perchè con una delle sue interfacce appartiene alla rete dove si collegano tutti gli atlri host. Il problema è capire quanti bit dell'indirizzo costituiscono il prefisso di rete e consequenzialmente il complemento a 32 che rappresenta l'indirizzo di host. Se stabilissi la struttura in maniera rigida, avrei una rigidità nella gestione, in quanto gestirei un numero elevato di reti, ma ogni singola rete potrebbe contenere solo pochi host (ache perché non tutti gli indirizzi sono assegnabili ad interfacce di host). Nel corso del tempo si è passato da una gestione statica ad una gestione più facilmente adattabile alle esigenze di ogni singola organizzazione.

Gestione per classi.

Una prima versione suddivide lo spazio di indirizzamento in classi. Quello che deve essere univoco è la combinazione indirizzo di rete - indirizzo di host. All'interno di una stessa rete non possono esserci due host con indirizzo di host uguale, tra più reti sì.

Si divide lo spazio di indirizzamento in intervalli ben definiti detti **classi**. E' stato partizionato l'intero spazio di indirizzi in 5 intervalli:

- **Classe A**: rientrano nella prima metà, e visti in binario hanno la caratteristica di avere il primo bit a 0 (sono del tipo 0.x.x.x). Hanno 8 bit per la rete e 24 per la parte host (realmente liberi sono 8-1).
- **Classe B**: un quarto dello spazio di indirizzamento, è costutuita da indirizzi che in binario cominciano per 10. Hanno 16 bit (realmente liberi 16-2) per la rete e 16 per l'host.
- **Classe C**: è un ottavo dello spazio di indirizzamento ed è costituita da da indirizzi che in binario cominciano per 110. Hanno 24 bit per il campo rete e solo 8 bit per l'host.
- **Classe D**: è un sedicesimo della classe di indirizzament ed è costutuita da indirizzi che in binario cominciano per 1110
- **Classe E**: costutuita da indirizzi che in binario cominciano per 1111

Queste classi sono di non uguale ampiezza come numero di indirizzi compresi: la parte più grande è rappresentata dagli indirizzi di classe A. Le classi A,B e C sono utilizzate per identificare univocamente host o router all'interno di internet. La classe D identifica gruppi di host che ricevono pacchetti in multicast, quando ci sono gruppi di host che vogliono ricevere determinati collegamenti. La classe A non è mai stata utilizzata completamente.
Gli indirizzi vengono gestiti a lotti: quando un'entità chiede di avere assegnati degli indirizzi IP, l'entità che li gestisce assegnava un'intera classe a seconda delle esigenze di quell'organizzazione.

Le reti di classe A sono 2^7=128
Le reti di classe B sono 2^{14}=16k
Le reti di classe C sono 2^{21}=2M

Per effetto di questa tecnica di gestione c'è stato un utilizzo con un'efficienza estremamente bassa, perchè c'è stato un rapido esaurimento dello spazio degli indirizzi. Questa tecnica di gestione è stata

abbandonata.

- Classe A: [0.0.0.0 → 127.255.255.255]
- Classe B: [128.0.0.0 → 191.255.255.255]
- Classe C: [192.0.0.0 → 223.255.255.255]
- Classe D: [224.0.0.0 → 239.255.255.255]
- Classe E: [240.0.0.0 → 255.255.255.255]

Un router deve avere una entry per ogni indirizzo IP di destinazione? No, perchè a un router basta capire se la destinazione è direttamente collegata a una delle sue interfacce o se la destinazione non è situata nella rete locale. In realtà, tutti i router hanno delle entry di default, cioè se la destinazione non è nessuna delle reti nel pacchetto, allora instrada il pacchetto da questa parte. La cosa è diversa nei router di backbone perché i pacchetti possono andare verso una qualunque rete.

Indirizzi IP speciali.

Hanno una funzione di identificazione particolare e, quindi, non possono essere assegnati a nessun host.
La rete ha un suffisso con tutti zero, che quindi non può essere assegnato ad un host.
Se abbiamo una rete 142.224.0.0, questa rete è identificata collettivamente da un indirizzo IP che ha nella parte host tutti 0. Poi c'è l'indirizzo 0.0.0.0 che, a livello locale, identifica tutti i possibili indirizzi di destinazione. Molte tecnologie di rete locale, consentono una trasmissione di pacchetti in **broadcast**: è indirizzata a tutti gli host che si trovano all'interno di una rete. Quando un host vuole inviare un pacchetto a tutti gli host della rete usa un indirizzo che nel campo host tutti 1, ad esempio 124.224.255.255.
A livello locale si può usare anche 255.255.255.255, che è utilizzato per un broadcast solo per la rete locale, dove l'altro può a volte varcare i router.

L'indirizzo 0.0.0.0 è il **this computer address**, perchè è l'indirizzo assegnato prima che sia assegnato dalla rete un indirizzo IP. L'indirizzo IP è assegnato o in maniera statica (l'IP è scritto in un file del sistema operativo e viene **assegnato staticamente** all’accensione della macchina, scritto nelle configurazioni) **o** in **maniera dinamica**, attraverso la quale c'è un'entità esterna alla macchina che, su richiesta, fornisce un indirizzo IP alla macchina, mediante opportuni protocolli. Questa assegnazione è opportuna quando gli indirizzi disponibili sono limitati e sono in numero inferiore rispetto alle macchine che si trovano sulla rete.Quindi gli indirizzi possono essere assegnati dinamicamente, e quindi tutte le macchine avranno un indirizzo. Le macchine che offrono servizi tipicamente usano una strategia di assegnazione statica per quelle macchine stesse, perché devono essere sempre disponibili ad un indirizzo noto.

L'indirizzo 127.0.0.1 è utilizzato per identificare come **destinazione il computer stesso**. Tutta la classe 127.x.x.x è speciale, ha come destinazione il computer stesso. Ogni macchina ha sempre tra le sue interfacce disponibili un'interfaccia virtuale (eth0) che ha assegnato l'indirizzo 127.0.0.1, e non è associata a nessuna interfaccia di rete fisica, però esiste ed è attiva. Quando si usa un indirizzo 127.x.x.x come destinazione, nessun pacchetto esce definitivamente dal router, ma tutte le informazioni restano all'interno della rete locale. Il client può raggiungere il server con 127.0.0.1 e si può fare testing anche senza essere collegati alla rete. Si copia il pacchetto da coda in uscita a coda di entrata, i pacchetti non escono dalla rete, e non sono sniffabili. Questa strategia non è efficiente, perchè ha comportato lo sperco di indirizzi. La scarsa disponibilità di indirizzi IP ha limitato l'estensione di internet. Sono state attuate due strategie: gli **indirizzi privati** e le **sottoreti**.

Strategie a indirizzi privati e a sottoreti.

La strategia a sottorete ha reso possibile la non assegnazione di indirizzi a blocchi. Si viola l'assunzione che ogni host è univocamente identificato attraverso un unico IP. L'idea è che avere un unico IP è necessario solo per le macchine server, quelle ben note e sempre raggiungibili. I router di frontiera (router edge delle reti) effettuano una traduzione e uno smistamento. Questo non va bene se nella rete locale ho un server.
Gli indirizzi 192.168.x.x sono gli indirizzi di una particolare classe C riservati per usi privati: questi indirizzi sono definiti privati perchè ogni gestore di rete li può liberamente assegnare agli host della propria rete senza preoccuparsi dell'univocità. Questi indirizzi vanno bene fino a quando le comunicazioni avvengono all'interno della stessa rete locale, ma non vanno più bene quando si vuole far uscire un pacchetto verso internet. Questo pacchetto non avrebbe più un mittente identificato univocamente nella rete, e allora questa tecnica va usata solo ed esclusivamente in reti private. Quando la rete si vuole collegare a internet occorre un meccanismo di traduzione degli indirizzi che opera il router, di tipo NAT, per cui i pacchetti, una volta che escono, hanno l'indirizzo pubblico del router.

Le 3 tipologie A,B,C, nella divisione in classi statica, si adattano alle esigenze di reti grandi, medie e piccole. Il vantaggio di questa tecnica è la sua semplicità. Questo comporta degli sprechi, perchè le classi A e le classi C possono essere numerose.

Subnetting.

Ciascuna coppia di interfacce collegate direttamente rappresenta una rete dal punto di vista IP. Altra caratteristica di questo collegamento è che le interfacce collegate sono solo 2, quindi ogni pacchetto inviato da una parte ha una sola destinazione possibile. IP gestisce collegamenti e indirizzamenti, dove ciascuno di questi collegamenti deve essere legato ad una rete. La rete più piccola che posso utilizzare è una rete di classe C. Per un uso più efficiente è stato utilizzata la tecnica del **subneting**: suddivisione delle reti in sottoreti. È una tecnica che mi consente di partizionare una classe di indirizzi in una molteplicità di sottoreti. Ad esempio, una rete di classe A che ha 24 bit per gli host può essere partizionata in 64 sottoreti, prendendo 8 bit dall'host per la sottorete. Adottando questa strategia ho partizionato una rete di classe A in 256 diverse sottoreti, dove ogni sottorete è identificata dal valore della stringa di 8 bit. Nella sua interezza, la rete è identificata dalla coppia rete+sottorete. Tutti gli host all'interno della stessa sottorete deve avere una stretta comunicazione. All'esterno la rete è vista come un unico indirizzo ma nello smistamento dei pacchetti l'organizzazione fa uso dello smistamento. Il vincolo che IP pone agli host che si trovano nella stessa rete viene tramutato in vincolo di raggiungibilità diretta tra gli host che appartengono alla stessa sottorete.
L'organizzazione ha una molteplicità di host da collegare alla rete per cui ha avuto assegnata una classe A. Però tutti questi host non ce li ha tutti nello stesso posto fisicamente, allora può partizzionare la sua rete in 4 parti e per il resto del mondo la rete è identificata come un unico indirizzo di classe A, ma quando i pacchetti arrivano, sfruttando l'indirizzo di sottorete i router capiscono un pacchetto che host ha come destinazione. Quando si fa questo, occorre che un router che fa le sue scelte di instradamento non può più guardare la parte rete, ma deve guardare l'insieme di rete e sottorete. E quanti bit deve guardare? Visto che l'ampiezza del numero di sottorete è variabile (da chi amministra la rete), il numero di bit che servono per identificare rete e sottorete non può essere più derivata dallo stesso indirizzo IP, in quanto il campo sottorete può variare in base a scelte effettuate per la divisione, ma occorre un'informazione ausiliaria, che deve essere mantenuta dai router.
Questa informazione è fornita ai router secondo una maschera di bit chiamata **Subnet Mask** (o

Netmask), che è una maschera di 32 bit che definisce, all'interno dei 32 bit dell'indirizzo quali sono quelli che appartengono all'insieme rete, sottorete e campo host. Nella netmask ci saranno tanti 1 per i bit che appartengono al campo rete e ci saranno tanti 0 per i bit che appartengono al campo host.Quando un router riceve un pacchetto con una certa destinazione, deve capire qual'è la sottorete dell'host a cui il pacchetto è indirizzato. Di fatto, viene fatta un'operazione di **AND** bit a bit tra indirizzo IP e corrispondente netmask. Si ottiene lo stesso valore dell'IP dove la netmask ha gli 1 e si ottiene il valore nullo (0) nei bit 0 della netmask.
Usando la Netmask, l'amministratore di rete ha dei gradi di libertà e una flessibilità nel dimensionare l'indirizzo IP.

Bridge.

Il bridge, a differenza del ruouter, è un oggetto che, nello stack di protocolli, si ferma a livello 2. Non partecipa, quindi, all'instradamento di livello IP. La netmask spesso si identifica con **/n**, dove n rappresenta il numero di bit di rete e sottorete, cioè numero degli 1 della netmask. Altrimenti si rappresenta in decimale con i punti. Il campo bit quindi non è necessariamente di 8bit, si usano spesso per individuare frontiera ed estremi dell'intervallo. Sono scelte di progetto.
La Netmask è un parametro che viene configurato per ogni interfaccia di rete per gli host che appartengono alla stessa rete.
La netmask più piccola è la 11[...]111100, ha 4 indirizzi e viene usata per i collegamenti punto-punto: 255.255.255.252 (cioè 6 bit a 1 e 2 a 0). Questa netmask lascia solo 2 bit nel campo host, cioè 4 configurazioni. 0 è per la sottorete, 1 è per il broadcast e le altre due per le due interfacce all'estremità del collegamento.

Reti di Calcolatori - 19/10/2012

Lezione 11. Trasmissione di un pacchetto sulla rete locale: ARP - RARP - DHCP

Ci sono 2 host A e B che vogliono scambiarsi i pacchetti e sono collocati in due reti locali differenti collegate attraverso dei router. La rete A dispone di un router R1 che è collegato, mediante un collegamento CDN (collegamento diretto numerico), al Router R2 della rete B. A livello 3, quando A vuole inviare un pacchetto a B si accorge che B non appartiene alla stessa subnet.

(A.IP && A.NETMASK) è la AND tra due stringhe di bit. Mi restituisce l'indirizzo della sottorete in cui è collocato A. Il test che fa è vedere se questa quantità è uguale a quello che si ottiene facendo (B.IP && A.NETMASK). Allora, se queste due stringhe di bit sono uguali, A deduce che B si trova sulla sua stessa sottorete, e allora stiamo in una situazione in cui A e B possono dialogare direttamente. Se la destinazione del pacchetto sta in una subnet differente, deduce che non può inviare il pacchetto direttamente ma lo deve inviare attraverso un router. La tabella di routing di A dirà che tutti gli host che non si trovano nella sottorete A possono essere raggiunti tramite il router. Nella ***slide 14***, la situazione è diversa: A non può inviare il pacchetto a B direttamente ma tramite un router. Tutti gli host che non si trovano nella stessa sottorete di A si trovano nella tabella R1.

Quando un pacchetto scende dal livello rete al livello data link, viene imbustato. Il pacchetto IP contiene un header e un payload. Nell'header le due informazioni significative sono gli indirizzi: mittente e destinazione. Questi due indirizzi non vengono mai alterati dai router nel percorso. A livello datalink un indirizzo non serve per prendere decisioni di instradamento, la connessione tra gli host è diretta. Il pacchetto a livello datalink ha un indirizzo che serve a farlo recepire dall'host verso il quale il pacchetto è destinato: gli altri host possono vederlo ma lo ignorano perché non c'è un indirizzo della loro interfaccia. Anche a livello datalink sono importanti gli indirizzi: c'è quello che identifica l'interfaccia mittente e quello per l'interfaccia del destinatario, in questo caso l'intefaccia del router.

L'**indirizzo data link** è un indirizzo hardware perchè viene cablato nell'hardware della scheda rete. Sono formati da 48 bit. Le interfacce sono assegnate dal costruttore, e sono associati alla scheda e quindi dipendenti dalla posizione della macchina all'interno della rete, e il costruttore si occupa di non fare mai due indirizzi macchina uguali (**MAC address**). C'è una parte dell'indirizzo che identifica il costruttore e un'altra parte che identifica la specifica scheda. Ogni costruttore ha assegnato dei lotti di indirizzo e a mano a mano inserisce numeri progressivi. All'interno della stessa rete locale possiamo avere più schede, e quando l'host A invia il pacchetto al router R1, xkè questo pacchetto venga recepito da R1, […].
Lo scopo del livello data-link è quello di far fare al pacchetto un salto verso il router successivo. Se nella rete di sinistra ci sono 2 router (***slide 3***), l'instradamento del pacchetto dipende dalle tabelle di instradamento dell'host A.
Il router R2 vede l'indirizzo IP destinazione e fa il controllo (R2.IP' && R2.NETMASK) == (B.IP && R2.NETMASK), [o (R2'.IP&&R2'.NETMASK)==(B.IP&&R2'.NETMASK) ??] e se questo controllo è positivo, allora l'host destinatario è collegato.
Come fa A a conoscere il MAC address di R1? Quest'informazione non può essere nota a priori. Occorre risolvere il problema di risoluzione dell'indirizzo, perchè A non conosce l'indirizzo fisico di R1, ma conosce soltanto il suo indirizzo IP. Questo meccanismo di risoluzione è offerto dal protocollo **ARP**.

Protocollo ARP (Address Resolution Protocol).

A conosce l'indirizzo IP di B ma non conosce il suo indirizzo fisico. ARP si basa sull'assunzione che la rete su cui sono collegati A e B sia una rete che supporti la trasmissione broadcast: se uno invia una frame con un determinato indirizzo datalink, quel frame viene ascoltato da tutti gli host della rete. Allora, A manda in broadcast un pacchetto che contiene l'indirizzo di rete di B, allo scopo di ricevere l'indirizzo fisico di B (**ARP request**). Questo pacchetto arriva in broadcast, ed è letto dall'host B, che risponde ad A inviando un messaggio di **ARP reply** nel quale, affianco all'indirizzo IP richiesto c'è il corrispondente indirizzo fisico MAC. Nel momento che ho due host, un attimo prima che veda viaggiare un pacchetto da A a B mi aspetto una ARP request e la corrispondente ARP reply. Con il caching, una volta che A ha appreso l'indirizzo di B, se lo memorizza e mantiene l'associazione indirizzo fisico - indirizzo logico in una memoria associativa locale, che è mantenuta dal sistema operativo, in maniera tale da non fare una request per ogni pacchetti.

Formato del pacchetto ARP.

Il protocollo ARP è un protocollo di livello 3, alla pari di IP, anche se non ha la funzione di instradamento. Ciò significa che i pacchetti ARP sono direttamente incapsulati nella busta Ethernet, così come è incapsulato il pacchetto IP. Per discriminare i due pacchetti bisogna andare a vedere il campo protocol type che sta nella busta ethernet. Il messaggio che A manda a B per conoscere l'indirizzo MAC di B avrà:

- Il **proprio indirizzo hardware**, che occupa una riga e mezza (xkè gli indirizzi sono lunghi 48 bit).
- Su 32 bit c'è l'**indirizzo IP di A**.
- Nell'ARP request, **l'indirizzo hardware** dell'host **B** non è noto, quindi ci sono tutti 0 perchè l'indirizzo target non è noto.
- L'**indirizzo IP di B**.
- **Operation** è un campo che serve a identificare il tipo di messaggio.
- **HLEN** e **PLEN**: Lunghezza degli indirizzi fisici e logici (potrebbe essere più generale, usato per fare risoluzione su protocolli differenti).

Il messaggio ARP è il broadcast a livello datalink. L'header del frame di livello 2 serve a far capire che il pacchetto è ARP e non è IP o qualunque altro protocollo di livello 3. Per capire se la destinazione sta nella propria sottorete si fa una AND bit a bit con la netmask. In questo caso, se sta nella stessa sottorete, sa che è direttamente raggiungibile e manda una ARP request in broadcast.
Il primo scenario ARP da considerare è quando A e B si trovano sulla stessa rete. Se non stanno sulla stessa LAN, la cosa diventa articolata, perchè si mettono in mezzo i router. Infatti, consideriamo un caso in cui vogliamo mandare un pacchetto da A ad X, con X che si trova in un'altra sottorete (internetwork di reti private). Se il controllo con le netmask fallisce A capisce che deve inviare le request al router R1. A fa un ARP request per conoscere il MAC di R1 (A conosce l'IP di R1 perché è scritto nella sua tabella di routing). Si incapsula il pacchetto a livello datalink al router R1, trova l'IP di R1 dalla tabella di routing, fa richiesta per il MAC address del router per consegnargli il pacchetto.

Proxy ARP. (*slide 30*)

Si istruisce il router R1 (slide 30) di rispondere a tutte le ARP request a livello superiore. Allora R1 risponde con le ARP reply a nome degli host a livello inferiore collegati alla sua interfaccia. Non da il vero indirizzo MAC di H3 (perché se H1 volesse mandare un file direttamente ad H3 non arriverebbe) ma da la propria interfaccia. H1 manda il pacchetto a R1 avendo come destinazione H4 e come indirizzo quello dell'interfaccia del router.

Protocollo RARP (Reverse Address Resolution Protocol).

L'indirizzo IP ad una macchina può sempre essere assegnato staticamente, e nel fare questo dobbiamo sapere che dobbiamo rispettare alcune regole:

- deve essere nella stessa sottorete dove vivono gli host collegati alla rete locale,
- deve essere univoco, cioè non deve essere stato scelto già da un'altra macchina.

La configurazione statica si può fare in reti piccole, quando c'è un unico amministratore di rete, ma nelle reti aziendali quest'approccio non va bene, perchè per ogni computer nuovo ci vuole una configurazione. Ci sono situazioni in cui la configurazione statica non è possibile. L'assegnazione IP viene fatta dal sistema operativo al momento del caricamento. Se la macchina è diskless non ha modo di configurare staticamente l'IP. Viene usato quindi il protocollo RARP (usata in casi diskless), in cui la macchina non conosce l'indirizzo IP all'accensione e allora invia una richiesta di assegnazione dell'indirizzo al server in broadcast, e se sulla rete c'è un server RARP attivo che sente questa richiesta e risponde assegnando alla macchina un indirizzo IP. Questo protocollo è stato sostituito dal DHCP.

Protocollo DHCP (Dynamic Host Configuration Protocol)

Utile principalmente quando ci sono macchine che si collegano in maniera saltuaria. L'indirizzo viene assegnato da server DHCP, distinto dal router. Quando un server DHCP non è attivo non si possono connettere nuove macchine alla rete. Il messaggio di richiesta da parte di un client si fa sempre in broadcast. Il primo messaggio non è rivolto direttamente all'assegnazione di un indirizzo ma alla scoperta di server DHCP (magari per preferire un server al posto di un altro): questa è la richiesta **DHCP discovery**. Il protocollo **funziona su UDP**. La destinazione è inviata all'IP broadcast locale. I messaggi hanno un numero progressivo di transazione per associare richieste a risposte. Il server risponde con un messaggio di offerta di indirizzo (**DHCP offer**). Anche questo messaggio è in broadcast perché il client non ha ancora acquisito l'indirizzo IP. Nel messaggio c'è un identificativo del server (il suo IP), l'indirizzo IP offerto (your IP address) che può essere assegnato alla macchina e c'è lo stesso transaction ID del client. In questo modo, se ci sono due client che hanno fatto richiesta nello stesso momento si hanno due risposte diverse che non si confondono grazie al transaction ID. L'offerta ha una validità (tot secondi). Se si accetta l'offerta si risponde (sempre usando come mittente tutti zeri perché non ha ancora acquisito l'IP) e la destinazione è ancora broadcast. Parte quindi la richiesta dell'indirizzo offerto. Il server quindi assegna l'IP.

Quest'assegnazione avrà la validità dei secondi assegnati dopo i quali i client deve rinnovare la richiesta (per cui otterrà lo stesso IP, e se non la rinnova per tempo ne avrà uno differente).
Il DHCP è una specie di protocollo applicativo, si colloca sopra il protocollo di trasporto, utile perché tramite quelle porte posso discriminare che sono messaggi generati dal protocollo DHCP.

Reti di Calcolatori - 23/10/2012

Protocollo ICMP (Internet Control Message Protocol)

E' un protocollo di servizio progettato per svolgere diverse funzionalità, che consentono di capire lo stato della rete, (es.: se c'è connettività tra due host, ecc...) viene anche usato per riportare anomalie tra router e router o tra router e end system. I messaggi ICMP sono trasportati in datagrammi IP, cioè ICMP sta al livello 4: un messaggio di ICMP è sempre incapsulato in un datagramma IP, e quindi si colloca allo stesso livello di TCP e UDP dal punto di vista di protocollo, ma non dal punto di vista della funzionalità. Un messaggio ICMP è sempre incapsulato in un datagramma IP invece di essere immesso in pacchetti ethernet come l'ARP.
Esistono varie tipologie di messaggi, discriminati dal valore di 2 campi: **Type** e **Code**. La combinazione di questi due valori determina il tipo di messaggio.
I messaggi che vediamo più frequentemente sono gli **echo reply** e **echo request**, chiamati ping perchè sono generati dall'applicazione **ping**, mediante la quale possiamo capire se c'è connettività tra la macchina su cui stiamo operando e la macchina target, che possiamo individuare o con nome simbolico o con indirizzo IP. Questi messaggi vengono propagati dai router sulla rete fino a quando arrivano alla macchina di destinazione e tipicamente un host è configurato per far sì che quando riceve un messaggio, risponda automaticamente con un messaggio di echo reply. Chi ha inviato l'echo request vede l'echo reply e capisce che c'è connettività nella rete tra quella macchina e quella remota potendo calcolare anche il tempo dei pacchetti per fare un round trip (tempo di andata e ritorno dei pacchetti).

Ci sono altri messaggi:

- **destination port unreachable**: generato da un host quando un altro host ha provato a contattarlo su un numero di porta che su quell'host non è utilizzato. Se proviamo a contattare una macchina sulla quale non c'è in esecuzione una macchina web server, non si può stabilire una connessione TCP da entrambe le parti perchè la porta della macchina server non è abilitata, e quindi viene generato un messaggio di destination port unreachable. Questo è ancora più utile nel caso di protocollo UDP, perché non c'è un timeout che scade.
- **Messaggi rimandati da un host mittente da router intermedi**. Può accadere che vogliamo contattare una macchina ad un certo indirizzo IP. L'inoltro del pacchetto sulla rete avviene andando a guardare la rete di destinazione, quindi i router intermedi faranno sempre le stesse scelte di instradamento. La differenza si fa soltanto nell'ultimo router. Quando una determinata macchina non esiste nella rete, il pacchetto viaggia sulla rete, ma poi l'ultimo router non riesce a consegnare quel pacchetto perchè nessuno risponde all'ARP request e allora viene generato il **dest host unreachable**, che riporta al mittente che il pacchetto ha viaggiato sulla rete ma non è stato consegnato. Il pacchetto si può fermare prima, se una sottorete non è stata collegata, e allora il pacchetto fa una serie di percorsi e poi quando arriva ad un router quella rete non è raggiungibile. Viene generato un **dest network unreachable**. Unreachable è quando esiste nelle tabelle di routing dei router ma il pacchetto non può essere consegnato. Se non esiste la regola di routing, il router non sa proprio come raggiungerla e dà unknown.
- **TTL exceeded**: viene generato da un router quando deve inoltrare un pacchetto il cui campo TTL vale 1. Se il router si accorge che questo pacchetto deve essere ulteriormente consegnato, non lo inoltra ulteriormente, scarta il pacchetto, e chi ha inviato il pacchetto viene notificato con un messaggio di TTL exceeded. Se il mittente vuole che il pacchetto faccia un viaggio più lungo sulla rete serve un TTL più grande.

L'applicazione ping serve a verificare la connettività tra due macchine, vedendo se i pacchetti

riescono ad arrivare all'altra macchina. Per capire dove nasce il problema, c'è di ausilio il comando di **Traceroute**, che consente di capire la sequenza di router attraversati dai pacchetti fino ad arrivare a destinazione. Potremmo capire fino a che punto arriva il pacchetto nella rete e poi non va più avanti. La possibilità dei percorsi alternativi si può verificare solo quando esistono, perchè ci potrebbero essere anche dei percorsi obbligati. Il comando traceroute dà la sequenza di router attraversati. Viene inviato un pacchetto da A a B e il datagramma con TTL 1 arriva al router 1 e quindi viene fermato e viene inviato indietro un messaggio ICMP del tipo TTL expired. Allora, il client ha l'informazione che il primo router che vede il pacchetto è il router 1, e vede l'interfaccia collegata ad A. Capisce che se vuole raggiungere B deve aumentare TTL. Invia un pacchetto con TTL 2, che da R1 viene inoltrato a R2, e R2 non riesce a consegnare il pacchetto direttamente al destinatario e quindi invia un pacchetto ICMP con destinazione A, ed A apprende l'interfaccia di R2 con R1. Lo stesso avviene quando A invia il pacchetto con TTL 3. Inviando un pacchetto con TTL 4, il pacchetto riesce ad arrivare alla destinazione, e tipicamente viene consegnato all'entità IP che lo riceve e non genera indietro nessun messaggio.
Per capire che sono arrivato a B occorre che l'arrivo del pacchetto al destinatario deve generare un evento notificato, e si possono usare 2 tecniche alternative:

- Il pacchetto che A invia incrementando il TTL è un pacchetto IP con protocollo di trasporto UDP, che ha come destinazione un destination port (x) un numero di porta introvabile, dove non c'è nessuna applicazione in ascolto. Se questo è il caso, quando il pacchetto arriva al destinatario, il destinatario non riesce a consegnare il pacchetto e viene generato un messaggio di destination port unreachable, e A capisce che il pacchetto è arrivato a destinazione.
- Il messaggio che viene inviato da A deve essere un messaggio di echo request che, quando arriva a destinazione, genera un corrispondente messaggio di echo reply.

Problema: può accadere che il routing sia asimmetrico, cioè il percorso che fanno i pacchetti da A a B può essere diverso dal percorso che i pacchetti fanno da B ad A. Ma anche quando il router è simmetrico, nel momento in cui B fa il tracerout verso A ha comunque un percorso diverso. C'è una sequenza di indirizzi diversa da quella di prima per un fatto di interfacce. Questo rende difficile la scoperta della tipologia della rete: potremmo avere delle difficoltà ad associare due indirizzi IP diversi allo stesso dispositivo, perchè potrebbero essere due interfacce diverse.
Se un router sulla catena è configurato per non inviare il TTL expired, il router non viene evidenziato nel traceroute. Se succede questo, scatta un timeout, non è arrivato nessun pacchetto, e capisce che il pacchetto è morto nella rete. Comunque A manda l'altro pacchetto e questo arriva nel 3° router. Nella sequenza potrebbe mancarmi un passo, e questa informazione mi dice che tra due router cen'è un altro di cui non sono in grado di scoprire l'indirizzo IP. Può accadere che ci sia anche più di un router che non riesco a scoprire. Un ISP può configurare i propri router per non inviare TTL expired, perchè una delle informazioni che riesco a ricavare dal traceroute è la topologia della rete, e quindi del mio ISP. Questa è un'operazione sensibile, che gli ISP tendono a nascondere. La rete è configurata come rete di reti, quindi all'interno di una rete le scelte di instradamento vengono fatte dagli ISP, poi c'è un instradamento tra reti, che segue altre strategie nella scelta dei percorsi.
Il tempo che impiega il router a generare il messaggio di TTL expired è variabile e può essere maggiore al tempo che impega per inoltrare il pacchetto. Può accadere che la sequenza dei tempi di risposta non sia monotona: il tempo che impiega il router a rispondere è variabile e magari è maggiore rispetto al tempo impiegato per inoltrare il pacchetto.
Il traceroute collega i messaggi di richiesta e risposta e quindi ci dà il tempo di attraversamento.

Esercizi.

Esercizio 1

A quante subnet differenti appartengono gli indirizzi IP riportati di seguito, se immaginati tutti associati alla netmask 255.255.248.0?

255.255.248.0=/21
[248_d=11111000_b]
[47_d=00101111_b]
[23_d = 00010111_b]
[24_d=00011000_b]

Dobbiamo vedere ogni indirizzo a quale subnet appartiene.

172.16.16.223 && 255.255.248.0 = 172.16.16.0 SI
172.16.47.47 && 255.255.248.0 = 172.16.40.0
172.16.23.43 && 255.255.248.0 = 172.16.16.0 SI
172.16.24.212 && 255.255.248.0 = 172.16.24.0
172.17.16.100 && 255.255.248.0 = 172.17.16.0
172.16.26.1 && 255.255.248.0 = 172.16.24.0

Gli IP 1 e 3 stanno sulla stessa subnet.
172.16.24.212 && 255.255.248.0 = 172.16.24.0

Esercizio 2

Dato l'indirizzo IP 192.33.28.0, nel caso in cui si applichi una maschera 255.255.255.240, sottorete e relativo indirizzo di broadcast.

194.33.28.0 ␣ Classe C
Subnet ␣ 255.255.255.240

Di fatto, degli 8 bit che sono rappresentati dall'ultima cifra decimale, ne sto prendendo 4 per indicare la sottorete e gli altri 4 li lascio per gli host.

[240_d=11110000_b]

194.33.28.0 è una classe di indirizzi cui si applica una netmask 255.255.255.240. In questa classe di indirizzi, usando il subnetting, vediamo che di fatto, degli 8 bit che sono rappresentati dall'ultima cifra decimale, sto usando 4 per usare una sottorete e altri 4 bit per gli host. Sto dividendo una rete che in origine riservava 256 indirizzi per gli host in 16 diverse sottoreti, ciascuna delle quali comprende 16 host. All'interno di una sottorete ci sono almeno due indirizzi per usi speciali, non assegnati a nessuna interfaccia (0000 indica una sottorete, 1111 indica una sottorete broadcast). Posso organizzare quindi 16 sottoreti differenti con massimo 14 host.

Quali indirizzi sono validi per indicare una sottorete e il rispettivo broadcast?

1. 96=0110|0000 → Sottorete 6,host tutti zeri. È proprio l'indirizzo della sottorete. Il broadcast è 0100|1111=96+15=111. La prima risposta non è corretta.

2. 12=0000|1100 → Già non va bene.

15=0000|1111 → Andrebbe bene come indirizzo broadcast

3. Va bene, è il primo caso ma corretto

4. 128=1000|0000. Va bene, il corrispondente broadcast è: 1000|1111=143. Non è corretta

5. Non è corretta, perché 240 è un indirizzo valido di sottorete ma non di broadcast.

6. 0=0000|0000 → Va bene per la sottorete
   255=1111|1111 → Non va bene, dovrebbe essere 0000|1111. Va bene come broadcast, ma non è il broadcast di quella sottorete, ma di un'altra sottorete.

Esercizio 3

Indicare tra i seguenti un gateway valido per l'host avente indirizzo 10.16.65.203 avente netmask 255.255.255.224
Il gateway sarebbe il router, l'indirizzo di gateway deve essere nella stessa subnet dell'host, però non deve coincidere e non deve avere l'indirzzo di broadcast.
Dobbiamo vedere qual'è la subnet di appartenenza di 10.16.65.203 facendo l'and bit a bit tra host e netmask:

10.16.65.203 && 255.255.255.224 = 10.16.65.192

Quindi, nessuna delle precedenti.

Reti di calcolatori - 25/10/2012

Lezione 13. Le socket di Berkeley

Le socket di Berkeley rappresentano un'astrazione di un canale di comunicazione. Vengono utilizzate dalle applicazioni per inviare o ricevere dati sulla rete. Costituiscono un'astrazione nel senso che come i dati vengono effettivamente inviati e ricevuti non dipende dalla tecnologia utilizzata. Sono nate negli anni '80 quando iniziava ad affermarsi il protocollo IP per dare la possibilità agli sviluppatori UNIX di usare le possibilità offerte dalla rete (protocollo TCP/IP). Sono state disponibili dall'edizione UNIX 4.1. Rappresentano un canale di comunicazione: sono un **insieme di funzioni C**, cioè una **libreria C** che il programmatore può usare per sfruttare le potenzialità della rete.
Quando gli sviluppatori svilupparono il socket aderirono al modello di **UNIX**: ogni risorsa vista da una sistema operativo è ascrivibile ad un file. Le socket aderiscono a questo modello e la socket è vista come un file che si può aprire, ci si può oprerare sopra e si può chiudere. Un file è un'astrazione di risorsa vista dal sistema operativo: si può aprire, modificare, chiudere. La socket è vista come se fosse un file. L'operazione di scrittura nella socket sarà quella di invio dei dati. E' stata una delle prime librerie ad essere usata e si diffuse su UNIX che era moto usato all'epoca: è diventata lo standard di fatto anche se non esisteva ancora l'ISO.
Il fatto che le librerie siano in C non vuol dire che non si possano fare librerie per la rete anche in altri linguaggi. Le applicazioni Java oggi sfruttano le socket. La libreria C, tra tutte quelle dello stesso tipo, è quella con la potenza espressiva più elevata. Lo stesso non si può dire per le altre librerie ad alto livello come Java, che implementa solo un sottoinsieme delle funzionalità delle socket.
La libreria delle socket è un interfaccia verso il sistema operativo, dove sono implementate le interfacce di rete. Il primo messaggio che la socket da al sistema operativo è chiedere le funzionalità di rete necessarie. Questo step corrisponde all'apertura del file: il sistema operativo crea la socket e la istituisce creando un **socket descriptor**: un puntatore che aiuta a riferire alle risorse che sono state allocate. L'applicazione che ha questo socket descriptor può leggerlo, usarlo e chiuderlo. Un'operazione di lettura e scrittura significa inviare e ricevere dati. La socket prima si richiede, poi si scrive e si legge sopra e poi si richiude.
Ogni volta che si crea una socket dobbiamo riferirci al socket descriptor, ci da il permesso di accedere alla risorsa individuandola tra le tante presenti. Un'operazione di lettura e scrittura significa inviare e ricevere file, poi con la chiusura vuol dire che la socket non serve più. Per questo l'astrazione fornita dalla socket è quella del file.

Comunicazione locale. (*slide 7*)

Caso d'uso: abbiamo due applicazioni sullo stesso host (macchina collegata alla rete), che vogliono scambiarsi dei dati, e possono farlo attraverso le socket. Dico al sistema operativo di creare una socket, si crea un canale di comunicazione tra le due applicazioni e si può scrivere e leggere dalla socket: è un meccanismo di **inter process comunication**. In questo caso la socket è definita come **socket UNIX domain**. Le socket sono una libreria ma è giusto avere un'interfaccia,perché le risorse sono comunque nel sistema operativo.

Comunicazione remota via TCP/IP. (*slide 8*)

Comunicazione di due applicazioni sulla rete: abbiamo due host, entrambi creano una socket e comunicano attraverso la rete internet, e quindi è un meccanismo a livelli: la socket interagisce col livello TCP, che interagisce col livello rete e così via. I normali programmi che utilizzano la rete sono realizzati con socket.

Abbiamo due programmi applicativi che girano sui due host che comunicano attraverso il sistema operativo. Si crea un canale di comunicazione tra le due applicazioni, creato dal sistema operativo e dalla libreria. Le due applicazioni dialogheranno tra di loro conoscendo un protocollo, interpretato solo dalle applicazioni e non dai livelli inferiori. Le due applicazioni comunicano tra loro attraverso un protocollo, che non è conosciuto dalla rete ma usato solo dalle applicazioni. C'è un livello inferiore che consegna le informazioni senza capire a cosa servono, poi le informazioni e il significato del messaggio viene interpretato a livelli superiori. Ogni livello ha una certa semantica di comunicazione. Un'applicazione che fa questa cosa, composta da diverse entità distribuite sulla rete è detta **applicazione distribuita**.

Esempio: bitTorrent

Gira da solo ma comunica con altri bitTorrent su altre macchine. L'insieme di processi che scambiano informazioni è detto applicazione distribuita. La connessione di rete effettua solo una consegna dei bit, il protocollo è a carico dell'applicazione.

Se vediamo un'applicazione distribuita basata sul protocollo TCP/IP abbiamo due programmi C (ad esempio) che comunicano con un certo protocollo usando le Socket e le NT API per interagire con il sistema operativo. Il sistema operativo UNIX più lo stack di rete creano il canale di comunicazione. E' un'**architettura cross platform** (***multipiattaforma***), si adatta a più sistemi operativi. Per Windows si usano le **WinSocket**, che comunque sono molto simili alle UNIX socket e interagiscono tra loro.

E' possibile creare un canale di comunicazione, ma ha dei problemi: come faccio connettere i due poli? Come faccio ad essere sicuro che dall'altra parte c'è qualcuno che mi risponde?

Un paradigma fondamentale è quello client/server. Per poter fungere da server un host deve ricordare il proprio indirizzo e mettersi in attesa di connessioni: offre un servizio agli altri host della rete. Un client deve instaurare la connessione e poi può servirsi delle risorse dell'host.

Il sistema operativo, per identificare la socket, utilizza una **quintupla**. A livello di sistema operativo, ad ogni socket è associata questa struttura dati. Nel mondo IP questa struttura diventa indirizzo IP e indirizzo di porto.

- **Protocol**: UDP/TCP
- **localaddress**: Indirizzo IP locale
- **local process**: Porto
- **foreign address**: Indirizzo IP dell'host straniero
- **foreign process**: Processo straniero

Praticamente se arriva un pacchetto dalla rete e sulla macchina ci sono x processi il sistema operativo non sa dove dare questo pacchetto. Quindi con l'indirizzo IP si arriva alla macchina e col numero di porta si identifica il processo cui è destinato il pacchetto.

Server concorrente e iterativo. (slide 14)

- **Server iterativo**: fino a quando non ho finito di servire il primo client, non passo ad un altro client.
- **Server concorrente**: tutti i client si collegano contemporaneamente e quindi il server deve aprire più canali, quindi più socket, dove ogni socket si identifica con quella quintupla. Il server riceve i pacchetti destinati a lui e il sistema operativo li smista.

Il server iterativo è più semplice e si crea più fila. Il server concorrente è più complicato, gestisce processi in parallelo, richiede più risorse, ma consente di soddisfare più richieste contemporaneamente.

Connection Oriented.

Un primo tipo di caratteristiche è quello legato alle caratteristiche Connection-Oriented (corrisponde a **TCP**). Il canale ha più proprietà. Possiamo fare affidamento sul fatto che, se inviamo due messaggi uno dopo l'altro, siami sicuri che il secondo messaggio arriva dopo il primo, e non ci sono inversioni all'interno del canale. È affidabile e c'è correzione degli errori. Questo canale trasporta flussi: si crea un flusso continuo tra un host e un altro. E' dedicato: se due host stanno parlando tra di loro non può entrare un altro host con altri processi.

Datagram.

Il secondo paradigma è a Datagram (corrisponde a **UDP**), che non ha le caratteristiche di prima, perchè prova a recapitare il messaggio ma non è affidabile, e il messaggio può andare perso o può essere corrotto. Non preserva l'ordine dell'informazione. Quando chiediamo al sistema operativo di inviare un certo messaggio non è detto che dall'altra parte il messaggio arrivi integro: può arrivare diviso in frammenti.

```
int num = write (socket, buf, 10);
```

Questa funzione restituisce il numero di byte inviati, che potrebbe essere anche minore di 10. In caso, se ne inviamo 5, dobbiamo fare una nuova chiamata: si passa di nuovo la socket descriptor e si manda il messaggio composto da buf+num. In questo modo si preservano i limiti dei messaggi.

Il paradigma è condiviso perché posso ricevere messaggi che non sono destinati a me: devo controllare i messaggi, potrebbero essere di un altro processo. Nulla vieta di creare un pacchetto con porte di destinazioni diverse. Nessuno controlla da dove arriva il pacchetto perché non c'è la creazione del canale. A livello di protocollo bisogna riconoscere i pacchetti ricevuti.

Il canale secondo paradigma Datagram è più "economico" del canale connection oriented, ed è più facile da instaurare. La connection oriented è affidabile e controlla il flusso (non permette l'invio di troppi dati per evitare congestione). Il datagram non controlla il flusso e magari può creare congestione a livello locale o a livello di rete. La connection oriented è affidabile, se lo inviamo siamo sicuri arriverà. Il suo svantaggio è l'instaurazione della connessione che nel caso datagram non serve. Nel datagram c'è un basso overhead, bisogna mandare meno informazioni nella rete. Non c'è nessun controllo sulla consegna però.

Naming / Binding. (*slide 20*)

Viene dato un nome alla socket e questo viene reso noto ai client.

1. creo la socket
2. associo il nome alla socket (indirizzo IP e port number)
3. accetto la connessione

Byte stream e Datagram. (*slide 21*)

Posso impostare il protocollo usato per il trasferimento dati specificando uno di questi due attributi. Quando creo una socket questa può essere o **sock_stream** o **sock_diagram** per specificare il tipo di canale voluto.

Progettazione di un server TCP. (*slide 22*)

Si crea la socket facendo richiesta ad un sistema operativo. Si fa il binding. Appena arriva la chiamata da un client il server lo accetta: legge cosa vuole il client e risponde. E' analogo a quello che si fa con un file in UNIX. L'end point corrisponde alla socket descriptor.

`int fd = open (...)`	*restituisce un file descriptor*
`write(fd, buf, 10)`	
`read(fd, buf,10)`	
`close (fd)`	*uguale per tutti*

[**fd**: file descriptor o socket descriptor]

La close va sempre inviata, altrimenti il sistema operativo aspetta che sia passato un certo tempo prima di chiuderla automaticamente. Con il TCP nel caso dell'open viene fatto un three way handshake. Per la chiusura della connessiona si usa un four way handshake.

Con `socket()` creo il file descriptor, con la `connect()` stabilisco la connessione, con `close()` la chiudo. Per creare un applicazione che si connetta al web devo richiamare queste operazioni nella libreria socket. Nel caso in cui serva, si usa la `socket()` per crearla, il `bind()` per associarle un nome, creare un indirizzo, la `listen()` per mettersi in ascolto e la `accept()` per accettare la connessione. Da qui, `read()`, `write()`, `read()` e `close()` per chiudere la connessione. Sono chiamate di sistema prescritte, da effettuare in quest'ordine.

Progettazione di un server UDP. (slide 27)

Nel caso UDP manca `connect()`, ci sono direttamente `send()` e `write()`.
Nel caso server si fa `socket()`, `bind()`, `recvfrom()` in cui posso specificare anche da chi voglio ricevere. Il client fa una `socket()`, `bind()` (facoltativo, ignoriamolo cit.) e poi si fa una `sendto()` specificando a chi inviare datagrammi. Il client può fare una `recvfrom()` per ricevere dati dall'esterno. Nel caso TCP c'era una `connect()` da parte del client e una `receive()`, non la `receivefrom()` perché già sapevo da chi ricevere avendo stabilito una connessione.

La programmazione delle socket. (slide 31)

Strutture dati per le socket.

[vedi slide]

Reti di Calcolatori - 26/10/2012

Lezione 14. Network Address Translation (NAT) e IPv6

Network Address Translation (NAT).

Esistono degli indirizzi riservati per l'utilizzo di reti private. Questi indirizzi possono essere assegnati dall'amministratore di rete, e possono essere utilizzati o all'interno di reti private oppure quando queste reti private vogliono poi interconnettersi a internet, e quindi nasce il problema di consentire a host che sono configurati con indirizzi IP privati di scambiare informazioni con host che si trovano sull'internet pubblica.
Quasi tutti gli internet provider assegnano alla linea ADSL un unico indirizzo IP. Il router ha un indirizzo assegnato in modo dinamico, il router periodicamente deve fare richiesta di indirizzo IP, che va continuamente rinnovato e riconfigurato. Tutto questo avviene tramite il DHCP automaticamente. Il router ha una serie di connessioni ethernet per le connessioni casalinghe e la porta WAN per il collegamento ADSL con il doppino telefonico.

Una rete locale, dal punto di vista IP, è una subnet, e viene configurata con un range di indirizzi IP privato. Supponiamo di voler configurare la rete locale con subnet 192.168.0.0 con /24 come netmask, che è la configurazione di default. L'interfaccia router avrà un indirizzo privato, ad esempio 192.168.0.1, e i vari computer avranno indirizzi privati che possono essere assegnati automaticamente da un server DHCP all'interno del router, che quando i dispositivi si accendono assegna loro dinamicamente degli indirizzi.
Quando una macchina vuole comunicare con un server che ha un suo proprio indirizzo pubblico, genera un pacchetto che invia al router avente come indirizzo destinazione il sorgente S, e come indirizzo mittente l'indirizzo privato H1. Questo router non può inoltrare il pacchetto sul collegamento geografico così come gli arriva, perchè, se lo facesse, il router successivo nella catena lo scarterebbe, perchè vedrebbe un indirizzo privato come mittente. Se pure arrivasse al destinatario, perché di solito le scelte di routing sono fatte sull'indirizzo di destinazione, il destinatario non saprebbe a chi rispondere, perché di host con indirizzo IP H1 ce ne sono molti.
E' nata così la **traduzione degli indirizzi (*NAT*)**.

Vogliamo prevedere che tutti i pacchetti che sono generati dalla rete privata e vadano su internet abbiano come indirizzo mittente non la macchina ma l'indirizzo pubblico dell'interfaccia del router, che avrà sicuramente un'interfaccia con un'indirizzo IP pubblico per la connessione ad internet. Con questa tecnica, c'è un solo indirizzo pubblico e tutte le macchine che si trovano nella rete privata sono rappresentate dall'unico indirizzo pubblico del router.
Questo pone un problema: come si fa a discriminare la macchina di provenienza quando arriveranno i pacchetti di risposta? Quando arriveranno i pacchetti di risposta, il router che sta in mezzo a quale macchina della rete privata li inoltrerà? Abbiamo bisogno di un'informazione per fare il demultiplexing, che non possiamo trovare nell'header IP, (a meno di non modificare l'ip, ma questo non si può fare), ma abbiamo bisogno di un'altra informazione. Si va a prendere un'informazione che non è pertinente a livello rete, ma è pertinente a livello di trasporto, che è il port number, che viene utilizzato per collegare i pacchetti di risposta che arrivano dall'esterno con gli host che li hanno generati.

Una prima tecnica (poco furba) è di dare un numero di porto per ogni macchina, ma questa allocazione statica non va bene, perchè una sola macchina crea tante porte sul web, varie connessioni TCP per scaricare il parallelo file HTML da più macchine, ecc…

Non può essere fatta staticamente quest'assegnazione, ma va fatta **dinamicamente**. Viene fatta creando una **tabella di traduzione** che viene gestita dal router che realizza questa funzione di address translation (NAT).
Quando il router riceve un pacchetto dalla sua rete alla rete esterna salva nella tabella l'indirizzo e il porto del mittente e anche la nuova corrispondenza di indirizzo e porto verso l'esterno. L'indirizzo con cui i pacchetti vanno verso l'esterno, è sempre lo stesso, perchè il router ha solo un indirizzo verso l'esterno, e il port number sorgente del pacchetto che esce viene riassegnato perchè perché può accadere che due macchine diverse della rete privata utilizzino lo stesso numero di porta. Questo è possibile, perchè gli host scelgono i numeri di porto **casualmente** tra i numeri di porto nella macchina, a seconda dei porti che sono liberi sulla macchina in quel momento.

Due macchine differenti con lo stesso numero di porta creerebbero un'ambiguità. Tipicamente, il router riassegna gli indirizzi di porto in maniera sequenziale, rispetto al primo disponibile nel range. In realtà, siccome questo spazio di indirizzamento è limitato, e può contenere solo 2^{16} numeri di porto, una volta che sono passati 65mila pacchetti, i numeri di porto sarebbero stati utilizzati tutti. Allora, queste entry che vengono utilizzate sono associate ad un timer, fatto partire quando la entry viene scritta, e allo scadere del timer, quella corrispondenza viene cancellata, in modo tale che il numero di porta in uscita possa essere opportunamente riassegnato.

Di fatto, il router deve gestire **due tabelle distinte**: una per **TCP** e una per **UDP**, perchè gli spazi di indirizzamento di port number di UDP e TCP sono separati, in quanto concettualmente diversi. Infatti, posso avere nella stessa macchina due processi che hanno porta 80 TCP e porta 80 UDP.
Il router manipola il pacchetto a livello rete andando ad avvalersi di un concetto a livello trasporto. Lo può fare perchè allo stesso concetto di port number esiste sia in UDP e TCP, e il port number in entrambi i casi hanno lo stesso range di indirizzamento. Il meccanismo è trasparente al sistema operativo, e la macchina continua a lavorare su quella porta.

Le azioni che fanno i router NAT quando vedono i pacchetti sono:

- Per i pacchetti in uscita, il router sostituisce la coppia indirizzo IP sorgtente e numero di porta con la coppia indirizzo IP NAT e nuovo numero di porta.
- Il server risponde con un paccchetto che ha per destinazione l'IP del router e come numero di porta quello assegnato dal NAT
- Il NAT ha la tabella di assegnazione del numero di porta e quando arrivano i pacchetti in entrata fa l'operazione duale: sostituisce la porta originaria e l'IP privato della macchina origine. Dal numero di porta il router capisce a chi è indirizzato il pacchetto.

C'è bisogno della tabella per fare la traduzione inversa.

Il fatto che lo spazio dei porti sia limitato e condiviso da tutte le macchine della rete privata è una limitazione, perchè posso avere solo 60mila connessioni simultanee. L'uso dei NAT ha il difetto peggiore di violare il fatto che il pacchetto debba viaggiare nella rete inalterato, e quindi è un concetto controverso, non visto spesso di buon occhio. I NAT sono un'istanza particolare di **middle boxes**, che rompono la continuità della trasmissione del pacchetto alterando alcuni campi. I router NAT quindi non si comportano da veri router. Secondo i puristi, il problema che ha generato questa tecnica dovrebbe essere risolto con l'uso di indirizzi IPv6. Uno dei fattori che ha causato l'aumento dei terminali è la connessione dei cellulari, ad esempio.

Port forewarding e Nat traversal problem.

La tecnica più semplice è quella di configurare una traduzione di indirizzo di tipo NAT staticamente nel router: se nella macchina 10.0.0.1 ho un web-server in ascolto sulla porta 80, e non ho nessun

altro web server, allora posso dire al router che tutti i pacchetti che arrivano con porta di destinazione 80, devono essere tradotti e inoltrali a quel web-server 10.0.0.1/80, per non cambiare il numero di porta. Viene cambiato comunque l'indirizzo, perchè l'indirizzo pubblico diventa l'indirizzo privato della rete. Non posso avere più server sulla stessa porta, la convenzione è farne girare uno sulla porta 80 e un altro sulla 8080, ad esempio. Questa è una soluzione per reti di piccole dimensioni, la cosa già non funziona con 5-10 server pubblici.
Questa tecnica di configurazione di router viene detta **port forwarding**, perché cambia solo l'IP. Anche questa tecnica non risolve il problema di **NAT traversal** per tutte le applicazioni,soprattutto non va bene per applicazioni peer-to-peer, dove un host funziona sia da client che da server. Per questo tipo di applicazioni, la scelta dei numeri di porta non può che essere dinamico, allora occorrono dei meccanismi più sofisticati.

UPnP: Universal Plug and Play

Il meccanismo più efficace sta nella soluzione di Universal Plug and Play, che consente di realizzare una connessione tra un dispositivo e un router attraverso un protocollo IGD, dove un host può conoscere l'indirizzo IP pubblico del router, mentre senza questo protocollo non potrebbe saperlo. Un host che si trova dietro un NAT tipicamente non sa né di essere nattato né quale sia l'IP pubblico del router. Per il peer to peer serve perché non si può associare un proprio indirizzo privato (ce l'hanno tutti). Un'altra azione che si può effettuare è quella di chiedere al router di aggiungere o eliminare delle azioni di traduzione di porto, con delle scadenze temporali: il peer riserva per i suoi usi il numero di porte da 5000 e 5010, ad esempio. Tutti quelli che arrivano in quell'intervallo vengono mandati lì lasciando inalterato il port number: è un port forewarding dinamico. E' un **port forwarding dinamico**. In questa maniera posso mantenere in esecuzione più istanze di applicazioni peer-to-peer sulla mia rete privata. C'è un'azione di richiesta che può avere una risposta negativa o positiva da parte del router. In questa maniera, quando l'host ha la risposta può annunciarsi sulla rete dicendo il proprio range di porti. Fatta una richiesta per le porte questa potrebbe anche rispondere negativamente perché le porte sono già state assegnate. Una volta che ha la risposta gli host possono informare gli altri peer della rete sulle sue porte di contatto.
BitTorrent è un protocollo che definisce delle modalità di interazione tra peer

Skype.

Relaying: la comunicazine tra un client che ha un indirizzo privato e un client che sta dietro ad una rete nattata, avviene grazie ad una macchina che si mette in mezzo. La comunicazione avviene attraverso una terza entità, una specie di server noto all'applicazione. All'inizio, il client si presenta col proprio indirizzo pubblico del router e stabilizza una connessione. Quando un altro client vuole parlare, si collega allo stesso relay e attraverso queste due comunicazioni punto punto che si stabiliscono sempre dal client verso il relay, questi due possono parlarsi e, per esempio, negoziare dei numeri di porta. Alla fine, la comunicazione può avvenire attraverso il relay, oppure attraverso il relay si ha uno scambio di informazioni per stabilire la connessione. La connesione tra i due peer in realtà si stabilisce dal client che vuole essere contattato. Se entrambi sono dietro NAT la connessione diventa più complicata.

IPv6.

Il problema del passaggo da 4 a 6 sta nel fatto che questa transizione coinvolge sia gli end-system che i router. Ci sono computer obsoleti e computer moderni, e router che devono essere riconfigurati. La transizione di versione IP da 4 a 6 nei router è particolarmente onerosa, perchè non è detto che un router fatti 15 anni fa supporti il passaggio alla versione 6, per l'elaborazione in hardware. Resta comunque non trascurabile il problema della transizione, perchè riconfigurare milioni di reti e quindi miliardi di dispositivi è un'operazione che richiede tempo. Una serie di altri

protocolli di routing dovrebbero anche essi essere aggiornati. Una transizione tra IPv4 e IPv6 non può avvenire nell'arco di periodo limitato, ma può essere solo un upgrade graduale. Non possono che coesistere nella rete come già accade adesso. IPv6 già è utilizzato in parallelo a IPv4, e i due protocolli coesistono.

Caratteristiche.

Il motivo scatenante è stato l'allargamento dello spazio di indirizzi IP da assegnare alle interfacce. Nel fare questo, una serie di altre cose sono state modificate, al fine di aggiungere o eliminare alcune funzionalità di IPv4. In IPv6 gli indirizzi sorgente e destinazione sono da 128 bit. Si passa da 4 byte a 16 byte, ho un incremento del 400%. Questo spazio di indirizzamento è gestito in maniera diversa, cioè alcuni range di indirizzi sono riservati per una serie di funzioni particolari e speciali. Una delle cose più significative è l'aggiunta di uno speciale tipo di indirizzi che consente la trasmissione di tipo **anycast**. Una trasmissione da un host ad un altro è detta **unicast**. Per applicazioni particolari l'IP nella versione 4 supporta una trasmissione da una sorgente a più destinazioni, il **multicast**.La trasmissione multicast è la trasmissione di un pacchetto da una sorgente ad una molteplicità di host. Un gruppo di host si connette con un determinato indirizzo e i router di una rete inoltrano i pacchetti in modo tale che da un unico pacchetto originale si formino gli n pacchetti che arrivano agli n host.
La trasmissione anycast è una trasmissione di un pacchetto ad un unico host, dove però la destinazione può essere scelta dalla rete all'interno di un insieme di possibili destinazioni: un certo numero di host si iscrivono ad un gruppo anycast e la rete, quando vede arrivare un pacchetto, inoltra il pacchetto ad uno degli host iscritti al gruppo che è quello più indicato a ricevere quel pacchetto.

Il criterio generale che si è adottato in IPv6 è quello della semplificazione dell'header: alcuni campi ritenuti non più necessari sono stati eliminati o resi opzionali. Nonostante gli indirizzi di IPv6 sono più lunghi di quelli IPv4 quattro volte, **l'header** è solo il **doppio**.
E' stato migliorato il modo nel quale si modificano le opzioni che si aggiungono all'header. Si è implementato un meccanismo che consente l'aggiunta di nuove opzioni in maniera più semplice. È stato aggiunto un supporto della **Quality of Service**: perchè possa garantire che forma di qualità tra due end system attraverso una commutazione di pacchetto (la qualità è quantificata da delay, jitter, throughput etc), per poter dare una garanzia, ad esempio, al throughtput, devo potere discriminare i pacchetti della comunicazione nella rete e trattarli meglio rispetto ad altri pacchetti che viaggiano nella rete relativi ad altre comunicazioni. Occorre che i router non abbiano più una gestione FIFO, ma occorrono dei meccanismi di schedulazione dei pacchetti che siano in grado di garantire una maggiore priorità a dei pacchetti rispetto ad altri. Quindi nei router deve esserci la possibilità di gestire code di **schedulazione** differenti per diversi tipi di comunicazione e il router deve avere delle informazioni per discernere i flussi di informazioni con diversi tipi di comunicazione che hanno certe prerogative. Tutte le tecniche di qualità del servizio sfruttano il concetto di **flusso**, che in una rete a datagrammi, di commutazione di pacchetto, non esiste. Occorre che due pacchetti scambiati successivamente da A e B siano riconosciuti allo stesso flusso. Il router può vedere il pacchetto e riconoscerlo uguale (stessi indirizzi e destinazione) ad un altro e quindi può capire che due pacchetti sono dello stesso flusso (classificazione indiretta). Identificare un flusso in una rete, cioè assegnare un pacchetto ad uno specifico flusso può essere fatto esplicitamente o implicitamente.

L'assegnazione ai flussi può essere esplicita, usando un campo specifico in IPv6 che mi consente di dire il pacchetto che viaggia nella rete da A verso B, non è proprio un pacchetto isolato, ma è uno che prima ha dei pacchetti e poi ne ha degli altri. Questa identificazione avviene attraverso un campo detto **flow label**. I progettisti di IPv6 hanno messo nell'header questo campo così che chi ci

vuol costruire sopra lo può fare: questo apre la strada a trattamenti di circuito virtule anche se la rete nel suo modello base è a datagrammi. La flow label è nell'header non nelle opzioni, ma spesso non viene utilizzata.

Esistono altri meccanismi che servono a rispondere a esigenze di sicurezza. Per esempio, in una rete IP normale non si ha nessuna garanzia dell'identità del mittente di un pacchetto. Nulla vieta, nella rete, ad un host di inviare un pacchetto che ha come indirizzo IP sorgente non l'indirizzo IP vero ma l'indirizzo IP di un'altra macchina (***IP spoofing***), cioè un host invia dei pacchetti con un IP non vero ma con un altro IP di un altro host. Chi fa IP spoofing non ha intenzione di comunicare, ma ha intenzione di dare fastidio. In questo modo il destinatario pensa che il pacchetto l'abbia inviato un altro e non il reale mittente.

Header di IPv6. (slide 11)

L'header di IPv6 consiste in una parte minimale comune e una parte di opzioni. Un pacchetto che non ha estensioni ha un header minimale: numero di versione, campo di **priorità**, campo **flow label**, **lunghezza del payload**, il campo di **next header**, che, oltre ad essere un aggancio alle opzioni, definisce il protocollo del pacchetto, e in realtà, mediante degli appostiti valori, il next header può anche essere il puntatore ad ulteriori opzioni dell'header. Nell'opzione ci sarà un **ulteriore campo next header** che mi punta al resto dell'header. L'**hop limit** è l'equivalente del TTL. Ci sono poi **indirizzo sorgente** e **indirizzo destinaizone**.
Non supporta la frammentazione, e se c'è la necessità di frammentazione, il router scarta il pacchetto e manda a chi ha inviato il pacchetto un messaggio ICMPv6 all'host mittente che lo avvisa del fatto che il pacchetto è stato buttato e deve essere fatto più piccolo. Chi si occupa di trasmettere poi il pacchetto più piccolo è il mittente. Sarà poi il destinatario a riassemblare il tutto.
In generale comunque gli host si accorgono sull'MTU, la dimensione del pacchetto da inviare.
In IPv4 il mittente non veniva avvisato della frammentazione, ma essa avveniva in maniera del tutto trasparente. Il mittente non doveva fare niente e l'onere della frammentazione lo avevano i router, non sapeva di questo evento e continuava a persistere nell'invio di questi pacchetti grandi. In questa maniera, invece, il mittente ha l'avviso e non commette più l'errore.

Non c'è più **la checksum**. Questo perchè, per vari motivi, si scarica il router di questo compito. I router non sono più costretti a ricalcolare la checksum perchè quest'informazione non c'è proprio. Il problema, è che il pacchetto potrebbe perdersi nella rete per l'assenza di questo campo. L'arrivo al destinatario di un pacchetto corrotto è un fatto grave, ma delle conseguenze non si fa carico IP, ma se ne fa carico TCP, che calcola la checksum sull'intero pacchetto, per assicurarsi che il pacchetto arrivi al destinatario in maniera non corrotta.
La possibilità di aggiungere campi opzionali è introdotta tramite il campo next header: c'è il "**packet too big**" e altre **opzioni di management di gruppi multicast**.

C'è un campo di **priorità** che codifica, attraverso dei valori, le priorità con le quali il pacchetto dovrebbe essere elaborato all'interno dei router. Nella versione 4 i router ignorano ciò che c'è scritto in questi campi, però potrebbe essercene qualcuno che lo considera.

Il campo **next header** è un campo di 8 bit, e alcune configurazioni servono a identificare speciali protocolli di trasporto. Sono gli stessi valori del campo protocol nell'IPv4. Significa che due end-point possono comunicare con il protocollo TCP anche se viene utilizzato IPv6, poiché i pacchetti sono veicolati nella stessa maniera: la modifica del protocollo di livello rete non richiede la modifica a livello trasporto. Altri valori di questo campo serrvono ad identificare o altri protocolli di livello trasporto, o altri protocolli speciali (ICMP, IGMP, ICMPv4,) oppure una serie di altre configurazioni sono utilizzate per campi opzionali, come routing header, fragment header, e così

via.
Il valore 4 nel next header identifica IPv4, e ciò significa che sto mettendo in un pacchetto IPv6 un'intestazione IPv4, che poi contiene il payload. In questa maniera, sto trasmettendo su parte della rete da un host IPv6 ad un'altra interfaccia un pacchetto IPv4. Ci sono varie possibilità di realizzare il nesting di molteplici tipi di header l'uno nell'altro. Ho un pacchetto IPv6 che trasporta un TCP in maniera nativa. Poi ho un IPv6 che trasporta il TCP con un'estensione... **(slide 16)**.
Il valore 63 nel next header è il **routing header**. Serve a capire i campi come sono strutturati e che forma hanno. C'è un header che punta ad un altro header e questa sequenza a puntatori può essere considerata come una lista linkata. Si può richiedere che il pacchetto segua una certa sequenza di router: chi manda il pacchetto definisce anche il circuito. Questa cosa è spesso un'intenzione perché gli ISP non si fanno dire il pacchetto che percorsi deve fare, li sceglie secondo i suoi criteri. Anche perché altrimenti si potrebbero fare attacchi alla rete. Poi ci sono varie possibilità di estensione.

Per quanto riguarda il formato degli indirizzi, l'intero range di indirizzi è individuabile in intervalli cui sono assegnate delle funzioni particolari. Il primo range inizia in binario con 8 zeri: rappresenta 1/256 dell'intero spazio di indirizzamento e sono usati per mappare automaticamente gli indirizzi IPv4. Noi trascriviamo i pacchetti in forma esadecimale in gruppi da quattro separati da due punti. Si possono sostituire gruppi di quattro zero con **::**. Gli indirizzi IPv6 sono scritti in forma esadecimale come 8 numeri naturali separati dai due punti.

Esiste una tecnica di mapping che mi consente di rendere IPv6 con IPv4 mettendo a sinistra tutti zeri. Questo tipo di rappresentazione è usato nei nodi a doppio stack che comunicano con IPv6 su un'infrastruttura IPv4 **(slide 20)**.

Tunneling. (slide 21)

Ci sono delle porzioni di rete di backbone che non comunicano ancora in IPv6: dal punto di vista mittente destinatario il pacchetto ha gli indirizzi giusti. Ma come si fa? Con il **tunnelling(?)** si perdono molte informazioni e per non perderle si fa il contrario: si trasmette il pacchetto IPv6 con il suo payload dentro un pacchetto IPv4. Questo contenitore IPv4 non ha veri indirizzi mittente e destinatario ma ha quelli di due router in mezzo che supportano l'IPv6. Questa tecnica di imbustamento è detto tunnelling o anche incapsulamento. Il router incapsula l'IPv6 nell'IPv4. Il tunnel così stabilito è solo una parte del percorso complessivo. I router in mezzo che usano solo IPv4 non vengono visti ma c'è il tunnel al suo posto.

Reti di Calcolatori - 30/10/2012

Lezione 15. Routing

Routing nelle reti a commutazione di pacchetto.

A livello datalink abbiamo tanti host che i pacchetti devono seguire per arrivare dal mittente al destinatario. Per motivi di ridondanza, esistono più percorsi disponibili. La scelta di questo percorso vene fatta a livello rete, dove a livello trasporto ci si occupa della comunicazione mittente ▭ destinatario. C'è un problema di determinazione univoca delle interfacce, e dal punto di vista del routing occorre risolvere questo problema con degli algoritmi che partono dalla topologia formale della rete. Tipicamente ci si avvale della modellazione della rete come un **grafo** dove i **nodi** sono i router e gli **archi** sono i link fisici. Mettiamo in evidenza i router, gli end-system partecipano anche loro all'instradamento, ma in questo caso l'instradamento è più semplice: gli end system hanno un unica interfaccia di instradamento alla rete tramite cui si connette alla rete locale dove c'è un solo router. Ad esempio, nella rete IP, la scelta di instradamento dell'host mittente prevede una prima biforcazione: o il pacchetto deve essere consegnato nella rete locale, o il pacchetto deve essere inoltrato ad un router in una rete non locale. Al primo hop il pacchetto è consegnato al router di quella rete (questa è l'unica scelta) e da lì inizia il processo di instradamento. Ci sono anche situazioni più complicate con una rete locale con più router disponibili.

Nella scelta dei percorsi, bisogna individuare un percorso per ogni pacchetto in funzione della destinazione, quindi è la destinazione del pacchetto che guida la scelta di instradamento. Gli allgoritmi tendono a scegliere i percorsi che ottimizzino una certa funzione obiettivo. Ci sono algoritmi che scelgono percorsi più corti, ma questo discorso si può estendere associando una metrica di costo che ad ogni link che pesa il costo dell'attraversamento del pacchetto in quello specifico link. Una classe di algoritmi è quella che determina il **cammino a costo minimo**. Il costo di un cammino si ottiene sommando il costo di ogni link fisico, quindi la **metrica** è **additiva** (ma non è sempre così).

Parametri che possono essere considerati nella scelta dei percorsi

- **Bandwidth**: capacità di un link, definita per bit/secondo. Esempio di metrica non additiva.

Supponiamo che il collegamento diretto tra A e C avviene attravesro 100 Mbit/s, ma quella da A a D è sempre 100 e tra D e C sono 10, già so che sul percorso tra C e D la capacità del link è minore. Ciò che limita il througtput è il collegamento tra D e C. La capacità di un percorso risulta essere la minima capacità di un link che fa parte di quel percorso.

- **Delay**: Il ritardo è la somma dei ritardi. In questo caso la metrica è additiva
- **Hop Count**: Numero di router da attraversare: si associa il costo unitario a ciascun link e si sommano tutti gli hop. Si sceglie il percorso a costo minimo, cioè col minimo numero di hop (metrica additiva).

Graph abstraction: costs. (slide 7)

- **Link**: collegamento diretto.
- **Path**: percorso, concatenazione di link.

Il costo di un link in generale non è simmetrico. Cioè il costo del collegamento da x a y non è sempre uguale al costo di collegamento da y a x. Nel caso, lo si esplicita.

Occorre esprimere il costo di un collegamento tra una coppia di router x e y, e si rappresenta questa grandezza con **c(x,y)**, che rappresenta il **costo del collegamento**, che si può ridurre al costo di uno specifico link. Questo costo del collegamento da x a y non è detto che sia uguale al costo del collegamento tra y e x. Accade solo se il costo è simmetrico. Il **costo di un singolo percorso** è dato da: $c(x_1, x_2, x_3, \ldots, x_n) = c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{n-1}, x_n)$.

Un algoritmo trova i percorsi di costo minimo su un grafo, che vanno calcolati tra qualunque nodo sorgente e qualunque nodo destinazione. Un algoritmo può essere eseguito avendo la conoscenza della topologia della rete e dei costi che poi guidano la scelta dei percorsi attivi.

Il processo di routing. (slide 8)

Il problema che si pone è chi è che prende queste decisioni e l'entità che prende queste decisioni come ottiene le informazioni sul costo e sul numero di link. In realtà, a questo **processo decisionale**, partecipano in maniera attiva i router, che svolgono anche la funzione di **forwarding** (attività che il router fa con grande velocità e che consiste nell'accettare pacchetti in ingressi e nello smistare velocemente questi pacchetti verso i link in uscita grazie ad una tabella che il router consulta ogni volta che arriva un pacchetto e in cui ci sono scritte le informazioni per l'instradamento). Questo attraversamento del router da parte del pacchetti non è una semplice copia, ma richiede anche una manipolazione del pacchetto, in quanto richiede di decrementare il TTL e verificare la checksum. Quest'operazione deve avvenire più velocemente possibile, in quanto quest’attività determina la velocità e le prestazioni del router.

Affianco a quest'attività di forwarding, il router partecipa all'attività di **routing**, svolta a livello globale di rete: la rete determina il percorso di un pacchetto. Questo processo decisionale non viene preso una volta per tutte: il grafo che rappresenta la tipologia della rete non è detto che contenga informazioni che sono valide indefinitamente nel tempo, perchè può accadere che il router si guasti, e quindi vengono meno tutti i link che collegano quel router al resto della rete. Questa cosa deve essere rilevata dai router vicini, e comporterà un cambiamento topologico della rete che comporterà, a sua volta, un ricalcolo delle scelte di instradamento (da parte del resto dei router della rete). Oppure, può succedere che il router sta andando in congestione, e questa cosa richiede una reazione dai router vicini, in modo che non gli inoltrino più altro carico (traffico) e cambino le scelte di instradamento.

Esempio. (slide 10)

Abbiamo una rete fatta da 4 router. Agganciati a questi router ci sono delle reti locali. I router devono scambiarsi delle informazioni. La tabella di routing del router X dice al router che quando arriva un pacchetto con destinazione A, questo può essere consegnato con 0 passi al destinatario che sta nella rete A.

Il risultato del processo di routing è la **materializzazione di tabelle** di questo tipo nei router.

Tecniche di routing. (slide 11)

Sono le modalità attraverso le quali viene scelto il percorso.

La prima tecnica possibile è la scelta del percorso attraverso l'indirizzo del destinatario (scelta usata nelle reti IP). In questo tipo di reti, ogni pacchetto ha l'indirizzo del destinatario, che è contenuto nelle tabelle rei router usato come chiave per accedere alle tabelle di instradamento. Viene inviato in base a scelte effettuate precedentemente in base a scelte già fatte. Il router lavora su due scale di

tempi: c'è il processo di forwarding che lavora in tempi molto brevi, perchè si deve fare soltanto una ricerca in tabella (***lookup***), e poi c'è il processo di routing, che lavora su una scala dei tempi maggiore, e questo processo di scelta dei percorsi deve essere sempre attivo anche se non ci sono cambiamenti nelle tabelle perchè deve sempre stare attento a modifiche. Se dovesse venire meno un router, per qualche decina di secondi i pacchetti non arrivano più al destinatario, ma i router si parlano tra di loro, si aggiornano le tabelle, e così si cambia percorso. L'inoltro di un pacchetto coinvolge tempi nell'ordine dei microsecondi. Questa tecnica è classica nelle reti di commutazione di pacchetto a datagrammi.

In alcune tecnologie di rete si usa un'altra tecnica, ideale per le reti a circuiti virtuali, dove la scelta dei percorsi viene fatta all'atto della creazione di un circuito virtuale tra una coppia di terminali. Quando viene scelto il percorso, si instaura il circuito e i pacchetti non sono più etichettati con l'indirizzo del destinatario e sono etichettati con un identificativo che serve ad associare i pacchetti al circuito. Questo marcatore messo nel pacchetto e usato da ogni router serve ad identificare il circuito virtuale instaurato attraverso ad un protocollo di segnalazione. Viene creata una connessione prima di inviare dati, e in questa connessione si fanno le scelte di instradamento che dovrà fare poi il pacchetto. Il circuito è scelto dall'insieme di router o switch. In seguito ad un'azione esplicita di segnalazione, analoga ad una telefonata, c'è quest'azione dei router: instaurato il circuito, se si ottiene una risposta positiva dai router, i router del percorso sono configurati in modo tale da inoltrare i pacchetti da quel momento in poi secondo quel percorso. Questa tecnica è detta di **label swapping**.Questo tipo di approccio è usato in X25, APM, TPLS...

Nelle reti a datagramma può accadere che due pacchetti con stessi indirizzi mittente e destinatario inviati consecutivamente seguano due percorsi differenti, perchè un router potrebbe cambiare la sua tabella di instradamento nel frattempo. In realtà le tabelle di instradamento non cambiano frequentemente, sono abbastanza stabili. Questi eventi di percorsi alternativi ed eventi fuori sequenza sono abbastanza rari.

Tabella di routing per un router IP.

La tabella contiene delle entry, dove ogni entry è collegata (*relativa*) ad una certa subnet, che contiene indirizzo IP e netmask, e a questa subnet di destinazione è associato un link di uscita. In base all'entry matchata nella tabella si fa una certa scelta di instradamento. Queste entry contengono non solo la destinazione, ma anche una netmask, perchè in realtà è opportuno che quando esistono percorsi comuni a reti che hanno una parte di indirizzo uguale, è opportuno che queste entry vengano il più possibile fuse tra loro.
Se avessi due reti 200.23.16.0/24 e 200.23.54.0/24 e non c'è nessun'altra entry che va in conflitto con queste, e allora queste due reti possono essere fuse in una sola voce 200.23.0.0/16, cioè ho trovato un **pattern comune**, e le ho rappresentate come una sola. Il vantaggio è avere una tabella più compatta: questo tipo di operazione è necessaria e porta dei benefici, perchè fare una ricerca su una tabella che ha meno entry richiede meno tempo. Per questo motivo, la logica hardware dei router applica un algoritmo di ricerca che viene detto di **longest prefix match**: quando arriva un pacchetto con una specifica destinazione, vede tutte le entry che matchano quella destinazione, e la scelta finale di instradamento viene fatta scegliendo la entry più lunga, che matcha l'indirizzo di destinazione con un numero di bit più grandi.

Esempio.
Immaginimo di avere una rete 8.8.0.0/16. E' una rete aziendale fortemente subnettata perché troppo grande per avere tutti gli host in un solo comprensorio fisico. Però si annuncia come un'unica rete. Però in realtà può accadere che al suo interno ci sia una biforcazione: 8.8.4.0/24 e poi c'è il resto della 8.8.0/16 in un altro router. La tabella di routing del router principale ha una entry che dice che

8.8.0.0/16 è raggiungibile attraverso un dato link. Ma c'è anche un'altra entry che dice che 8.8.4.0/24 è raggiungibile attraverso un altro link (nella biforcazione del router principale).

Quando al router esterno arriva 8.8.4.18 manda il pacchetto nel router principale che manda il pacchetto nell'entry che matcha di più l'indirizzo di destinazione. Nei router esterni però c'è un'unica entry in tabella: quando il pacchetto arriva ad un router esterno, a monte, questo sa che qualunque sia la destinazione bisogna inviare il pacchetto al router principale prima della biforcazione.
Però le entry non sono tutte della stessa lunghezza: per questo si fa il lookup del longest match.

Costruzione delle tabelle di routing.

Concettualmente, sono possibili due casi:

- **Routing centralizzato**: L'entità che prende la scelta di instradamento è una sola centralizzata nella rete. Occorre che conosca la topologia della rete e lo stato dei link nella rete. Se un link è più congestionato ha un costo maggiore, se è meno congestionato ha un costo minore. Questo approccio centralizzato è quello più semplice da immaginare, ma presenta delle controindicazioni:
    - Se l'entità dovesse venire meno, tutta la rete non è più in grado di riconfigurarsi
    - La quantità di informazioni che possiede quest'entità può essere troppo grande
    - Essendo separata dalla rete, potrebbe non avere un'informazione aggiornata sullo stato della rete, ma potrebbe ricevere in ritardo informazioni su di essa, e quindi reagirebbe in ritardo e configurerebbe in ritardo la rete. La rete, quindi, funzionerebbe male, perchè le conseguenze negative nella rete si propagano.
- **Routing distribuito**: In realtà gli algoritmi sono distribuiti, nel senso che la scelta dei percorsi avviene con la compartecipazione dei router, che danno informazioni sullo stato della rete e prendono delle decisioni. Quindi, il processo decisionale non ha un solo responsabile. Questo approccio distribuito si può concretizzare in due tipi di algoritmi: **distance vector** e **link state**. L'ultimo è quello usato nelle reti attuali e in internet.

A causa dell'eterogeneità della rete e del numero di link coinvolti questa complessità si traduce in una configurazione gerarchica, non paritaria, nel senso che ci sono delle scelte di instradamento all'interno della singola rete e scelte di instradamento tra reti. Le scelte di percorso vengono prese, quindi, con criteri differenti all'interno della rete.

Un **algoritmo di routing** deve prendede delle decisioni conoscendo lo stato della rete, che deve essere rappresentata e comunicata alla rete stessa. Il router decide quali informazioni condivide con la rete intorno e in base a queste si prendono le scelte di instradamento. I messaggi sono scambiati in maniera compatta per non sprecare memoria. Ci sono dei problemi di efficienza di utilizzo di operazioni computazionali. Occorre anche che gli algoritmi siano **robusti** a vari eventi negativi: devono funzionare anche in situazioni di mal comportamento o di mal funzionamento, anche se un router comunica, ad esempio, agli altri router delle informazioni errate. In virtù del tipo di approccio distribuito, i percorsi non vengono costruiti instantaneamente, perchè all'accensione di tutti i router, essi hanno solo un'informazione parziale sulla rete. Le informazioni poi si propagano nella rete, fino ad arrivare ad una condizione di convergenza: gli algoritmi convergono su scelte stabili nel tempo. Questa **convergenza** si raggiunge in un tempo piccolo in funzione delle caratteristiche dell'algoritmo e della frequenza del messaggio e si raggiunge quando non avvengono cambiamenti di stato o tipologia nella rete. Ogni volta che cambia la tipologia, si innesca un **transitorio** che deve esaurirsi fino ad arrivare di nuovo alla condizione di convergenza. L'efficienza di un algoritmo, quindi, dipende dalla durata di questo transitorio. I tempi prima di raggiungere la stabilità sono circa

una decina di secondi.

Lo scambio di informazioni avviene secondo l'approccio di un **invio periodico** di informazioni: ogni x secondi un router invia comunque nella rete agli altri router delle informazioni aggiornate sullo stato della rete, anche se non è cambiato nulla, ed ogni router conosce, con una certa affidabilità, lo stato dei link direttamente collegati. Questa informazione di stato nella rete deve essere continuamente aggiornata, e questa scelta di periodo ha una certa importanza e un certo impatto nell'efficienza della rete. Eventi di una certa importanza non vengono inviati secondo un approccio periodico, ma con un approccio **event driven**: quando si verifica un certo evento significativo, un router può decidere di notificare subito gli altri router sulle informazioni di questo cambiamento di stato. Si può usare un approccio o l'altro oppure si possono combinare. Questo perchè l'approccio periodico può dare luogo a fenomeni di lentezza di reazione ma mi limita la quantità di informazioni che viaggiano nella rete, mentre l'altro approccio può dare luogo a reazioni più rapide ma può dare anche luogo ad un traffico di controllo più oneroso, in quanto si possono generare troppi messaggi di controllo. Il secondo approccio, però, paradossalmente, può anche generare un traffico di controllo meno oneroso, perchè non manda messaggi periodici, ma li manda solo in casi critici, mentre l'approccio periodico invia traffico anche quando non succede niente.

La scelta dell'algoritmo di routing dipende in particolare dalla complessità di implementazione dell'algoritmo e della rete che vogliamo controllare.

Gli algoritmi sono distribuiti ma possono funzionare con una conoscenza globale della rete o con una conoscenza locale informata della rete. Gli algorimi globali sono quelli di tipo link state e quelli decentralizzati di tipo distance vector, ma i decentralizzati sono comunque distribuiti.

Algoritmi di routing Link State. (slide 22)

L'algoritmo routing link state parte dall'assunzione che ogni router conosce la tipologia completa della rete (sia geografica che dal punto di vista dei costi). Sulla base di questa conoscenza completa della rete, ogni router autonomamente decide i percorsi per ogni singolo instradamento, e ha una struttura dati che è un albero che ha la radice nel nodo stesso e come foglie ha tutti gli altri router della rete. Attraverso la determinazione dell'albero il router determina i percorsi di costo minimo verso una qualunque destinazione. L'acquisizione della conoscenza della rete viene fatta attraverso lo scambio di pacchetti detti **Link State Packet**, che contengono informazioni relativi ai costi dei link collegati a quel router. Questi LSP vengono propagati nella rete in ***broadcast***, cioè una trasmissione che raggiunge tutti i nodi della rete. Sulla base di queste informazioni, il router conosce la topologia e i costi della rete. Questo approccio al routing è utilizzato in alcuni protocolli di instradamento, e soprattutto nel **protocollo OSPF**.

Per il funzionamento di un protocollo link state è fondamentale il pacchetto LSP (link state packet): contengono lo **stato di ogni link connesso al router**, l'**identità di ogni vicino connesso all'altro estremo del link**, il **costo del link**, il **numero di sequenza perl'LSP** (vengono trasmessi continuante per aggiornare lo stato della rete, e il router deve capire se i due LSP sono lo stesso o generati sequenzialmente), la **checksum** (informazione di controllo che serve per capire se l'LSP è stato corrotto nel viaggio) e il **lifetime** (tempo di scadenza oltre il quale l'informazione non è più valida).

Il costo è determinato dall'amministratore della rete: maggiore è il costo che attribuiamo al link e tanto meno quel link verrà utilizzato nelle scelte di instradamento.

LSP flooding (slide 24)

L'LSP viene trasmesso ogni tot secondi, anche se non varia nulla. Vengono comunque trasmessi degli LSP di aggiornamento ogni tanto. Se ci sono aggiornamenti importanti si trasmettono subito. Questi LSP sono tramessi in flooding: una tecnica che mi consente di realizzare una trasmissione broadcast. L'obiettivo è quello di far sì che gli LSP trasmessi da un router sono trasmessi a tutti gli altri router della rete. Ad un router possono arrivare più copie che passano per router diversi. Quando arrivano i pacchetti vengono ritrasmessi in broadcast su tutti gli altri link, tranne che su quello dove il pacchetto è arrivato. Con questa tecnica arriva su tutti i router della rete, anche in più copie. Il fatto che ci sia un identificativo nell'LSP fa in modo che non vengano inviati più pacchetti inutilmente.

I pacchetti arrivati formano una mappa completa e aggiornata della rete, e quest'informazione viene mantenuta in ogni router in un **Link State Database**. Un link state database può essere rappresentato con una tabella, che mi dice quali router è in grado di raggiungere un router con annesso il costo del link. Quest'informazione è, a meno di errori, uguale per tutti, cioè se tutti propagano tutti gli LSP, non c'è motivo di avere incongruenze, e quindi tutti i router hanno la stessa visione della rete, in quanto dovrebbero avere lo stesso LSP Database. In realtà questo database può essere rappresentato mediante una **matrice di raggiungibilità**, dove sulle righe ci sono tutte le destinazioni e sulle colonne ci sono tutti i mittenti. Ogni casella rappresenta il costo del collegamento tra sorgente e destinazione. La matrice è sparsa (non è completamente riempita). È solo una tecnica di rappresentazione delle informazioni di costo.

Gestione degli LSP (slide 28)

Quando un router riceve un LSP, se non ha mai ricevuto LSP dall'altro router oppure l'LSP è più recente di quello prima memorizzato, allora memorizza il pacchetto, cioè acquisisce l'informazione ricevuta e poi la ritrasmette in flooding agli altri router a sé vicini.
In alternativa, se l'LSP ricevuto ha lo stesso numero di sequenza di un altro LSP ricevuto prima allora non fa niente, perchè si tratta di un pacchetto duplicato.
Come terza alternativa, se LSP è più vecchio di quello posseduto, il router che gli ha propagato quell'LSP deve essere aggiornato dell'informazione più recente e corretta, allora si invia una notifica con una copia più recente a quel router.

Questa conoscenza topologica non è suffuciente di per sé a determinare i percorsi, perchè contengono solo i costi dei collegamenti tra due router, ma i router non sanno come raggiungere altri router. Interviene allora **l'algoritmo di Dijkstra**, che decide, sulla base di ogni possibile destinazione, qual'è il next hop, attraverso un calcolo nel quale ogni nodo si mette come radice di un albero e raggiunge, con le sue foglie, tutti gli altri nodi della rete. Un router che lavora con un algoritmo di routing di tipo link state ha un'architettura interna particolare.

- C'è il processo di forwarding **implementato in hardware**,
- Ci sono pacchetti che si originano nel router e pacchetti che sono ricevuti,
- Quando i pacchetti sono ricevuti, ogni singolo pacchetto viene elaborato da un processo di ricezione il quale, su una base dell'informazione nella tabella di forwarding mette il pacchetto sulla coda di routing.
- C'è un processo decisionale su cui gira l'algoritmo di Dijkstra che costruisce la tabella di instradamento sulla base delle informazioni ricevute.

Vantaggi:

- Va bene per reti di grandi dimensioni.
- Ha una convergenza rapida.

- Difficilmente genera loop: bisogna sempre fare in modo che il pacchetto non faccia mai un circuito circolare all'interno della rete. È un problema molto grave, ed è un evento che può accadere perché nessun router scrive nei pacchetti il percorso completo, e se non c'è detection nessuno potrebbe rivelare la presenza di un percorso circolare. Per questo, IP si difende da questa cosa con il TTL.

Svantaggi:

- E' molto complesso da realizzare

Tabelle di instradamento (slide 32)

La tabella di instradamento contiene per ogni destinazione qual'è il link di uscita. La tabella di B mi dice che se vuole raggiungere A deve usare il link L1, e così via… Questa informazione scaturisce dall'algoritmo di Dijkstra, che ha costruito sulla rete l'albero, che è un grafo che la radice nel nodo B e le foglie in tutti gli altri router, dove ogni nodo è raggiungibile con un percorso e ogni percorso […].
In questo albero non sono utilizzati una serie di link fisici, anche se vengono collegati tutti i router presenti nella rete. Quindi l'albero è un sottoinsieme del grafo della rete.

Reti di Calcolatori - 06/11/2012

Lezione 16. Algoritmo di Dijkstra e Protocollo OSPF

Lo **Spanning Tree** è un algoritmo utilizzato per realizzare reti complesse (a Livello fisico) con percorsi ridondanti utilizzando tecnologie di Livello datalink (il livello 2 del modello OSI) come IEEE 802.2 o IEEE 802.11.Lo spanning tree viene eseguito dai bridge, e mantiene inattive alcune interfacce in modo da garantire che la rete rimanga connessa ma priva di loop.
L'algoritmo di spanning tree è un algoritmo distribuito, che opera su tutti i bridge, facendo in modo che in ogni istante la rete sia connessa ma priva di cicli, ovvero che il grafo dei collegamenti disponibili sia "coperto" da un albero. Ciò si ottiene mediante la creazione di una gerarchia di bridge. Un bridge viene individuato come radice dell'albero coprente ("root bridge"), e una parte dei collegamenti tra bridge disponibili viene messa in standby, portando in stato "BLOCKING" alcune delle porte dei bridge. Nel caso in cui un nodo diventi irraggiungibile, oppure cambi il costo di connessione, il bridge cercherà di arrivare al nodo attivando i percorsi alternativi che sono in stand-by, ripristinando in questo modo la connettività completa della rete (se possibile). Questo processo avviene periodicamente per cui, se si scollega un bridge o si interrompe un collegamento, si ricostruisce lo spanning tree e la rete continua a funzionare.
L'algoritmo tende automaticamente a mantenere in funzione i collegamenti di capacità superiore, ma talvolta la scelta di collegamenti da mantenere attivi è inadeguata alle caratteristiche della rete o del traffico che la attraversa. Configurando opportuni parametri sugli switch, è possibile influenzare sia la scelta del root bridge che la scelta dei collegamenti da mantenere in servizio.
L'algoritmo di Spanning Tree permette di estendere reti locali mantenendo un buon grado di ridondanza, ma presenta alcuni limiti:

- I tempi di convergenza, ovvero il tempo necessario al protocollo per reagire al guasto di un elemento della rete o al suo ripristino, tendono a crescere con il numero di switch coinvolti nel processo.
- Il protocollo di spanning tree genera a sua volta traffico sulla rete, che può contribuire alla sua saturazione.
- La capacità dei collegamenti lasciati in stand-by non può essere sfruttata (ovvero questi collegamenti sono usati come riserva fredda).

Algoritmo di Dijkstra.

Ogni nodo ha a disposizione il grafo della rete. Ogni nodo utilizza l'algoritmo per instradare i pacchetti. L'algoritmo è di tipo iterativo: se i nodi sono n, in n passi viene completato. Ad ogni iterazione si determina il costo minimo verso un altro nodo, quando sono fatte n iterazioni e completata l'intera rete. Indichiamo con:

- **C(i,j)**: il costo del collegamentro tra i e j. Se non c'è un collegamento il costo è infinito. Assumiamo il costo simmetrico, anche se non è strettamente necessario.
- **D(v)**: il costo attualmente calcolato del percorso che va dalla sorgente al nodo v.
- **p(v)**: predecessore nel cammino di costo minimo dalla sorgente fino a v.
- **N**: insieme dei nodi per i quali la distanza minima è stata trovata. Alla fine dell'algoritmo, questo N deve coincidere con la totalità dei nodi sulla rete.

Algoritmo. (*slide 5*)

C'è una fase di inizializzazione dove si comincia a sistemare nell'insieme n il nodo A, il nodo radice che sta calcolando l'algoritmo. Sa che il percorso per andare ad A passa per se stesso: è la radice. Possiamo dire che il nodo A è il primo nodo che andiamo a sistemare nell'albero. Sempre in questa

fase, associamo a tutti i nodi v che sono adiacenti ad A come valore D(v) il costo del collegamento diretto da A a v. Per gli altri nodi associamo a D(v) il valore infinito. Questi costi non rimarranno tali, ma ci possono essere aggiornamenti. Inizialmente, si va a prendere w non di N tale che D(w) è minimo, e si aggiunge w a N. Si aggiorna D(v) per ogni altro nodo adiacente in w e non in N. Questa distanza viene calcolata come D(v) precedente e la somma di D(w) e c(w,v). Questo significa che per i nodi raggiungibili tramite w, il costo minimo o è il costo che è stato calcolato precedentemente o è il costo del cammino più breve fino a w più il costo da w a v. Questo passo completa l'iterazione, e l'algoritmo termina quando N è completato, cioè fintantochè non sono stati collocati tutti i nodi della rete nel minimo spanning tree, al loro posto.

Interpretazione. (*slide 6*)

Consideriamo una rete di esempio fatta di sei nodi. Vogliamo trovare il minimo spanning tree che collega A con tutti gli altri nodi della rete. Eseguiamo allora l'algoritmo mettendoci nei panni di A.

Step 0.
Al passo di inizializzazione, nell'insieme N si mette solo il nodo A, e tutti i nodi che non sono direttamente collegati ad A hanno distanza D(w) da A infinita, mentre i nodi direttamente collegati ad A, nel passo di inizializzazione definiamo come costo il costo del collegamento diretto. Il predecessore del percorso minimo, al momento, è A stesso. (tabella *slide 7*)

Step 1.
Entriamo nel ciclo. La prima cosa è scegliere uno dei nodi che non stanno nello spanning tree e soprattutto quello che ha il w minimo. Tra i 5 nodi, il nodo che ha il D(w) minimo è D, perchè per questo abbiamo determinato valore 1. Allora prendiamo questo nodo D e lo aggiungiamo allo spanning tree. Scegliamo D perchè, siccome tutti gli altri link che escono da A hanno costo maggiore e la metrica è additiva, allora qualunque altro percorso nella rete che mi può portare da A verso D avrà sicuramente costo maggiore di 1, in quanto gli altri link hanno costo almeno o 2 o 5. Nessun'altro percorso, nella rete, mi può portare da A a D con costo minore di 1. Ho trovato quindi il percorso di costo minimo che mi collega da A a D. Per questo collego D sotto A con collegamento diretto. Avendo scelto di collocare nello spanning tree il nodo D ,devo ricalcolare le distanze per i nodi raggiungibili da D: B,C ed E. Devo dire che la distanza di B è il minimo tra la distanza calcolata precedentemente (2) e il percorso che passa per D: devo considerare il costo tra A e D e il costo da D a B. Questo percorso costa 3, e costa di più del collegamento diretto, quindi per B non vado a modificare la distanza. Per quanto riguarda C, ho calcolato precedentemente che posso arrivarci con costo 5. Considerando anche D, la distanza di D da C è 3, e quindi il percorso da A a C passando per D costa 4, che è di meno, quindi devo modificare l'etichetta per C. Cambia quindi il predecessore: il costo minimo per arrivare a C non è quello diretto ma è quello che passa per C. L'etichetta di E la devo cambiare, perchè prima non ci potevo arrivare, ma ora, passando per D, posso arrivare ad E con costo 2. Per F non faccio nulla, perchè ad F non ci so ancora arrivare. Così termina la prima iterazione dell'algoritmo.

Step 2.
A questo punto, D non lo porto avanti perchè è stato collocato a costo minimo nello spanning tree. Scelgo quindi il nodo che ha il costo del percorso minimo da D. Abbiamo i percorsi di costo 2 che vanno da B ad E. La scelta cade per il nodo E, che ha costo 2. Questo nodo, nel suo percorso, ha come predecessore D. Alla seconda iterazione, lo spanning tree diventa A ⎵ D ⎵ E. Avendo fatto questa scelta, devo ricalcolare i costi dei percorsi per i nodi che non sono già stati inseriti in N e che sono collegati direttamente ad E: C ed F. Per questi due nodi, devo ricalcolare le distanze. Io so che posso arrivare a C con costo 4 e il percorso di costo minimo è quello che passa per D. Adesso, invece, devo ricalcolare la distanza a C, con il minimo di 4 precedente e la somma del percorso fino

ad E e il percorso del collegamento diretto tra E e C, che sarà 3. Cambio l'etichetta al nodo C, perché ora ha costo 3, passando per il predecessore E. Devo poi ricalcolare l'etichetta per F, che prima aveva costo infinito, e ora che ho E ha come costo 4. Termina la seconda iterazione.

Step 3.
Comincia la terza iterazione. Scelgo il nodo che ha costo minimo: il nodo di costo minimo è B, che ha costo di percorso 2, e quindi aggiungo il nodo B allo spanning tree. B si va a collocare sotto ad A col collegamento diretto da A a B. Devo ricalcolare le distanze dei nodi collegati direttamente a B che non stanno in N: C. La distanza di C è il minimo tra 3, passando per E. Se passassi per B ci arriverei attraverso un percorso che ha un costo di 2+3. Siccome 3<5, non modifico l'etichetta della distanza di C, in quanto min(3,5)=3. Il fatto di aver scelto B non migliora la scelta di percorso per arrivare a C. Dopo quest'iterazione, le etichette non cambiano.

Step 4.
Nella quarta iterazione devo prendere il nodo C, che ha costo di percorso 3. Aggiungo il nodo C allo spanning tree, e questo nodo C lo colloco sotto E: so che posso arrivare a C avendo come predecessore nel percorso il nodo E. Devo ricalcolare i costi dei nodi che sono collegati a C e che non sono ancora stati inseriti nello spanning tree: F. Devo considerare se è necessario cambiare la distanza di F. Per F già so che posso arrivarci con costo 3. Se volessi passarci per C, ho un costo di 3+5=8. Non ho un costo minore di prima e quindi l'etichetta di F non cambia.
Step 5.
Nell'ultima iterazione devo inserire nello spanning tree l'ultimo nodo F.

L'algoritmo, quindi, ci ha dato tutti i percorsi con costo minimo tra tutti i nodi. E' come se avessi “potato” la rete. È come se gli altri link non esistessero: A non li usa sicuramente. Posso ottenere il costo della congestione collegando il costo del link alla congestione del link stesso. In una rete complessa non è sempre facile prevedere l'effetto di una variazione di costo su tutta la rete.

Applet su internet: *Dijkstra's Shortest Path Algoritm.*

In conseguenza dell’esecuzione dell'algoritmo, un nodo costruisce le sue tabelle di instradamento. Il nodo A sa che può arrivare ad altri nodi attraverso i collegamenti a costo minore. Ha segnato nella tabella il link al router successivo che fa in modo che l'instradameto sia a costo minore.

(*slide 10*)

La funzionalità di scegliere il costo rispetto al traffico è utile, perché se il costo di un link è proporzionale al traffico su quel link, allora possono essere generati problemi di oscillazione. Immaginiamo di avere una rete con 4 router A,B,C,D. C'è una certa quantità di traffico di peso 1 che entra nella rete da D. Supponiamo che questo traffico debba andare verso A. Un'altra quantità di traffico unitaria entra da B. Poi c'è una piccola quantità di traffico 'e' che entra da C. L'algoritmo assegnerà dei costi. Dovendo arrivare da A da D, il traffico che entra da D a A va attraverso il link diretto, con costo 1, mentre il link da B ad A sarà attraversato dal traffico che viene da B (1) + il traffico che entra da C (e).
Con questa situazione, quando si ricalcola l'algoritmo periodicamente, sarà opportuno cambiare le decisioni di instradamento, perchè il router B che deve buttare il traffico verso A sta scegliendo un percorso 1+e, ma vede che c'è un percorso che ha un costo minore, che passa per C. Allora B cambia la decisione di instradamento e butta la sua aliquota di traffico unitario verso C, invece che sul percorso diretto. Anche C cambia la sua decisione e si accorge che può buttare il traffico verso D. Sia il traffico di C che quello di B vanno verso D, e si sommano, e il link da D ad A avrà un'aliquota di traffico 2+e. Nel momento in cui si dovesse rifare il calcolo, cambiano le scelte di

instradamento, perchè se il traffico da D andasse verso C per arrivare ad A avrebbe costo 0. Quindi la situazione si ribalta perchè D ributta il traffico verso C, che tende a preferire il percorso verso destra, e pure B fa la stessa scelta. Quindi, se i router si sincronizzano contemporaneamente e periodicamente, si ha un rimbalzo di instradamenti tra i vari router, e queste oscillazioni di traffico avvengono periodicamente con il periodo con il quale i router scelgono di ricalcolare il routing.
Ho traffico che si sposta alternativamente da una parte e dall'altra, mentre sarebbe preferibile se questo traffico si distribuisse uniformemente. Facendo questo tipo di scelta, cioè attribuendo il costo dei link al volume di traffico, questo problema si esaspera quando i router cambiano periodicamente le scelte di instradamento e si sincronizzano tra di loro. È stato dimostrato che questo fenomeno di sincronizzazione tende a generarsi nei protocolli link state, con il calcolo dell'algoritmo, a meno che non intervengano altri meccanismi. Non è mai una buona scelta quella di innestare le scelte di ricalcolo delle rotte con un timer periodico, ma bisogna anche fare in modo che **un evento possa in maniera asincrona determinare il ricalcolo delle rotte**. I router non si accorgono mai di questa cosa, ma cambiano le scelte periodicamente. Se un router si accorgesse della congestione potrebbe far partire un evento che causerebbe un ricalcolo.

Protocollo OSPF (*Open Shortest Path First*)

Gli algoritmi di routing sono insiemi di regole che governano il calcolo delle rotte, sono quindi degli algoritmi eseguiti nei router per determinare le rotte di instradamento. Si basano su una conoscenza dello stato della rete grazie ad una conoscenza in broadcast. Le conoscenze si acquisiscono perché vengono mandate informazioni dai router (nel caso degli algoritmi link state abbiamo i **link state packet**). Gli algoritmi vengono eseguiti dai protocolli di router che creano le tabelle di forewarding.
Da un punto di vista implementativo, gli algoritmi di routing vengono eseguiti come dei **processi** in esecuzione sulla CPU che si trova sempre onboard di un qualsiasi router (che ha sempre una logica di controllo che fa girare un SO proprietario e una parte di hardware dedicato). In realtà, esistono vari modi di fare router, e si parla anche di **software router**, che sono quei dispositivi nei quali anche l'azione di inoltro dei pacchetti avviene attraverso il coinvolgimento della CPU del router, e quindi via software, invece che via hardware. In questo caso, la CPU è un calcolatore general purpose che non ha hardware dedicato e anche per l'inoltro dei pacchetti agisce come un normale calcolatore. Una qualunque macchina UNIX può essere configurata per agire da router se ha due interfacce di rete, cioè quando le arriva un pacchetto che ha come destinazione non uno degli indirizzi IP suoi ma un IP diverso non lo scarta ma lo instrada su un'interfaccia esterna. Una macchina UNIX che non è configurata per funzionare da router, quando si vede arrivare un pacchetto non indirizzato a lei lo scarta. Se invece è configurata per funzionare da router lo copia verso l'interfaccia di uscita secondo la tabella di routing. Quest'azione di copia e inoltro viene fatta a livello IP, e quindi nel kernel del sistema operativo. Se stiamo usando una macchina UNIX come router, la funzione di inoltro viene implementata nel kernel ed è governata dalla tabella di routing.
Chi scrive la tabella di routing? Nella macchina linux, esiste il comando **`netstat`**, che, con l'opzione **`-nr`**, mostra il contenuto della tabella di routing del nodo(anche **`route -n`**, dove -n vuol dire che non faccio la risoluzione DNS e ho direttamente indirizzi numerici).
Tipicamente per una macchina end-system, se la macchina ha una sola interfaccia eth0, configurata, per esempio, con un certo indirizzo IP 192.168.1.21/24 ed è collegata con uno switch ad un'intera LAN ethernet, e su questa LAN c'è collegato il router che ha l'indirizzo 192.168.1.1, che farà anche la funzione di natting, la tabella di routing dell'host 198.168.1.21, conterrà due entry: una entry che mi dice che è possibile raggiungere tutti gli indirizzi del tipo 192.168.1.0/24 direttamente attraverso l'interfaccia eth0, mentre l'altra entry mi dice che qualunque altro indirizzo, designato come 0.0.0.0 (rappresenta l'indirizzo base e quindi tutta la rete), è raggiungibile attraverso il router che ha

interfaccia 192.168.1.1/24 con eth0. Si può fare anche con il comando **route -n**. La tabella di routing di un router è di solito più complicata (tranne questa, che comunque passa i pacchetti in uscita al suo next stop router, di un altro provider).

Queste tabelle di routing possono essere scritte o **manualmente** o possono essere prodotte **dopo l'esecuzione di un processo** che implementa un algoritmo di routing e il corrispondente protocollo di routing. Se vogliamo usare una macchina linux come router, dobbiamo anche preoccuparci di come vengono scritte le tabelle di routing. Nei router che si trovano nel centro della rete, le tabelle di instradamento non vengono scritte manualmente, ma con un algoritmo link-state, eseguito da un processo in background sulla macchina, che implementa quell'algoritmo e implementa il corrispondente protocollo. Lo scopo del protocollo è quello di standardizzare la modalità di interazione tra queste entità.

Il protocollo è un **protocollo standard IETF**, diffuso in due versioni. Le specifiche del protocollo sono aperte e disponibili. Lo scopo del protocollo è quello di consentire lo scambio delle informazioni contenute nei Link State Packet, più altre funzionalità, non sempre tutte utilizzate.

Una delle caratteristiche fondamentali è che **OSPF** prevede anche una strutturazione gerarchica della rete. Dovendolo utilizzare per reti DSP di una certa complessità, OSPF prevede che i **router** siano **distribuiti in aree**, indipendenti l'una dalle altre, collegate attraverso un'**area di backbone** (area 0). Ci sono **router interni**, quando tutte le interfacce appartengono alla stessa area, **router di bordo**, che posseggono interfacce in 2 o più aree distinte, e **router di backbone**, che posseggono almeno un'interfaccia sull'area 0. Poi, si parla di router attraverso i quali si realizza l'interconnessione tra provider e resto di internet (**boudary router**).

OSPF è un **protocollo di router intradominio** (la rete internet è organizzata in tanti domini indipendenti, dove ognuno ha un unico amministratore, ogni dominio si collega con internet attraverso dei router e la decisione delle rotte è compito di un protocollo di routing interdominio), che fa delle scelte assumendo come entità di scelta l'autonomous system. Il router di frontiera (boundary router) deve partecipare alle decisione di routing intradominio (giù) e alle decisioni di routing interdominio (su). Lo scopo della gerarchia a due livelli è quello di gestire la complessità: il router di area 3, quando esegue l'algoritmo, non deve conoscere l'algoritmo della rete di area 1, ma deve sapere solo che nell'area A ci stanno i pacchetti A, B e C e come ci può arrivare. Come il percorso si articola nell'area 1 non serve ai router dell'area 3. E' un meccanismo che consente di ridurre la complessità, perchè lavoriamo su una quantità di informazioni più piccola. Il compito di mostrare l'area come un solo router è proprio compito dei router di frontiera.
Da un punto di vista dei messaggi, i **messaggi OSPF** non sono vincolati attraverso un protocollo di trasporto, ma **sono incapsulati in datagrammi con un numero di protocollo 89**. Il fatto che l'entità OSPF scambia messaggi non avvalendosi del servizio del protocollo i trasporto, fa sì che le entità di sicurezza del trasporto devono essere gestite dal protocollo OSPF. Il protocollo prevede dei meccanismi di sicurezza per far sì che tutti i messaggi inviati sono autenticati: quando un router riceve un Link State Packet che sembra provenire da un altro router, può essere sicuro che quel Link State Packet è stato veramente inviato da quel router, e non da qualcuno che sta fingendo di essere quel router.
OSPF prevede inoltre la **possibilità di gestire cammini multipli con lo stesso costo**, cosa utile quando l'amministratore di rete vuole fare bilanciare il carico tra percorsi alternativi.
OSPF può essere esteso per consentire l'instradamento relativamente al traffico multicast.
I router di una rete OSPF si inviano dei messaggi che servono a indicare ai vicini che sono ancora vivi e questi **messaggi** vengono detti **di HELLO**.
Ci sono poi gli Link State Packet che contengono informazioni sulla tipologia della rete. *[Il formato*

dei messaggi non lo diciamo in dettaglio]

Distance Vector.

La differenza è relativa a come vengono propagate le informazioni tra i router e che tipo di informazioni vengono propagate. Sul modo c'è da dire che mentre nel routing links state ogni router invia in broadcast qualche informazione, nel routing distance vector, il router parla cone i router che gli sono direttamente collegati, quindi non ci sono più messaggi in broadcast. Cambia il modo con il quale i messaggi viaggiano nella rete.
Nel routing distance vector, di fatto le scelte di instradamento non vengono fatte sulla base di una conoscenza precisa della topologia della rete, e questo ha delle conseguenze negative. Non si applica più Dijkstra, l'algoritmo non è più su grafo.

Ogni router inizialmente mantiene una tabella di tutti gli instradamenti a lui noti, cioè di tutte le reti che è in grado di raggiungere, e quest'informazione è relativa alle reti a cui i router sono collegati direttamente (rete vista come insieme di reti, o come insieme di indirizzi). Alle informazioni di raggiungibilità sono associati dei costi. I costi possono essere unitari, e queste informazioni sono organizzate in una tabella, che contiene la rete raggiungibile, il next hop e il numero di hop necessari per raggiungere la destinazione,. Queste informazioni vengono comunicate periodicamente a tutti i vicini. Quando un router riceve questo messaggio, aggiorna le proprie tabelle, a volte cambiando le proprie scelte di indirizzamente, e sulla base di quest'aggiornamento propaga l'informazione ai suoi vicini. Comunica un messaggio di aggiornamento che contiene tutte le reti che è in grado di raggiungere con le relative distanze (vettore delle distanze) ai suoi vicini.

[Vedi esempio distance vector sul libro o sulle slide]

Reti di Calcolatori - 08/11/2012

Lezione 17. Le socket di Berkeley (2ª parte)

Le socket rappresentano un'astrazione per un canale di comunicazione, e servono per trasmettere messaggi alla rete o su un processo che gira sulla stessa macchina. stessa macchina. Funziona grazie alla libreria che ci dà le funzionalità per accedere alle socket. Non sono altro che **API**, sviluppate prima su UNIX e poi sugli altri sistemi opertivi, fino a diventare uno standard di fatto. Il **paradigma** più utilizzato è quello **Client/Server**. Il server può essere concorrente e iterativo, dove il primo può soddisfare più richieste contemporaneamente, mentre il secondo ne può soddisfare una sola per volta.

Le socket sono di due tipi principali:

- quella **orientata alla connessione**: trasporto affidabile, in ordine, con garanzia di consegna intatta del messaggio
- quella **orientata a datagram**: invio di pacchetti con meno garanzie, dove ogni pacchetto può andare perso, possono arrivare non in ordine, ecc...

System call `socket()`. (*slide 1.34*)

La system call socket manifesta l'intenzione di usare le socket. Ha 3 parametri:

```
int socket (int family, int type, int protocol);
```

Viene restituito il **socket descriptor**, che è un tagliando usato ogni volta che chiamiamo una funzione, si porta fino alla fine del programma. All'atto di creazione della socket specifichiamo anche quale tipo di socket vogliamo (orientata alla connessione, orientata a datagram). Le socket che comunicano con rete si indicano con **`AF_INET`**, mentre le socket che comunicano nella stessa macchina si indicano con **`AF_UNIX`.** Il secondo parametro va nello specifico: come vogliamo comunicare? Con questo parametro **type** specifichiamo se vogliamo una conessione affidabile (**`SOCK_STREAM`**) o orientata a pacchetti (**`SOCK_DATAGRAM`**). Poi ci sono degli step intermedi tra connessione veloce e affidabile che possiamo scegliere: **`SOCK_RAW`**, **`SOCK_SEQPACKET`**, **`SOCK_RDM`**. Nel caso in cui utilizziamo i due protocolli principali, il campo **protocol** va inizializzato a 0, mentre nei casi intermedi può servire.
L'obiettivo delle varie chiamate di funzione è di andare a valorizzare la quintupla, che è in grado di definire univocamente una connessione. Con questa chiamata abbiamo solo specificato il protocollo.

System call `bind()`. (*slide 1.35*)

La system call bind valorizza l'**indirizzo locale** e il **processo locale**. Questa chiamata viene utilizzata da un server, perchè vogliamo assegnare l'indirizzo locale e il processo locale alla socket per essere riconoscibile all'esterno. Quindi, con la bind, associo un nome alla socket, e il nome corrisponde all'indirizzo locale e al processo locale. Nel caso delle socket UNIX, che lavorano sulla stessa macchina, non specifichiamo l'indirizzo perchè non c'è bisogno ma specifichiamo solo un nome. Associa, quindi, un nome alla socket.

```
int bind (int sockfd, struct sockaddr *myaddr, int addrlen);
```

Viene utilizzata questa chiamata in una serie di casi, come quando lancio un processo su una

macchina, creo una socket, ma non posso ricevere una connessione fino a quando non ho associato un indirizzo a quella socket. Una olta fatta la `bind()` e aver specificato il porto, una macchina si connette e il server accetta la connessione. È come se dicessimo “questo è il mio indirizzo e tutto quello che arriva alla macchina con quell'indirizzo e quel porto deve arrivare a me!”. Tipicamente è il server a fare il bind, ma lo può fare anche un client che specifica il suo indirizzo. Per il client, però, non è necessario, perchè deve specificare soltanto l'indirizzo del server, non è obbligatorio specificare il suo indirizzo e il suo numero di porto, perchè gli viene dato dal sistema operativo, ma se ne voglio uno particolare faccio la bind: se è disponibile il SO me lo dà, altrimenti dà errore.

System call `connect()`. (*slide 1.37*)

Il valore restituito ci dice se l'operazione ha avuto successo o la chiamata è fallita. Deve sempre essere controllato il valore di ritorno.

```
int connect (int sockfd, struct sockaddr *servaddr, int addrlen);
```

Poichè viene chiamata da un client, vado a definire l'indirizzo esterno del server e il processo esterno. L'indirizzo locale e il processo locale vengono stabiliti dal sistema operativo se non me li assegno manualmente con la `bind()`. Dal punto di vista del client, alla fine della `connect()` la socket è univocamente determinata, e i valori caratteristici sono quei cinque, che sono ben definiti. Alla funzione dobbiamo dare l'indirizzo del server e la lunghezza del campo sockaddress. Chiamiamo una connect se siamo un client, però quello che viene fatto dal sistema operativo dipende dal tipo di socket. In realtà, se la socket è orientata alla connessione, il sistema operativo stablisce una connessione TCP e quindi, per stabilire una connessione, invia un pacchetto e si aspetta una risposta per poter iniziare. Viene fatto quello che si chiama three way handshake. Prima di fare la `connect()` non sono obbligato a fare la `bind()` perchè se non ho un indirizzo mi viene assegnato dal sistema operativo. In un client senza connessione, se facciamo una connessione sul client UDP, i pacchetti non vengono inviati perché, dato che non c'è connessione, non c'è neanche bisogno di instaurarla. In effetti, la connect serve solo a memorizzare l'indirizzo del server. Nel primo caso è fatto il three way handshake per definire indirizzo e processo remoto, nell'altro caso lo si memorizza ma non viene fatto three way handshake.

System call `listen()`. (*slide 1.39*)

La chiamata `listen()` viene fatta dal server, e manifesta la volontà di ricevere connessione, e viene fatta dopo di `socket()`,`bind()` e prima di `accept()`.

```
int listen (int sockfd, int qlen);
```

C'è sempre il socket descriptor e poi associo la lunghezza della riga. Sono il server, mi metto in ascolto, e dico crea una coda di richieste di client, e la coda deve essere lunga qlen. Manifesta la volontà di ricevere connessione. Il valore restituito va sempre controllato per vedere se l'operazione ha avuto successo o meno.

System call accept(). (*slide 1.40*)

```
int accept (int sockfd, struct socckaddr *peer, int *addrlen);
```

Mette realmente in attesa di connessione, quindi voglio accettare la connessione. Con la

`listen()` dico che mi voglio apprestare a riceverle e la lunghezza della coda è quella. Quando faccio la chiamata di sistema `accept()`, significa che voglio accettare la connessione. Ciò significa che se ci sta un client in attesa, il sistema operativo mi metterà in attesa con quel client. Quindi, ci stanno dei parametri di ingresso e uscita con relativa lunghezza perché quando accetto le informazioni di un client devo sapere il suo indirizzo e la sua porta sorgente. L'`accept()`, quindi, stabilisce le connessioni con i client. Quindi rileva la richiesta e crea una nuova socket, che ha le stesse caratteristiche della socket inizale, se ci sono client in attesa.
Se la chiamata di funzione è andata a buon fine restituisce fino a tre valori: se è andata a buon fine restituisce il descrittore di socket, l'indirizzo del client e la lunghezza dell'indirizzo. Se non è andata a buon fine restituisce l'errore. Questa **chiamata** è **bloccante**: se non ci sono client in attesa, mi blocco là. Di default vale per tutte le chiamate.

Il server usa due socket diverse per ogni connessione con un client: la socket di ascolto, creata dalla conenssione socket, e il socket connesso, creato dalla funzione `accept()`.

Client/Server.
In questo caso, la chiamata la fa il server, non il client, perchè il server accetta le connessioni dall'esterno. Il server fa:

`int ls=socket(…);`	*Chiama la funzione socket, che restituisce il socket descriptor, e indica quale tipo di socket vuole.*
`bind(ls,...);`	*Associa l'indirizzo alle socket.*
`listen(ls,qlen);`	*Si mette in ascolto.*
`int cs=accept(ls,client-info);`	*Viene creata una seconda socket, che serve per la connessione. Quella di prima serviva per l'ascolto. La prima continua ad essere di ascolto di altre connessioni mentre questa socket mi serve per la connessione con un determinato client.*

(Esempio di codice ***slide 43***)

System call `send()` e `sendto()`.
Sono chiamate di funzioni bloccanti.

System call `recv()` e `recvfrom()`.
Sono chiamate di funzioni bloccanti.
Se la `recv()` non riesce a scrivere niente si blocca.

System call `close()`. (*slide 1.48*)

```
int close(int fd);
```

Chiude una socket e quindi permette al sistema operativo di liberare le risorse allocate. In caso di connessione TCP viene fatta una four way handshake (a 4 vie).

Marshalling dei parametri. (*slide 1.49*)

Le socket nascono in un ambiente eterogeneo: non so con chi sto comunicando. Per comunicare con chiunque devo usare gli opportuni accorgimenti. Con una operazione della marshalling codifico le informazioni in un formato neutro che il ricevitore può decodificare. Le funzioni sono dette **byte ordering routines**: stanno attenti all'ordine dei byte a livello degli interi. Le architetture sono diverse, posso avere gli interi scritti in maniera diversa.

Operazioni sui buffer. (*slide 1.51*)

Conversione di indirizzi. (*slide 1.53*)

Reti di Calcolatori - 09/11/2012

Lezione 18. Routing Distance Vector

Il concetto fondamentale è il calcolo, da parte dei nodi della rete, dell'**equazione di Bellman-Ford**: un nodo apprende dai suoi vicini le distanze da tutte le altre reti note ai vicini e usa queste distanze per aggiornare le proprie distanze da questa rete sulla base della conoscenza del costo del collegamento che collega il nodo ai vicini stessi. **(*slide 6*)** Immaginiamo che il nodo X è collegato ad A,B,C e riceve da loro le distanze che questi vicini A,B,C conoscono per raggiungere una rete Y che non è direttamente collegata a X. A,B e C annunciano X che sanno raggiungere Y con una certa distanza. X deduce come costo minimo per raggiungere Y il minimo della somma del costo del collegamento diretto tra X e V, dove V è uno dei suoi vicini, e la distanza che il vicino V annuncia con Y. Prende il minimo tra tutte queste quantità. Determina anche attraverso quale next-hop può raggiungere Y, ed è il nodo a cui X decide di passare per inviare pacchetti a Y.
Per poter calcolare queste distanze, ogni nodo invia ai nodi adiacenti un **distance vector**, costituito da un insieme di coppie indirizzo-distanza, dove la distanza è espressa tramite metrica. Quando un nodo riceve un distance vector, lo memorizza, e aggiorna le informazioni mantenute precedentemente. Calcola le tabelle di instradamento e se le per effetto della ricezione del distance vector il nodo decide di cambiare le proprie decisioni di instradamento e invia ai nodi adiacenti un nuovo distance vector.
Gli eventi che determinano il ricalcolo delle tabelle sono:

- variazioni della topologia,
- cade una linea attiva,
- cambia il costo di una linea,
- ricezione di un distance vector aggiornato diverso da quello utilizzato.

Questo tipo di algoritmo che procede in maniera distribuita sulla rete è semplice da implementare. È più semplice da implementare dell'algoritmo di Dijkstra. Ci sono però alcuni difetti: per come funziona questo tipo di algoritmo nessun nodo ha una mappa completa della rete, l'informazione che possiede ogni nodo è solo **un'informazione di raggiungibilità**, ed è legata alla topologia ma non è la topologia esatta della rete, che non sarà nota a nessuno.
Per ogni rete che conosce, il router sa qual è il next-hop router e conosce anche la distanza. Ad esempio, il router B può raggiungere 1 e 2 direttamente, può raggiungere la rete 4 attraverso la rete L,ecc... Ad un certo momento il router B raggiunge un messaggio di aggiornamento dal router A. Questo messaggio contiene la lista di informazoni di raggiungibilità. La ricezione di questo messaggio costringe B a fare degli aggiornamenti, che consistono nell'aggiunta di una nuova entry, perchè attraverso questo messaggio il router B conosce la raggiungibilità di una rete che prima non conosceva.

Esempio. **(*slide 13*)**

Un router mantiene una tabella di instradamento attraverso la quale sintetizza le informazioni di raggiungibilità che sintetizzano le informazioni vicine.
Immaginiamo di avere una rete di tre soli nodi X,Y e Z, con dati costi di collegamento. Dobbiamo vedere inizialmente per ogni nodo qual è l'informazione mantenuta e come quest'informazione si aggiorna in ogni nodo. Inizialmente, il nodo X sa solo ciò che gli deriva dalla conoscenza dei link che escono direttamente da lui. Al tempo 0 non ha ancora ottenuto informazioni di raggiungibilità da Y e da Z. Lo stesso accade nel nodo Y e nel nodo Z. Una volta stabilita questa conoscenza iniziale, le informazioni vengono propagate: il nodo X trasmette ai vicini la sua informazione di raggiungibilità: trasmette a Y e Z quello che lui sa della raggiungibilità della rete. Quando i vettori arrivano al nodo X, queste informazioni vengono messe nella tabella. Sulla base di queste

informazioni, X applica le equazioni di Bellman-Ford, e nel caso cambia le tabelle di instradamento. La stessa cosa vale per Y e Z. Con l'algoritmo di Bellman-Ford, quindi, si calcola la minima distanza tra il collegamento diretto e quello attraverso gli altri nodi.
In questa situazione, l'algoritmo converge: tutti i 3 nodi alla fine convergono sulla stessa informazione.

Algoritmo. (*slide 11*)

E' un algoritmo in cui gli scambi di informazione sono causati o da un cambiamento di costo di un collegamento, e il cambiamento di costo al caso limite può essere visto come il link che va giù, e quindi il costo diventa infinito. In alternativa, un ricalcolo è determinato dalla ricezione di un messaggio da un vicino: quando un vicino mi aggiorna, mi costringe a ricalcolare la tabella delle distanze. L'algoritmo procede in maniera distribuita: ogni nodo va per fatti suoi e l'algoritmo in condizioni normali **tende a convergere**: le decisioni di instradamento sono coerenti tra di loro (non si creano dei loop). L'algoritmo è **asincrono**: la tempificazione dei messaggi è data da eventi non determinati da un clock, ma è determinata dagli eventi stessi (accensione di un nodo o variazione di costo di un link). C'è una prima sequenza di inizializzazione, in cui il nodo X, per tutti i nodi adiacenti V mette nella matrice nella posizione che collega i nodi adiacenti come minima distanza il costo del collegamento diretto. In tutte le altre posizioni, mette infinito. Dopo aver fatto questo, per tutte le destinazioni Y, manda il minimo della distanza di raggiungibilità per quella destinazione Y ad ogni vicino. Entra poi in un ciclo, in cui aspetta che si verifichi un evento che lo mette in attività, che può essere la ricezione di un messaggio dal vicino oppure l'aggiornamento del costo di un collegamento. Se l'evento che ha fatto uscire da quell'attesa è stata la variazione di costo di un collegamento, allora la variazione di costo cambia il costo verso tutte le destinazioni che sono raggiungibili attraverso quel vicino V. Allora, per tutte le destinazioni raggiunte da V cambia la distanza di raggiungibilità. Se invece l'evento è stato un messaggio di aggiornamento V verso una destinazione Y, vuol dire che il cammino minimo da V a Y è cambiato. Questo nuovo valore costringe il nodo X a cambiare la sua stima della minima distanza con cui si riesce ad arrivare ad Y attraverso il nodo V. Se è cambiato qualcosa, il router X manda ai vicini l'aggiornamento che è stato fatto sulle tabelle di routing, affinchè essi aggiornino le loro tabelle di instradamento.

Vantaggi:

- facilità di implementazione

Svantaggi:

- In certe circostanze, le decisioni di instradamento non convergono velocemente alla configurazione finale. Quando si adotta questo modo di instradamento, le buone notizie viaggiano velocemente e le cattive notizie viaggiano lentamente. (***Good news travels fast, bad news travels slow***)

Esempio di buona notizia: il **costo** di un collegamento **diminuisce**. Al tempo t_0, il router Y si accorge di questo cambiamento e aggiorna la sua matrice di stima delle distanze e informa i suoi vicini, che sono X e Z. Nel ricevere quest'informazione, Z calcola un nuovo costo minimo per raggiungere X attraverso Y. Questa variazione della stima del costo con cui Z raggiunge X, che da 5 passa a 2, viene trasmessa al tempo t_2 da Z a Y. Questa cosa non produce nessun effetto al nodo Y. Dopo questo scambio di messaggi, la situazione raggiunge una nuova convergenza.
Esempio di cattiva notizia: il costo di un collegamento aumenta. Y si accorge del fatto che il collegamento diretto tra Y e X non è più 4 ma 60. La sua stima di costo minimo per arrivare a X diventa 6, perchè sapeva che Z gli aveva dichiarato che lui era in grado di raggiungere X con costo 5. Siccome Y ha ricevuto quest'informazione da Z, questa cattiva notizia è compensata dal fatto che Z sa raggiungere X con costo 5. Per effetto di questa scelta sbagliata, quando Y vede un pacchetto

con destinazione X, lo butta a Z, e non sul collegamento diretto. Per Z, succede che Y cambia la sua stima della raggiungibilità: è convinto di poter arrivare a X con costo 6, e dichiara quest'informazione a Z. Siccome Z apprende da Y che Y è in grado di raggiungere X con costo 6, nel momento in cui riceve quest'informazione aggiorna la sua stima di costo e la fa diventare 7. Ai pacchetti con destinazione X succede che si crea un routing loop: un percorso di instradamento circolare. Questo loop esiste fintantochè i due router Y e Z non si accorgono che ad un certo punto stanno sbagliando, e cioè ad un certo momento, mano a mano che le stime crescono, Z si rende conto che può arrivare ad X grazie al collegamento diretto. Il loop si rompe perchè Z non manda più i pacchetti verso X attraverso Y ma li manda direttamente. Il conteggio all'infinito si ferma, perchè si ha una nuova situazione di convergenza.

Esempio topologico. (*slide 19*)

Consideriamo una rete con i nodi messi in fila e supponiamo il costo unitario per ogni link. A informa ai suoi vicini la sua matrice di distance vector, in cui dice ai suoi vicini che può raggiungere B,C, D ed E con costi 1,2,3,4, e anche il contrario. Ad un certo momento, il nodo A si spegne. Succede che il nodo B, che prima sapeva di poter raggiungere A con un solo passo, adesso conclude che sa raggiungere A in 3 passi, perchè prende per buona l'informazione di C (che aveva detto che sapeva raggiungere A con 2 passi), e quindi non deduce che A non è più raggiungibile immediatamente, ma deduce che A è raggiungibile ora con costo 3. Questo succece perchè questo tipo di algoritmo non si basa sulla conoscenza topologica, ma solo sulle informazioni di raggiungibilità. B non sa che C raggiunge A attraverso se stesso, ma immagina che C abbia un altro collegamento diretto verso A. Allora, B sa che può raggiungere A con costo 3, e invia quest'informazione a C. Quando B dice a C che sa raggiungere A con costo 3, questo determina una variazione anche in C. Che incrementa il costo per arrivare ad A, ed informa questa variazione B e D. Allora, D deve cambiare la sua stima, e anche B rivede la sua stima e adesso per raggiungere A deve pagare costo 5. Questo perchè può immaginare che in C c'è stato un evento che gli ha fatto aumentare il costo. Questo processo non converge mai, ma diverge fino all'infinito. Nella realtà non può convergere perchè A è diventato irraggiungibile dagli altri nodi. B,C,D e E dovrebbero dedurre che A ha raggiunto distanza infinita. Ma fino a che B,C,D ed E se ne facciano una ragione, ci mettono un tempo infinito. Per poter risolvere questo problema, C non dovrebbe mai dire a B che sa arrivare ad A se passa proprio per B.

Split Horizon con Poisoned Reverse (*slide 20*)

La tecnica viene detta **poisoned reverse**: con invio avvelenato di raggiungibilità sul percorso inverso: se Z raggiunge X attraverso Y, allora Z dice a Y che la sua distanza ad X è infinita. Quindi gli dà un'informazione di fatto ***incorretta***. Sostanzialmente C, siccome sa che per arrivare ad A lo fa attraverso B, già da subito quando scambia i vettori delle distanze, visto che per arrivare ad A deve passare per B, dice a B che sa arrivare a A con distanza infinita (lo dice solo a B). E' un'informazione fasulla che viene data solo in una direzione, e non su tutte le direzioni. Questa tecnica garantisce una **più veloce convergenza**.
Esempio. Il collegamento X-Z ha costo 50, e il collegamento X-Y passa da 4 a 60. Se non si usa la tecnica del poisoned reverse, Z dice a Y che è in grado di raggiungere X con costo 5, e questa cosa innescava il loop. Da subito, quindi, Z, quando deve annunciare la raggiungibilità a x attraverso z, dice che lo sa raggiungere con costo infinito. Per quanto riguarda Z, sa che può arrivare a X con costo 50 […]. (vedi ***slide 20***).

(slide 26) A volte si possono innescare questi fenomeni, in altri no: dipende da quando arrivano i messaggi di aggiornamento dei router. In questo tipo di topologia si può innescare un loop tra A e B che non sparisce mai, in quanto si innesca un count di infiniti tra A e B nell'arrivare alla stima della

distanza D che di fatto è infinita.

Protocollo RIP.

È il più diffuso protocollo **IGP (*Interior Gate Protocol*: *protocollo per il routing intradominio*)**. La prima versione risale al '69, poi è stato aggiornato, un'implementazione come daemon in UNIX risale all'82 quando da router faceva una macchina UNIX. Funziona sul concetto di routing distance vector. Ha una serie di particolarità che derivano dalla semplicità di implementazione. Non fa distinzione tra reti e host singoli, quindi nelle tabelle di routing ci sono entry che possono puntare ad un singolo host o ad una intera rete. Di solito conviene mettere voci che aggregano più reti, gli host a maggior ragione non si mettono quasi mai. Prevede che a partecipare al processo di instradamento non sono solo il router veri e propri ma anche gli end system, che partecipano in maniera passiva, cioè ricevono le decisioni di instradamento dei router ma non possono decidere. Ogni 30 secondi sono inviati messaggi dalle entità attive in broadcast per la coordinazione. Ogni messaggio di raggiungibilità contiene fino a 25 reti di destinazione. Un host aggiorna una rotta solo se ne apprende una migliore. Ogni informazione ha un timeout di 180 secondi, e l'unica metrica utilizzata è il numero di hop, quindi non va bene quando i link diretti tra due router non hanno costo 1. Se non c'è il messaggio di advertisement dopo 180 secondi, il link è considerato morto, quindi i router che attraversano quel router annullano tutte le rotte che attraversano quel vicino. Da un punto di vista protocollare, il protocollo RIP si colloca sopra UDP, è applicativo, e le entità sono scambiate incapsulate in datagrammi UDP: si usa questo perchè i messaggi sono piccoli e inviati regolarmente, e non c'è bisogno di garantire l'affidabilità completa, ci vuole solo un minimo di affidabilità data dal timeout, che è necessario. *[Non bisogna sapere il formato dei messaggi!]* Un pacchetto UDP con destinazione/mittente 520 è un pacchetto RIP.
C'è un campo che specifica il **tipo di comando**, il campo **versione**, e poi c'è un **identificato di address family** che consente al RIP di non essere collegato direttamente a IP(soprattutto IPv4, infatti devo indicare il formato degli indirizzi. Per IPv4 il valore di address family è 2). Ci sono poi delle **informazioni di raggiungibilità** che possono essere inserite nei campi come indirizzo e **sua corrispondente metrica**. Il valore massimo della distanza tra due router è pari a 15, e quindi un valore maggiore è considerato infinito, il che significa che questo protocollo di routing va bene per sistemi a piccolo diametro (su internet non funzionerebbe per vari motivi, uno è questo della distanza massima, si applica per reti di piccole dimensioni). Soffre dei problemi del routing distance vector, ma comunque risolve alcuni problemi, come il fatto che il conteggio all'infinito ha il vantaggio di non arrivare all'infinito, ma di arrivare a 15. Però, è facile mettere in difficoltà la rete inducendo nella rete delle informazioni inesatte. Questo è inaccettabile quando devono parlare router soggetti a diversi ISP. Non c'è abbastanza affidabilità nella comunicazione.
Ci sono due tecniche fondamentali:

- **Split horizon**: non si inviano al router le rotte che passano per quel router
- **Split horizon with poison reverse**: si indicano come infinite ad un vicino tutte le rotte passanti per il vicino stesso

In realtà, è opportuno che i messaggi vengano anche scambiati anche in seguito a eventi che danno la necessità di aggiornare, oltre all’aggiornamento periodico.
Queste informazioni si concretizzano in una **tabella di routing**, di un host, dove ci sono dei **flag**.

Alcune volte, le rotte tra i router possono essere modificate in seguito ad un messaggio di **ICMP redirect**, se i router si accorgono che ci sono dei percorsi più veloci. Quando avviene, queste rotte così modificate sono segnalate con dei flag. La versione 2 introduce la possibilità di gestire gli indirizzi non per classi ma attraverso un subnetting identificato da una netmask (indispensabili). Quando un router dà la raggiungibilità della destinazione, affianco alla raggiungibilità della

destinazione ci aggiunge anche la netmask nel messaggio.

IGRP.

IGRP è un protocollo distance vector proprietario della CISCO, che introduce la possibilità di usare tecniche metriche di costo più sofisticate, e lo scambio di messaggi avviene attraverso TCP.

Reti di Calcolatori - 13/11/2012

Lezione 19. Tecniche di trasmissione broadcast, IP multicasting IGMP

Tecniche di trasmissione broadcast.

La trasmissione broadcast consegna i pacchetti da una sorgente a tutti gli altri nodi della rete. In realtà, IP di per sé non supporta una trasmissione broadcast su scope globale della rete, perchè sarebbe troppo pericoloso, e produrrebbe un sovraccarico eccessivo sulla rete. Esiste, in maniera nativa, la possibilità di trasmissione broadcast a livello locale, sul segmento di rete IP a cui appartiene un host che, inviando un pacchetto all'indirizzo 255.255.255.255 ottiene che il pacchetto viene ricevuto dagli host che stanno sulla stessa subnet IP (rete locale). I pacchetti broadcast con questo indirizzo non attraversano il router, ma sono broadcast locale.
Però, a volte, il paccetto deve poter viaggiare all'interno di una rete non locale ma almeno bisogna consegnare il pacchetto ai nodi all'interno di una porzione di rete. Nel contesto del router link-state i pacchetti si consegnano ai router e non agli end-system, quindi comunque è stato fatto. Se non ci fosse un supporto da parte dei router, l'approccio più semplice è quello di realizzare la moltiplicazione del pacchetto da parte della sorgente: se il pacchetto deve essere consegnato ad n destinatari, allora la trasmissione broadcast si trasforma in n trasmissioni unicast verso ciascun singolo destinatario. Questo tipo di approccio ha dei problemi: si può realizzare con la normale trasmissione unica mittente-destinatario, ma come fa la sorgente a conoscere la lista di tutti gli indirizzi? Inoltre, sul link che collega il mittente del pacchetto al primo router c'è un numero eccessivo di pacchetti, quindi si ha un traffico eccessivo nei router vicini alla sorgente. Si potrebbe risolvere inviando i pacchetti in tempi diversi, ma si incorrerebbe in altri problemi, come l'allungamento del tempo richiesto per l'invio a tutti.

Una soluzione più efficiente è quella di chiedere ai router della rete di realizzare, al posto della sorgente, la moltiplicazione dei pacchetti, ogni volta che i pacchetti devono seguire percorsi diversi. Il mittente invia una sola copia del pacchetto al router successivo, e il router produce dal pacchetto originale due repliche uguali che vengono inviate verso due router diversi, e così via. Laddove si dovessero raggiungere altri router più distanti si avrebbero ulteriori moltiplicazioni dei pacchetti. Questo tipo di tecnica è più efficiente, però richiede il supporto in trasmissione da parte dei router.

In-network duplication. (*slide 5*)

Il **problema** sta nella **decisione**: se ho un router che smezza i pacchetti tra R3 e R4, il router R3, quando si vede inviare un pacchetto, deve inviarlo a R4? Se lo fa, ad R4 arrivano due pacchetti. In realtà, i router R3 e R4 non lo sanno a priori che ad R4 il pacchetto arriva da un'altra strada. Si deve fare il **flooding**: quando un nodo riceve un pacchetto broadcast, lo invia a tutti i suoi vicini tranne a quello che gliel'ha inviato. Quindi, con questa tecnica, R3 lo deve trasmettere ad R4, e quando R4 si vede inviare il pacchetto da R3, dobbiamo stare attenti a non far generare cicli o tempeste di broadcast. Allora,, il flooding è inefficiente, in quanto l'approccio è inefficiente, perché circolano molti pacchetti inutili.
C'è quindi il **controlled flooding**, che consiste nel non ritrasmettere un pacchetto se questo pacchetto è stato già ricevuto precedentemente. Ha 2 possibili implementazioni. Il router R3 trasmette il pacchetto verso R4, che ha due copie del pacchetto. Se arriva prima la copia da R2, quando arriva la seconda copia da R3, R4 deve capire che già l'ha ricevuto,e non lo deve ritrasmettere. Si può implementare mettendo nel pacchetto un numero di sequenza. Se arriva ad un router un pacchetto con un numero di sequenza minore o uguale del numero di sequenza del pacchetto che già gli è arrivato in precedenza, non viene ritrasmesso. Si mantiene traccia del

numero di sequenza più grande ricevuto fino a quel momento.
Una tecnica più intelligente è quella dell'**RPF (Reverse Path Forwarding)**: un nodo invia il pacchetto in flooding su tutte le interfacce solo quando questo pacchetto gli arriva dal link che fa parte del percorso più breve tra quel nodo e la sorgente del pacchetto (scritta nel pacchetto). Ad esempio, se R3 riceve il pacchetto da R2 e lo trasmette ad R4, perchè il pacchetto gli è arrivato dal lato buono, che è il link che fa parte del percorso più breve che collega R3 con la sorgente R2, e quindi R3 si pone il problema: se devo trasmettere il pacchetto ad R1, utilizzerei il link verso R2 (supponiamo i percorsi simmetrici), e allora, siccome il link da cui arriva il pacchetto fa parte del percorso minimo che unisce R3 e R1, il pacchetto deve essere ritrasmesso sulle altre interfacce. Ad R4, il pacchetto che arriva da R2 arriva dal lato buono, mentre quando gli arriva il pacchetto R3, non gli arriva dal lato buono, perchè non viene dal link appartenente allo shortest path, e quindi non viene ritrasmesso il pacchetto arrivato da R3 che non partecipa ad ulteriore flooding. Eventualmente si può combinare questa tecnica con il flooding controllato.
Un'ultima tecnica più efficiente è quella dello **spanning tree**: i router della rete, insieme a R1, si organizzano in maniera da creare un albero di copertura minimo che collega R1 ad ogni altro singolo router della rete, usando l'algoritmo di Dijkstra. Quando R1 vuole inviare il pacchetto in broadcast, questo pacchetto viene fatto viaggiare lungo i lati dello spanning tree. Lo manda ad R2, da cui si diramano due rami tra R3 e R4, che sono nodi foglia, e quindi non devono propagare ulteriormente il pacchetto. In questo modo, viaggiano sulla rete il minor numero possibile di pacchetti che consentono di far arrivare i pacchetti a tutti i nodi.
I router della rete si accordano per creare un unico spanning tree, in modo tale da eleggere un router come router di radice di uno spanning tree che li copre tutti, e quando un altro nodo della rete vuole trasmettere il pacchetto sulla rete, trasmette il pacchetto al nodo che è stato eletto in unicast e poi si fa il broadcast.

Si costruisce un nodo centrale (***nodo di rendez-vouz***), radice dell'albero, nel caso di shared tree (albero condiviso), dove c'è un unico albero condiviso da tutti i nodi della rete, che ha un'unica radice, e allora per la propagazione dei messaggi c'è una trasmissione unicast dei pacchetti verso questa radice e poi la propagazione dei pacchetti lungo l'albero. In realtà, la costruzione dell'albero viene fatta così: ogni nodo manda un messaggio di adesione all'albero al nodo radice: se tutti i nodi del percorso non sono già entrati nello spanning tree lo si inoltra alla sua destinazione (anche se potrebbe essere intercettato da un nodo lungo il percorso, se qualche nodo ha giù trovato una sua collocazione nello spanning tree, in quel caso si allunga il ramo e si colloca il nodo mittente del join nello spanning tree).

Se G manda il pacchetto di join verso A, e questo pacchetto arriva a D che fa parte dello spanning tree, D capisce che vuole trovare il posto dell'albero e lo aggiunge a se stesso. Se D rifiuta il join con G, avrebbe propagato il messaggio più a monte.

Trasmissione multicast in reti IP. (*slide 8*)

Nel caso broadcast, i pacchetti devono arrivare a tutti i nodi. Nella trasmissione multicast ogni nodo decide se deve ricevere i pacchetti legati a quella trasmissione oppure no. E' indirizzata ad un gruppo di indirizzi IP la cui **membership** è **gestita dinamicamente**, di cui fanno parte un numero variabile di indirizzi IP che, di volta in volta, decidono se ricevere quel pacchetto oppure no. Le applicazioni che possono beneficiare la trasmissione da un server ad un gruppo di ricevitori sono **aggiornamenti software**, **flussi multimediali** (audio, video, presentazioni(da un mittente ad una molteplicità di ricevitori variamente collocati nella rete, **applicazioni di teleconferenza**, **flussi di informazioni** generate in tempo reale, la **trasmissione di flussi multimediali in tempo reale**, come quotazioni in borsa, flussi televisivi, etc… E' utile la possibilità che può offrire la rete di supportare questo tipo di trasmissione multicast, quindi il mittente genera un unico pacchetto che

viene inviato a tutti i destinatari.
Si hanno le stesse strategie di prima: la trasmissione multicast si può creare facendo generare al mittente una ennupla di pacchetti e inviara nella rete, ma si può generare una congestione nelle vicinanze del mittente. Lo stesso gruppo di host può essere interessato a ricevere pacchetti da più host mittenti, ma si tratta la cosa come n problemi distinti. Se invece riesco a coinvolgere i router, quando è necessario, essi generano copie molteplici dello stesso pacchetto, e sanno queste copie verso quali link devono essere trasmesse Non si generano copie inutili del pacchetto perché non vengono trasmessi a endsystem che non sono destinatari del pacchetto. **(*slide 10*)** Quando si arriva ad una rete locale, siccome tutte le tecnologie LAN supportano relativamente la trasmissione multicast, la duplicazione non avviene sul segmento di rete più a destra, ma il router trasmette una sola copia del pacchetto con un indirizzo destinazione speciale, lo mette sul bus. Il pacchetto viene sentito da tutti gli host della rete locale, e ci sono host che lo ignorano e altri host che ricevono il pacchetto in maniera simultanea. Di fatto in un segmento di rete locale il router trasmette solo una copia del pacchetto, indipendentemente dal numero degli host in ricezione.
Ci sono dei problemi:

- Come identificare i ricevitori di un diagramma multicast? E' un problema di indirizzamento.
- Come si fa a gestire questa trasmissione? Come fanno i router a sapere verso quali link trasmettere i pacchetti e quando vanno duplicati?

Address indirection. (*slide 12*)

Il problema della individuazione degli host viene gestito attraverso un'**astrazione**, c'è un indirizzo speciale che identifica tutti gli host appartenenti ad un gruppo, e si assume che i gruppi abbiano una durata temporale finita, cioè un certo insieme di host nella rete sono interessati a ricevere un messaggio in una durata temporale. L'assegnazione tra un host e un gruppo viene fatta esplicitamente, però è un'assegnazione che non è intesa per durare un tempo indefinito. In questa maniera, lo stesso indirizzo di gruppo, nell'arco del tempo, può essere utilizzato per individuare indirizzi differenti, ma occorrono dei meccanismi esterni che gestiscono l'associazione tra indirizzo di gruppo ed evento, in quanto la rete non è consapevole di questo. Quest'associazione non avviene nativamente nella rete. Un determinato gruppo di host che sono interessati a ricevere una trasmissione in multicast è determinato da un indirizzo di classe D (lasciati per individuare i gruppi di trasmissione multicast). Questi indirizzi costituiscono 1/16 dell'intero spazio di indirizzamento IP, è grande ma non illimitato, e il numero di gruppi che si riesce a gestire è 2^{28}, pari a circa 268 milioni di gruppi.
Un certo indirizzo viene associato ad un evento secondo uno standard non definito a livello IP. Cosa fa si che un certo gruppo di host che partecipa ad un evento è autorizzato ad usare un indirizzo, che non può essere usato da altri host per un altro scopo, si basa su **meccanismi esterni** per evitare le collisioni, che non sono cablati nei meccanismi standard. Tutte le volte che si vuole usare un indirizzo, in generale, serve un'**autorizzazione** (diretta o indiretta, cioè statica o via ISP).
Nel caso degli indirizzi multicast, il problema delle collisioni degli indirizzi è mitigato da tecniche che limitano lo scope dei pacchetti multicast a livello di instradamento: una volta che si utilizza un certo indirizzo si può delimitare la portata di attraversamento dei pacchetti sulla rete. Se non ci sono meccanismi di limitazione dello scope gli indirizzi vanno usati univocamente.

Il range di indirizzi di classe D sono quegli indirizzi che vanno da 224.0.0.0 a 239.255.255.255. Ci sono poi degli indirizzi riservati. In particolare:

- l'indirizzo **224.0.0.1** è utilizzato per consegnare un pacchetto a tutti i sistemi (end-system e router) della subnet.
- L'indirizzo **224.0.0.2** è utilizzato per consegnare un pacchetto a tutti i router di una certa sottorete.

- L’indirizzo **244.0.0.13** è utilizzato per consegnare un pacchetto a tutti i router che fanno parte di una certa rete chiamata PIM.

Scope: sta a significare il fatto che i pacchetti viaggiano per un certo numero limitato di hop, per cui i pacchetti non possono fare un viaggio illimitato nella rete, ma si possono allontanare dal mittente per un certo numero di classi, e quindi limito le collisioni nella rete. E' come se partizionassi la rete, evito collissioni di indirizzi. Limitando lo scope limito collisioni nello stesso gruppo.

Gruppo multicast. (*slide 15*)

Abbiamo una rete dove abbiamo tanti host univocamente individuati da un indirizzo IP unicast. L'host mittente lo manda all'indirizzo del gruppo che lo invia ai vari host che fanno parte del gruppo.

SAP *(Session Announcement Protocol)*. (*slide 16*)

Occorrono dei meccanismi che consentano di capire quali indirizzi sono utilizzati e quali no e per quali scopi sono utilizzati. Nel protocollo SAP, i messaggi viaggiano su un gruppo multicast che ha scope globale (**224.2.127.254**), e su questo gruppo viaggiano pacchetti UDP indirizzati verso la porta 9875 con TTL 225.
Attraverso questi messaggi, vengono annunciate le sessioni, cioè trasmette gli indirizzi che verranno utilizzati in determinati archi temporali, e gli altri host sanno che quegli indirizzi vengono temporaneamente utilizzati per un'altra trasmissione. Supponendo che su questo gruppo viaggi un flusso video, bisogna anche sapere come questo flusso è individuato, e in questa maniera, gli host che sentono questo messaggio di announcement SAP e vogliono sintonizzarsi su questo gruppo, sanno che su questo gruppo viaggia un flusso video e che viene trasmesso sul port number UDP 8800, per esempio, e quindi sa che deve aprire un'applicazione e si deve mettere in ascolto di pacchetti sulla porta 8800. Questo protocollo evita collisioni e dà informazioni utili per sintonizzarsi al multicast, per poter agganciarsi al gruppo, trasmettere i flussi e dare informazioni sull'applicativo usato per decodificare il flusso. I messaggi che sono veicolati da questo protocollo sono formattati secondo un formato di descrizione **SDP *(Session Description Protocol)***, che descrive i tipi di informazione che viaggiano su questo gruppo, e *protocol* è usato impropriamente, perchè non è un protocollo, ma è un formato di descrizione, è un formato di metadato testuale in cui ci sono varie linee e ogni linea è formata da un nome di lettera che identifica un tipo di record, un uguale, e a destra dell'uguale un valore che ha significato in funzione al tipo di record (es. 'a = ...). Oggi si userebbe XML

La gestione dei gruppi. (*slide 17*)

Le proprietà di IP multicast sono definite dall'RFC 1112. Fatto dopo gli RFC di TCP, UDP e definisce le estensioni di IPv4 per il multicast (perché queste tecniche di trasmissione sono state implementate dopo, in quanto sono servizi aggiuntivi formalizzati successivamente).
L'appartenenza ai gruppi è dinamica, quindi quali host fanno parte di un gruppo è una cosa non predefinita, quindi un host può entrare o uscire da un gruppo quando vuole e si può partecipare a più gruppi differenti.
Per poter inviare messaggi ad un gruppo multicast, non è necessario appartenere a quel gruppo. Il join ad un gruppo deve essere fatto obbligatoriamente per ricevere pacchetti, mentre per poter inviare pacchetti ad un gruppo non è necessario fanne parte.
Gli host di un gruppo possono appartenere alla stessa rete fisica o meno. Se gli host stanno sulla stessa LAN, la trasmissione avviene secondo un supporto fisico e quindi, usando indirizzi speciali, posso far sì che il pacchetto venga ricevuto da più host. Quando invece devo viaggiare tra reti devo

necessariamente usare router multicast. Un router multicast copia dei pacchetti relativi ad un'interfaccia verso altre interfacce differenti al difuori della rete del mittente. Ci sono analogie e differenze col flooding: bisogna fare in modo che i pacchetti arrivino agli host interessati e non bisogna allargarsi a tutta la rete. Nel momento in cui un pacchetto multicast parte, viene inviato sulla rete locale. Se ho degli altri host che sono nella stessa rete locale del mittente, allora il pacchetto che il mittente invia sulla rete locale viene subito ricevuto anche dall'altro host che sta sulla stessa LAN, che lo prende perchè è indirizzato ad un gruppo che è interessato a ricevere. Il pacchetto lo vede e lo riceve anche il router multicast, che sa che i pacchetti indirizzati al gruppo non solo devono rimanere nella LAN ma devono anche andare nel resto della rete, e quindi prende il pacchetto e lo copia dall'altra parte della rete. Gli host destinatari fanno l'ultima operazione di trasmissione. Tutti i router nella rete inviano i pacchetti verso i destinatari e l'ultimo invia una sola copia nel pacchetto. Tutti i router che partecipano devono essere multicast. Ogni router deve sapere dove inviare il pacchetto che ha un indirizzo di gruppo.

Protocollo IGMP. (*slide 22*)

Interviene un protocollo ausiliario: il protocollo IGMP, che trova sua applicazione principale la trasmissione tra un end-system e il router multicast a lui più vicino. Definisce la trasmissione tra endsystem e il router multicast sulla sua rete locale: se voglio far parlare multicast e fare una trasmissione che esca fuori dalla rete, ho bisogno nella mia LAN un router multicast, che deve poter inviare informazioni di controllo (usando il protocollo IGMP), che servono a propagare informazioni di membership per vedere quali router partecipano alla trasmissione e quali non partecipano alla trasmissione. I destinatari devono mandare messaggi IGMP al proprio router per definire l'interesse a partecipare ad un gruppo. Non è sufficiente dire al router multicast che voglio appartenere ad un gruppo, ma il router deve propagare l'informazione di interesse ai router più all'interno della rete, in modo che vengano inviati pacchetti a quell'end-system.
Come i router multicast scambiano messaggi tra di loro è definito tramite protocolli di multicast routing, l'IGMP fa coordinamento locale. I **multicast routing** implementano routing che servono a fare meccanismi di instradamento all'interno della rete. A volte queste informazioni si possono anche trasmettere con IGMP. I messaggi che viaggiano dagli end-system verso router sono messaggi di **report**, mentre i messaggi originati dai router sono chiamati messaggi di **query**.

Ci sono 2 messaggi di **query**:

- **Generale**: messaggio che serve ad un multicast router per chiedere se ci sono nella rete locale host che sono interessati a partecipare ad un gruppo multicast.
- **Specifico**: se un router vuole sapere se ci sono host locali che partecipano ad un determinato gruppo multicast.

Gli host devono rispondere con messaggi di **report**, a volte sollecitati, a volte non, che sono di 2 tipi:

- **Membership report**: informa il multicast router locale che l'host vuole unirsi ad (o fa parte di) un determinato gruppo multicast.
  Quando un host vuole partecipare ad un gruppo può anche inviare un **join unsollecited.**
- **Leave group**: informa il multicast router locale che non sono più interessato a ricevere messaggi di un gruppo su cui ho fatto il join

(*slide 26*)

Questa informazione di entrata in un gruppo non deve rimanere nella rete locale, ma deve

coinvolgere gli altri router della rete, e questa funzione è affidata al multicast router, che parlerà con altri multicast router per aggiungere gli host al gruppo. In realtà, bisogna prevedere che di multicast router che ne può essere anche più di uno.

La **membership** è **dinamica**, cioè gli host che fanno parte di un gruppo possono variare nel tempo, allora sotto questi pacchetti viaggiano in datagrammi IP che possono andare persi, quindi l'informazione di membership che ha il router non è detto che sia sempre affidabile (ad esempio, un end-system fa un leave, ma il messaggio di leave si perde). La membership che mantiene il router deve essere gestita con accortenza, quindi il router fa un'azione di **polling** sulle varie reti locali, per vedere se c'è qualcuno che vuole ricevere informazioni, quindi si devono continuamente inviare messaggi di richiesta da parte dei router. IGMP prevede un meccanismo di coordinamento lasco tra i router, per evitare che si possano sovraccaricare la rete e i router con messaggi di controllo.
Il multicast router fa una **poll request** (si fa una richiesta di membership generale e non specifica).

Se un router fa un messaggio di membership query, si fa in modo tale che non rispondano tutti gli end-system della rete, ma è sufficiente sapere che ce n’è almeno uno, e non è necessario che rispondano tutti. Infatti il router comunque invia una sola frame ethernet con un indirizzo di destinazione particolare che manda il pacchetto solo agli host interessati, ma il pacchetto è unico. Allora, al router, non interessa sapere che ci sono 4,5 host connessi, ma gliene basta 1. Allora, gli end-system si accordano, cioè quando un host vede arrivare un messaggio di richiesta, aspetta un tempo **casuale** (su un intervallo massimo di 10 secondi) per rispondere, e se in questo lasso di tempo di attesa, l'end-system vede la risposta inviata da un altro end-system, inviata sempre in broadcast, si sta zitto, non invia la risposta, perchè c'è già qualcuno che ha inviato la risposta. In questa maniera si evitano di avere tante risposte superflue, ma se ne ha soltanto una.

Reti di Calcolatori - 15/11/2012

Lezione 20. IP multicast e multicasting routing

Quando un pacchetto che ha come indirizzo destinazione un gruppo multicast arriva al router che si trova immediatamente adiacente ai nodi di destinazione finali, questo pacchetto non viene replicato in n istanze se la rete locale supporta una comunicazione di tipo multicast, ma il router mette sulla rete locale un solo pacchetto con indirizzo destinazione quello del gruppo multicast, che viene incapsulato in un indirizzo che è funzione dell'indirizzo multicast IP. Quando invece il pacchetto arriva ad un router, questo forma, dall'unico pacchetto in entrata, due distinti pacchetti che vengono trasmessi sui due link differenti. Il router multicast ha quindi questa **capacità di generare copie multiple del pacchetto**. Sull'ultimo hop questa multiplazione non avviene.

Algoritmo che consente di mappare l'indirizzo multicast di destinazione con l'**indirizzo MAC** di destinazione. Questi indirizzi vengono mappati su 48 bit, rappresentati comesei coppie di cifre esadecimali, e l'indirizzo MAC serve a identificare univocamente una scheda di rete. Un indirizzo MAC è strutturato in **due parti**, una parte che identifica il costruttore e una parte che il costruttore dà ad ogni macchina. La prima parte è chiamata **OUI**.
Quando un host vuole inviare un pacchetto multicast, usa come OUI la sequenza **01:00:5E**, che è riservata per identificare frame ethernet che identificano multicast. La restante parte dell'indirizzo viene utilizzata per identificare il gruppo di destinazione del pacchetto.
Se un host o un router deve trasmettere su una frame ethernet un indirizzo di classe D (1110) questo si mappa su un indirizzo MAC che comincia per 01:00:5E, e i restanti 23 bit sono liberi, e a questi vengono assegnati i corrispondenti 23 bit nella destinazione. Quindi non si può fare un mapping bit a bit tra un MAC e un Ethernet, perchè 5 bit sono vincolati ad assumere una certa configurazione. Di conseguenza, pacchetti con indirizzo multicast distinti possono essere mappati sullo stesso **indirizzo multicasting map**, cioè sullo stesso indirizzo di multicast MAC su DataLink.
Questo significa che due gruppi di destinazione possono essere confusi sulla rete locale, però la probabilità di collisione è piccola, quindi si ignora volontariamente il problema. Vuol dire che arrivano più pacchetti a livello superiore, che però in caso, guardando l'indirizzo di destinazione, verranno eliminati. E' solo un problema di ricezione di pacchetti non richiesti. Quando un host vede un pacchetto con indirizzo di destinazione una stringa di bit che inizia con questa sequenza, sa che è multicast, e deve capire se i pacchetti gli interessano o meno, e la scheda di rete fa questo controllo, che è istruita su quale gruppo l'host ha fatto join. In caso sia iscritto, la scheda di rete prende il pacchetto e lo consegna ai livelli superiori, cioè IP. Altrimenti il datagramma viene ignorato.
L'indirizzo multicast ha 28bit liberi.
Nei 32 bit dell'indirizzo IP multicast i primi 4 sono vincolati ad essere 1110, e quindi i gruppi multicast che si possono gestire sono al massimo 2^{28}. Su un segmento di rete locale ne posso gestire al massimo 2^{23}, perché sul MAC ho liberi solo 23 bit, perché gli altri 25 sono vincolati ad assumere quella configurazione.
Le **frame multicast** sono diverse dalle frame broadcast, perchè mentre le seconde sono trasmesse in maniera tale che siano ricevute da tutti gli host, le prime sono sentite da tutti gli host ma non è detto che gli host le prelevino e le trasmettano ai livelli superiori.

Reverse Path Forwarding: instradamento dei datagrammi.

Ci sono analogie e differenze rispetto alla trasmissione broadcast. La trasmissione broadcast vuole far arrivare un pacchetto a tutta la rete. Il multicast fa delle modifiche che rendono la comunicazione più efficiente, perchè il pacchetto deve arrivare solo a quelle parti della rete che sono interessate a ricevere il pacchetto. Tra le tecniche la più efficiente è la **RPF *(Reverse Path***

Forwarding). Questa tecnica consiste nel trasmettere un pacchetto su tutte le altre interfacce rispetto a quella da cui arriva il pacchetto, ma solo per i pacchetti che provengono dal link che appartiene al percorso più breve tra destinatario e ricevente. Ogni router verifica se il link di provenienza è quello sul quale lui trasmetterebbe il pacchetto se dovesse inviarlo al mittente (per questo "reverse"). Ci sono comunque pacchetti inutili.

Esempio: Abbiamo un pacchetto trasmesso dal nodo sorgente al router A. Il router A riceve il pacchetto e lo invia sui due link. Quando C riceve il pacchetto da A vede che il pacchetto ariva dal lato buono per andare ad A. Quindi, applica la stessa tecnica di prima. *[…]* Questa tecnica non è massimamente efficiente perchè ci sono pacchetti inutili, in quanto i pacchetti che viaggiano da C a B sono mandati inutilmente. Se non fossero stati trasmessi sarebbe stato meglio. Quando C si vede arrivare questo pacchetto, non lo inoltra, perchè il pacchetto ha come mittente sempre A, e se dovesse trasmettere il pacchetto verso A non lo inoltrerebbe su quel link. Quindi il pacchetto non viene ulteriormente inoltrato. Se a B arriva un pacchetto dal lato sbagliato, esempio C, non lo inoltra. I pacchetti che arrivano dal lato buono sono inoltrati a valle. Anche F quando lo riceve da C lo inoltra ad E. Quando E si vede arrivare il pacchetto da C, lo ha ricevuto dal lato buono, e quindi lo duplica e inoltra il pacchetto verso F e verso D. Quando D lo riceve da B lo inoltra e lo copia. Quindi abbiamo due pacchetti per ogni link trasmessi inutilmente.
Tutti i router hanno ricevuto una copia del pacchetto, e quindi questa tecnica è una tecnica di broadcast. Ma noi vogliamo fare multicast. Nella trasmissione multicast, siccome l'appartenenza ad un gruppo può essere a macchia di leopardo, allora ci sono router che obbligatoriamente devono riceve il pacchetto e mandarlo avanti, altrimenti la trasmissione si blocca, anche se il router non è interessato al multicast. Ad esempio, se C non inoltra i pacchetti, ad E ed F non arrivano i pacchetti. Anche D e G ricevono i pacchetti perchè stiamo in broadcast, e potrebbero tirarsi fuori senza danneggiare nessuno. G è un router periferico, quindi i pacchetti, una volta che arrivano a lui, a valle non vanno più, perchè è finita la rete, quindi se G riceve i pacchetti e sa che non è nessun host a lui collegato è interessato a riceverli, allora dopo un po' di tempo può chiedere a chi gli sta inviando i pacchetti di smettere di fargli arrivare i pacchetti perchè lui non ne è interessato e non deve ritrasmetterli alla rete.
Allora c'è il **TRPF *(Truncated Reverse Path Forwarding)*** in cui un router, che non vuole più ricevere pacchetti, fa richiesta al router che gli sta a monte nel flusso di pacchetti di non ricevere più pacchetti, cioè di essere eliminato dalla trasmissione. Questo messaggio si chiama **pruning** (che vuol dire *"potare"*). Questo messaggio D lo prende in considerazione e dopo un po' di tempo succede la stessa cosa per D verso B, poichè non è interessato a ricevere il messaggio multicast. Questa tecnica funziona in periferia, ma i nodi che stanno più all'interno della rete devono collaborare nella trasmissione. C non potrà tirarsi fuori.
Si pensa che i router conoscano già l'instradamento, così da prendere decisioni. La tecnica crea un albero per ogni mittente ed essendo un albero che lavora per il broadcast non prende informazioni sui gruppi.
La tecnica costruisce un albero ad ogni mittente, e non tiene in considerazione gruppi perchè […].
In generale, la tecnica migliore per la trasmissione multicast è quella di organizzare un **albero minimo di copertura**, che ricopra i router della rete che hanno fatto join al gruppo multicast. Quando si crea un albero di trasmissione un pacchetto viene trasmesso dalla radice dell'albero in cascata a tutte le foglie, quindi arriva ai router interessati a ricevere pacchetti. Questa è una collaborazione tra router multicast che, quando ricevono i pacchetti, grazie al protocollo IGMP, sanno se i pacchetti devono essere inoltrati alla rete locale o meno.

In realtà, un albero ha una sola radice, e questa radice che relazione ha rispetto ai mittenti? Si possono usare due approcci:

- **Group-shared Tree**: Costruire un unico albero di copertura indipendentemente dalle

sorgenti, dove viaggiano i pacchetti di tutti i mittenti. Ogni mittente fa in modo tale da far ricevere il pacchetto dalla radice dell'albero e poi il pacchetto viene trasmesso alle foglie. Il pacchetto viene fatto viaggiare con una trasmissione unicast dal mittente alla radice.

- **Source-based Tree**: Per il gruppo si creano tanti alberi di copertura quanti sono i mittenti del pacchetto. Quindi vediamo disegnati, per lo stesso gruppo, più alberi.

I due approcci hanno vantaggi e svantaggi.

Il **primo approccio** ha il vantaggio che ogni albero di distribuzione può essere efficiente rispetto alla posizione del mittente. Quindi, se il mittente è D, il pacchetto deve prima viaggiare da D ad A, e poi c'è il problema del router da scegliere come radice dell'albero, e la scelta può avere influenze sulla distribuzione dei pacchetti. Estremizzando la situazione, supponiamo di avere 100 router negli USA e 1 in Giappone. Non conviene andare a mettere la radice dell'albero in Giappone, perchè il 99% dei pacchetti faranno un viaggio dal giappone agli stati uniti, ma conviene mettere la radice negli USA, visto che i pacchetti viaggeranno localmente negli Stati Uniti e solo una piccola frazione di pacchetti viaggeranno su un percorso lungo.
Il **secondo approccio** è efficiente nella distribuzione perché costruisce un albero per ogni sorgente, ma ha come lato negativo la complessità, perchè bisogna gestire tanti alberi di distribuzione differenti per ogni mittente, e questo vuol dire che è necessario mettere delle regole che tengano conto del mittente più della destinazione, cioè è necessario mettere nei router informazioni sulla sorgente del pacchetto oltre che la destinazione, cosa che non succede mai nell'IP.

L'approccio Group-shared Tree ha come problema fondamentale quello di andare a costruire l'albero nella maniera più efficiente possibile. E trovare il modo ottimo di costruire l'albero è un problema complesso e non è trattabile in maniera efficiente in tipologia arbitraria, quindi per costruire un albero ottimale ci vuole la soluzione ad un **problema di** costruzione dell'albero di **Steiner**.
Questo approccio ha anche altri inconvenienti: ogni volta che cambia la membership del gruppo, deve essere ricalcolato l'albero, e ciò può essere problematico per la dinamicità della membership.
Gli approcci di tipo shared tree eleggono a priori dei nodi privilegiati che hanno il **ruolo di core**: si definisce il nucleo dell'albero e l'albero viene costruito in maniera incrementale a partire dal core. Il core nasce subito come tale e gli altri router si aggiungono mandando dei messaggi di join e facendo l'innesti (***draft***) all'albero. Il nodo di core spesso viene detto ***nodo di rendez-vouz***.

Nell'approccio Source-based Tree viene costruito un albero per ogni sorgente in modo da minimizzare il costo di propagazione multicast dei pacchetti. In questo modo non c'è bisogno del pruning, perchè G e D non vengono proprio innestati al gruppo se non fanno la richiesta di join. C'è un meccanismo di innesti successivi.
Mentre l'approccio Group-shared Tree che costruisce un albero di steiner minimizza il costo globale, l'approccio Source-based Tree minimizza il costo dalla singola sorgente verso ognuna delle destinazioni.

Protocolli per il multicasting.

L'algoritmo implementato dai router che parlano **DVRMP** *(Distance Vector Multicast Routing Protocol)* è un algoritmo di tipo RBT, che è una variante della **RPF** (*Reverse path forwarding)*, che quindi si presta bene a situazione di tipo brodcast: si nasce per fare broadcast, e poi si passa al multicast. Uno dei problemi fondamentali del multicast è quello dell'interazione con i protocolli unicast: lo stesso RPF deve essere in grado di determinare la rotta unicast verso il destinatario del pacchetto. Questo tipo di interazione è semplice da realizzare quando il router multicast coincide fisicamente col router unicast, ma nella realtà, un router multicast è spesso un oggetto distinto dal

router unicast che si collega all'edge della rete, cioè tipicamente le funzionalità di router multicast e il protocolli di router multicast venivano implementati in delle macchine distinte dal router, e in questa situazione l'interazione diventa più complicata.

Il protocollo **MOSPF** *(Multicast Open Shortest Path First)* è un protocollo derivato da OSPF, facendo sì che i router si scambino informazioni relative all'appartenenza ai gruppi.

Il protocollo **CBT** *(Core-Based Tree)* implementa la filosofia GST in cui si crea un unico albero a partire da un unico centro (*"core"*), e poi esistono dei meccanismi di rinfresco che consentono di potare i rami inutili dall'albero. In diversi contesti si parla di ***soft-state***, che è un'informazione di stato (di qualunque tipo sia lo stato) che viene gestita da un dispositivo in una maniera particolare, tipicamente associata ad un timeout: si sa già in partenza che lo stato non viene mantenuto nel tempo, ma vale finchè non viene rinfrescato da un messaggio ulteriore da altro router che ne rivaluta lo stato. Se il rinfresco non arriva e scade il timeout, il collegamento sparisce. Questo tende a irrobustire il sistema, anche perchè evita lo spreco di risorse.
Uno **stato di un dispositivo di rete** significa una situazione per cui esiste una giustificazione per un certo traffico, per un certo uso di risorse. Uno stato nasce per una richiesta esplicita di altre entità, e in questo tipo di situazioni, quando nasce lo stato, esso deve anche morire, perchè le risorse possono non essere più necessarie. Siccome nelle reti IP i messaggi che veicolano queste richieste di allocazione di risorse possono andare persi, in quanto viaggiano su datagrammi, allora se un dispositivo crea un concetto di stato in un altro dispositivo e poi la richiesta di cancellazione dello stato e il rilascio di risorse viene perso, le risorse possono restare bloccate all'indefinito all'interno del dispositivo. Per questo si implementa il soft-state. Il timeout tipicamente viene messo pari a un multiplo del tempo di periodicità del rinfresco. Questo significa che si devono perdere 4-5 messaggi consecutivi di rinfresco affinché lo stato venga eliminato. Il soft state svanisce se non rinfrescato.
L'***hard-state***, invece, viene creato e deve essere esplicitamente eliminato senza timeout.

Il protocollo **PIM** *(Protocol Indipendent Multicast)* è indipendenti dai protocolli di unicast: essendo un protocollo che nasce per funzionare per reti grandi, fa i conti sul fatto che in una rete grande ci possono essere parti che funzionano con vari protocolli, e quindi non si possono collegare protocolli unicast a protocolli multicast. Il protocollo PIM si divide in **PIM Sparse mode** e **PIM Dense mode**, e sono protocolli differenti che vengono utilizzati in situazioni differenti. La diversità delle situazioni è legata alla densità di distribuzione dei nodi sulla rete. Un certo gruppo multicast è denso se la maggior parte dei router della rete è coinvolta nella trasmissione. Stiamo in una situazione che non è di broadcast, ma può essere approssimata a tecniche di tipo broadcast.
Il protocollo **PIMDM** usa quindi un approccio *Reverse Path Forwarding*, ed è quindi un protocollo che tende ad allagare la rete quasi completamente per poi fare pruning.
Il protocollo **PIMSM** ci vuole quando gli host sono distribuiti in maniera sparsa sulla rete, e quindi solo una parte di router deve partecipare alla trasmissione, per cui si utilizza un ***approccio center-based***, utilizzando un core e un unico albero di trasmissione, e i router che vogliono partecipare alla trasmissione usano join.

Quando parliamo di inoltrare pacchetti multicast interdominio sono usati altri protocolli, come **DVRMP** e **BGMP** *(Border Gateway Multicast Protocol)*.
Routing interdominio significa pacchetti che devono essere trasmessi nella stessa rete ISP, e l'esigenza di routing interdominio è chiara, perché è ovvio che in una rete ottimizzata per domini serve per avere comunicazione globale. Il routing interdominio di pacchetti multicast è un problema ancora non risolto, nel senso che la trasmissione di programmi multicast interdominio è una cosa che si vede poco nella realtà. In realtà, il traffico multicast non è tanto facile da gestire nei rapporti tra ISP, che stabiliscono tra di loro dei rapporti di tipo commerciale. Finchè si parla di traffico

unicast, è facile implementare degli accordi commerciali con dei criteri tra ISP, mentre nel caso di inoltro di pacchetti multicast questo problema diventa più complicato, perchè è difficile quando vedo un pacchetto multicast alla fine a chi arriverà, e di conseguenza gli ISP hanno tardato a ideare dei modelli di tariffazione che consentissero di scambiare questo traffico, che, tra l'altro, se utilizzato, genera anche volume di traffico elevati, e quindi gli ISP cercano di non far viaggiare traffico multicast tra vari ISP. All'interno di un singolo ISP, invece, c'è l'utilizzo della trasmissione multicast usando supporti proprio per il multicast.

Come si fa **trasmissione multicast a livello interdominio senza il supporto ISP**? Se l'ISP decide che un router non è abilitato all'inoltro del routing multicast, il multicast nella rete dell'ISP non passa, ma passa soltanto a livello locale. Il modo più efficiente di realizzare un multicast è attraverso l'uso di router che contemporaneamente supportino sia multicast che unicast. Quando questo non è possibile, nella rete posso assumere come esistente solo il servizio unicast, e allora si usa un approccio di rete Multicast BackBone.

La rete Mbone: Multicast BackBone. (*slide 19*)

Questo approccio è stato sperimentato nella rete Multicast BackBone, che è una **rete virtuale** creata **al di sopra della rete fisica**, in cui esistevano router multicast, che vivevano all'interno di normali host, e questi router multicast erano collegati attraverso **collegamenti** non più fisici, ma **logici**. Se devo far viaggiare dei pacchetti multicast e questi devono attraversare un tratto di rete con due router multicast (R1 ed R2, che sono normali workstation) e una rete internet che non supporta il multicast ma solo l'unicast, il pacchetto dovrebbe venire scartato. Se voglio farlo arrivare stabilisco un **tunnel virtuale**: quando arriva un pacchetto multicast, viene incapsulato in un pacchetto unicast in un IPv4 unicast che ha come destinazione un altro router. Quando il pacchetto arriva all'altra estremità del tunnel, il router R2 vede che il pacchetto è incapsulato ed è multicast e lo manda a destinazione riportandolo a pacchetto multicast. Gli end-system non sono consapevoli dell'esistenza dei tunnel. La membership è stabita con approccio soft-state. In questa maniera, si crea una topologia logica sovrapposta ad una topologia logica che di fatto è costituita da tunnel: è buono se la topologia logica è quanto più rispondente con la tecnologia fisica. L'MBone si basa su una **topologia di overlay** sviluppata alla fine degli anni '90 per provare questo tipo di trasmissioni. L'entità di router multicast era un daemon. Il router multicast era distinto dall'unicast e la rete poteva essere collegati con vari unicast.

Con degli **MBone tools** si possono vedere tutte le sessioni trasmesse da un MBone e le associazioni usate per trasmettere flussi (IPmit). Si mostra il titolo della sessione descritta da un metadato. Si poteva anche creare una sessione. Tipicamente i flussi trasmessi con multicast usano UDP a livello trasporto. Il TCP nasce per trasmissioni punto-punto. C'è anche un'informazione di durata temporale della sessione.
Il problema di allocazione di indirizzi multicast è spesso risolto da un server centralizzato (mask). L'assegnazione va fatta dinamicamente: un gruppo non può essere dedicato indefinitamente nel tempo (ad esempio gli announcement MBone venivano trasmessi dalla 'guida programmi'). Ma sono solo gruppi specializzati nell'uso. Gli altri sono assegnati limitatamente.

La rete Internet.

Internet è nata attraverso un'interconnessione di più reti differenti in gestione a provider commerciali, che non sono tutti uguali perchè, rispetto alla dislocazione geografica dei nodi chi gestisce i link intercontinentali è un provider che ha un ruolo abbastanza importante. Per quanto numerosi sono relativamente pochi perché molto costosi. Questi sono **operatori di backbone**, che

implementano la dorsale della rete. All'inizio, della rete internet c'era un solo provider di backbone, ma poi anche i servizi di backbone sono nati come provider commerciali, e quindi sono nati servizi di backbone. Queste reti si interconnettono in punti di interconnessione che sono **punti di accesso**, e tipicamente un provider di livello gerarchico alto danno connettività a provider di livello gerarchico più basso (**ISP**, che agiscono su scala limitata) in maniera commerciale portando un cavo fino a un dato punto fisico. I provider stabiliscono dei collegamenti di interconnessione paritetici: quando provider si interconnettono, e questo avviene in punti peering point. Un ISP piccolo si collega ad uno di livello maggiore perché gli offre connettività col resto del mondo, mentre due provider di uguali dimensioni si interconnettono perchè hanno un beneficio mutuo, quindi uno scambio alla pari, e se la cosa è vantaggiosa commercialmente per entrambi se stabiliscono una connessione paritetica che non è onerosa. Queste reti formano gli **Autonomous System (AS)**, che sono porzioni della rete internet, dove un AS è una collezione di reti amministrate da da un'unica autorità. Dentro l'AS tutte le problematiche di routing vengono gestite da chi possiede quell'AS, e le scelte di instradamento sono fatte per ottimizzare le scelta di invio delle risorse. I grossi ISP posseggono alcuni AS (i più piccoli 1, i più grandi di norma fino a 4). Il problema del routing è gestito in maniera differente quando parliamo di singolo AS e diversi AS. E' quindi un routing intradominio. Il routing tra domini viene gestito con tecniche differenti perchè nella valutazione di percorsi entrano altri fattori di tipo commerciale. A volte due ISP sono in concorrenza e magari un ISP preferisce scambiare pacchetti con un altro ISP. Quindi intervengono nella scelta dei percorsi altri fattori.

Reti di Calcolatori - 16/11/2012

Lezione 21. Internet e il routing gerarchico - Autonomous System

Il **routing intradominio** è l'instradamento interno al sistema autonomo, mentre il **routing interdominio** è l'instradamento tra più sistemi autonomi.
Di fatto, dal punto di vista dell'instradamento, i router partecipano a diversi livelli, perchè ci sono **router intradominio** e **router interdominio**. Al routing intradominio partecipano i router di frontiera, e le scelte che questi router fanno devono essere comunicate ai router interni.
(*slide 14*) C'è una differenza strutturale, e in generale gli Autonomous System si dividono in:

- Autonomous System che hanno un solo collegamento con il resto del mondo (**stub**)
- Autonomous System che si collegano ad almeno 2 o più Autonomous System (**multi-homed**), che possono giocare due ruoli differenti:
    - **Transit (*provider*)** sono quelli che accettano di essere attraversati da traffico diretto ad altri Autonomous System. Sono collegati a più collegamenti con il resto del mondo e possono avere questi due collegamenti perchè desiderano scambiare traffico verso due direttrici differenti.
    - **Non-transit (*grandi corporate*)** sono quelli che non accettano di essere attraversati da traffico diretto ad altri Autonomous System. Non consentono il traffico da ISP1 a ISP2.

Generalmente un AS ha più collegamenti per ridondanza, per bilanciare il carico, per essere sicuro che se va giù un link non rimane isolato, perché sa già che è comodo raggiungere alcune destinazioni da una parte o dall'altra. Non si fa attravesare dal traffico che non è scambiato direttamente da host che vi appartengono. Non ci sono flussi di traffico che non arrivino in un host che sta in AS1. L'AS può decidere di far passare il traffico che entra da un link ed esce dall'altra parte, e quindi di farsi attraversare, e quindi di giocare il ruolo di transit. Chi gioca questo ruolo è l'ISP, che decide di partecipare al routing in internet a livello interdominio.
Tutti i flussi di traffico che non sono locali all'AS devono necessariamente uscire attraverso il link di collegamento al mondo esterno.
Per l'AS multi-homed nasce un problema: quando c'è un flusso di traffico generato dall'interno dell'AS e che vuole raggiungere una certa destinazione che sta da un'altra parte e che magari è raggiungibile da entrambe le parti, c'è il problema di capire questo flusso di traffico se deve essere instradato verso una linea di uscita rispetto ad un'altra. C'è una scelta di instradamento da fare a monte, e una volta fatta questa scelta, il percorso che verrà attraversato sarà un percorso secondo criteri intradominio.

Le tabelle di instradamento dei router di un AS sono sempre composte mediante l'azione congiunta degli algoritmi dei protocolli di routing intradominio e quella dei protocolli di routing interdominio.

C'è una differenza di criteri della scelta dei percorsi inter e intra dominio. All'interno di un dominio ci sono dei protocolli orientati all'utilizzo più efficiente della rete dell'AS, mentre nella scelta dei percorsi tra domini, intervengono criteri che non sono di ottimizzazione ma che sono anche politici, cioè legati ai rapporti tra gli AS e tra ISP che li gestiscono. Allora, il vantaggio di avere un instradamento gerarchico è quello di migliorare la scalabilità, la complessità. Uno spezza il problema della scelta del percorso in due problemi: il primo sta nella scelta dell'AS da attraversare e il secondo sta nella scelta del percorso all'interno dell'AS. Ma, non avendo la visione completa della topologia della rete, questo può condurre a delle scelte non ottimali in termini assoluti.

(*slide 19*) Se nella scelta del percorso si fa una scelta gerarchizzata e quando si lavora interdominio

si fa una miniminizzazione del numero di AS attraversati si sceglie un certo percorso (quello di destra), ma questo percorso può essere scelto rispetto al percorso alternativo, migliore in termini di router attraversari, perchè a livello interdominio sembra conveniente perchè il numero di AS attraversati è minore. Però, facendo questa scelta a questo livello gerarchico superiore non ho visibilità di cosa succede dentro gli AS, e sto facendo una scelta sub-ottima, perchè non tengo in conto il fatto che questo percorso è contorto perchè devo attraversare molti router. Fare la scelta migliore non sarebbe comunque possibile perchè a livello globale posso sapere quanti AS posso attraversare ma il percorso all'interno dell'AS non lo posso sapere perchè gli AS non me li specificano.

Routing interdominio in Internet.

I **router** si distinguono in **interni e di frontiera**: le entry costruite dal routing intradominio servono a raggiungere dispositivi esterni. L'AS deve sapere quali reti di destinazione sono raggiungibili attraverso altri AS. Le informazioni di raggiungibilità sono acquisite inizialmente dai router di frontiera e poi sono propagate a tutti i router dell'AS. Quindi, anche i router interni hanno delle tabelle di inoltro costruire in base all'azione congiunta di algoritmi di instradamento interdominio e intradominio.
Se l'AS è multihomed. La scelta di quale strada deve fare il pacchetto deve essere funzione della destinazione, mentre per l'AS stub non ci sono scelte perchè c'è solo un link che porta fuori dall'AS. Per fare questa scelta, l'AS deve sapere quali reti sono raggiungibili attraverso gli altri AS, per cui alcune destinazioni sono raggiungibili solo da un AS o da più AS e quindi si dovrà operare una scelta. Queste informazioni sono acquisite attraverso i router di frontiera, ma devono anche essere propagate a tutti i router dell'AS.
Supponiamo di ragionare su una destinazione X **(*slide 5*)**. Ci possono essere più entry che matchano la destinazione, e normalmente si prendono le entry che matchano la destinazione con più numero di bit.

Supponiamo che l'AS1, attraverso un protocollo di routing interdominio, apprenda che la sottorete X sia raggiungibile attraverso AS3 e non attraverso AS2, e quest'informazione si propaga attraverso tutta la rete e il router 1.d, sapendo che X è raggiungibile attraverso AS3, deve determinare il percorso di costo minimo per raggiungere il punto di uscita 1.c. Se la stessa destinazione X diventa raggiungibile da AS2, l'AS1 apprende questa cosa successivamente e quando un pacchetto con destinazione X arriva a 1d bisogna scegliere il punto di uscita del pacchetto. Quale punto di uscita prendere è un problema interdominio. Il criterio che si sceglie per determinare il punto di uscita è il **criterio della hot potato**: quando si sa che per una certa destinazione x può essere raggiunta attraverso più punti di uscita, si prende il punto di uscita più vicino.

Protocollo di routing interdominio: BGP *(Border Gatewat Protocol)*

E' un protocollo complesso, la versione attuale è la 4 e viene usato per le scelte interdominio. Il compito principale di BGP è quello di consentire ad AS di conoscere informazioni di raggiungibilità verso aggregati di indirizzi IP, quindi attraverso BGP un AS conosce un'informazione di raggiungibilità di un insieme di reti dagli AS vicini, collegati direttamente. Dopodichè, queste informazioni di raggiungibilità sono propagate a tutti i router interni dell'AS. Poi, attraverso queste informazioni, il protocollo determinare rotte buone a seconda di criteri di scelta che sono configurati dall'ISP che gestisce l'AS sulla base di vari criteri, e queste policy decretano qual è il percorso a livello interdominio che vince. Il meccanismo di base è quello di consentire ad una rete di annunciare la sua esistenza attraverso internet.

Il protocollo **EGP** ***(Exterior Gateway Protocol)*** non è uno specifico protocollo, ma è una classe di protocolli interdominio. Si utilizza il **path vector**: vengono scambiate informazioni di raggiungibilità per ogni specifica rete associando alle possibili destinazioni la sequenza di AS che si attraversano per arrivare a quella destinazione. I router si scambiano informazioni tra di loro in sessioni, che sono instaurate mediante **connessioni TCP** sulla **porta 179**. Sessioni BGP si instaurano tra router direttamente collegati (es.: due router di frontiera di AS differenti), e a volte si possono creare sessioni BGP amche tra router non direttamente collegati, perché in ogni caso i router di frontiera si attraversano comunque. La sessione indica un dialogo che consente ad una coppia di router BGP di scambiarsi informazioni attraverso una connessione TCP.
I router BGP comunicano la sequenza di router da attraversare per una destinazione. Si parla di **sessioni esterne** e **sessioni interne**.
I router ai capi di una connessione TCP sono chiamati **peer BGP** e la connessione TCP, con tutti i messaggi BGP che vi vengono inviati, è detta **sessione BGP**. Nel caso in cui questa coinvolga due siostemi autonomi viene detta **sessione BGP esterna (*eBGP*)**, mentre quella tra router dello steso sistema autonomo è chiamata **sessione BGP interna (*iBGP*)**.

Un prefisso non sta in corrispondenza 1-1 con una singola rete fisica perchè gli indirizzi contigui possono essere aggregati considerando la loro parte comune. Annunciare un percorso significa che un router di frontiera può memorizzare per una certa destinazione Z un certo cammino, dove questo cammino sta a indicare la sequenza degli AS attraversati, dopodichè, quando questa informazione viene ricevuta da un gateway paritetico, la può prendere per buona oppure ignorarla, se nn è interessata a raggiungere la destinazione Z. Se la prende per buona aggiunge se stesso al percorso. Questa scelta del cammino è basata più su scelte commerciali che su costo di link.
Il router stabilisce la connessione con un altro **router IGP** (*Interior Gateway Protocol*) e dopodichè scambia informazioni di raggiungibilità e fa un monitoraggio della connessione.
Questi percorsi non sono associati a delle metriche. L'unica metrica possibile è quella di basarsi sul **numero di AS attraversati**, perchè non c'è nessun'altra metrica. Un percorso può essere buono o non buono se attraversa ISP che ci interessano o meno. Quando un router BGP riceve un annuncio può non considerarlo. Infatti, quando un router BGP riceve da un vicino un annuncio che contiene nel percorso dell'AS attraversato il proprio numero di AS viene ignorato (questo genererebbe un percorso circolare). Quando un router BGP riceve un annuncio, promette che, se un AS vicino gli dà un pacchetto verso una destinazione, esso è in grado di inoltrarla verso quella destinazione.
Con questo tipo di approccio (***path-vector***), quando ricevo da un vicino un cammino di raggiungibilità, posso guardare tutto il percorso e dire se il percorso mi piace o meno guardando non solo il next hop ma anche più avanti.
BGP fa una distinzione tra meccanismo di instradamento e politica di instradamento, nel senso che le scelte di percorso vanno fatte filtrando gli annunci dei vicini. In particolare, BGP non specifica il percorso da fare, ma l'aspetto tecnico riguarda solo la possibilità di annunciare i percorsi. In assenza di policy di preferenze il percorso è il più breve nel senso di minor numero di AS attraversati. Poi c'è l'attività di annunci di percorsi ai vicini.

Questi percorsi sono messi nei messaggi BGP attraverso degli attributi che contengono la rotta**. AS-PATH** contiene la lista degli AS attraversati e il **NEXT-HOP** ha l'identità del prossimo router da attraversare. Il protocollo prevede che i router scambiano dei messaggi di vario tipo:

- **OPEN**: inizializza la connessione e permette la comunicazione.
- **UPDATE**: aggiornamento sulla raggiungibilità già annunciata.
- **NOTIFICATION**: reazione a eventi considerati errati.
- **KEEPALIVE**: verifica che il peer sia ancora attivo.

Due router che creano una sessione si basano inizialmente sull'intera tabella di raggiungibilità, dopodichè le modifiche si fanno solo attraverso messaggi di update. Gli update possono essere di withdraw, cancellazione del percorso, o di annuncio di nuovi percorsi.
Due ISP non faranno parlare i loro router BGP se prima non hanno stipulato un contratto, e quindi si mettono anche d'accordo sul meccanismo di autenticazione. Il routing di questo tipo può essere soggetto a comportamenti anomali: se per errore un AS comincia a dichiarare informazioni di raggiungibilità verso grandi parti della rete in maniera errata con un basso numero di AS attraversati, può diventare un "***buco nero***" del traffico della rete internet: collassa tutto verso l'AS che va in congestione e non consegna il traffico alle destinazioni. In realtà, internet funziona basandosi molto sul corretto comportamento dei vari ISP.
Un router BGP deve essere anche in grado di agire come router interno.

Il RIB è la base di informazioni ricevute dai vicini. Contengono i percorsi con i messaggi di update ricevuti, poi informazioni di routing locale e di instradamento che i router BGP annunciano.
C'è l'informazione di **leave**, che contiene percorsi in percorsi differenti e ci sono […].

I provider di backbone devono interconnettersi necessariamente, ma anche quelli di livello inferiore preferiscono stabilire interconnessioni tra di loro, così evitano di passare per il backbone, laddove possibile. Gli accordi paritetici (connessioni) si stabiliscono tra provider equivalenti e avvengono tra punti privilegiati detti **IXP (*Internet eXchange Provider*)**, che dà il presupposto fisico affinchè possano stabilirsi queste interconnessioni. L'interconnessione avviene attraverso degli switch ethernet a fibre collegati in un territorio neutrale. L'IXP migliore è quello che sta vicino a più ISP. Chi gestice IXP non è un ISP, è solo una società che ha un locale in cui ha questi apparati che sono facilmente raggiungibili tra i vari ISP. L'IXP italiano più importante è il **MIX** *(Milan Internet Exchange)*.

Esempio: aggregazione degli annunci.

AS2 è collegato a 4 router con indirizzi:

a. 138.16.64.0/24
b. 138.16.65.0/24
c. 138.16.66.0/24
d. 138.16.67.0/24

e deve comunicare ad AS1, a cui è collegato, l'informazione,e invece di inviare questi 4 indirizzi li raggruppa in uno solo e gli manda la parte comune: 138.16.64.0/22
NB.:
$64_d = 01000000_b$
$65_d = 01000001_b$
$66_d = 01000010_b$
$67_d = 01000011_b$

Facendo quest'aggregazione, annuncio l'informazione con un solo indirizzo. Così tutti gli annunci sono gestiti con la **CIDR (*Classless Inter Domain Routing*)**.
128.16.67.0/24 è un annuncio più specifico, e vince sempre, perché gli annunci sono scelti attraverso il **Longest Prefix Match (*LPM*)**. Conviene aggregare perché le eccezioni alle aggregazioni sono gestite attraverso l'invio di informazioni più specifiche.

Il livello trasporto: Il protocollo UDP.

Il livello trasporto è il primo livello **presente solo negli end-system** (anche se ci sono comunque dei router che lo implementano). Qualunque protocollo di trasporto offre un canale di comuncazione logica tra applicazioni su host differenti o sullo stesso host. Differenze tra livello trasporto e livello rete: a livello rete ci si ferma alla consegna dei pacchetti mentre a livello trasporto si fa un'attività di multiplexing e demultiplexing di smistamento dei pacchetti. perché il destinatario non è solo la macchina ma il processo che gira sulla macchina. Il livello trasporto fa altre cose in più. Sostanzialmente il livello trasporto fa da cuscinetto tra rete e applicazioni: offre alle applicazioni un livello di comunicazione ad esse adatto sapendo che il livello rete è inaffidabile. Deve rimediare all'inaffidabilità del livello rete controllando errori, sequenza, flusso, congestione, etc… Il protocollo più completo da questo punto di vista è il **TCP**. L'**UDP** offre solo multiplexing e demultiplexing.
Un host deve potere gestire contemporaneamente comunicazioni tra coppie di processi differenti che girano o su macchine differenti o sulla stessa macchina. Il protocollo trasporto, essendo collegato sopra quello rete, arricchisce il messaggio applicativo di un proprio header così forma una propria **PDU (*Protocol Data Unit*)**, che viene messa come **SDU (*Service Data Unit*)**, protocollo nel datagramma a livello rete.
Il multiplexing viene realizzato attraverso il concetto di **porta**, che è un meccanismo di comunicazione che descritto da un numero intero tipicamente a 16 bit. In realtà, la comunicazione è identificato da un numero di porta sorgente e da un numero di porta destinazione. E' formato da una **quintupla** (come nelle socket): protocollo, IP mittente, IP destinatario, port number mittente, port number destinatario.
Il numeri di porto in alcuni casi sono prestabiliti, e questo è necessario per applicazioni che si mettono in attesa di richieste di servizio (ad esempio, un'applicazione server), e quindi devono essere raggiungibili. Un'applicazione server deve stare in ascolto in un numero di porto noto a priori, e alcuni numeri di porto sono associati biunivocamente al servizio che li utilizza (FTP, HTTP, DNS, BGP). Il lato client della comunicazione usa un porto effimero: quando un client può stabilire diverse comunicazioni contemporaneamente con più server o con lo stesso server non ci si può legare ad un unico porto di client, allora i porti di client sono scelti sulla base di quelli disponibili. Questa scelta lato client viene fatta dal sistema operativo quando apriamo la socket o facciamo il connect (dipende dalla comunicazione che si vuole stabilire). Il numero di porto realizza il multiplexing e il demultiplexing in corrispondenza dell'indirizzo IP, perchè un numero di porto X può essere usato da più client. Però, all'interno della stessa macchina, si devono evitare conflitti, e quindi si devono utilizzare numeri di porto diversi.

I protocolli di livello trasporto più usati sono **TCP** e **UDP**. Ne esistono anche altri per applicazioni particolari: un ulteriore protocollo per trasportare informazioni multimediali è **RTP** (si costruisce sopra UDP). Ci sono anche altri protocolli come **CDP**.

Come si fa la scelta tra TCP e UDP? Ci sono protocolli applicativi che funzionano su TCP, protocolli applicativi che funzionano su UDP e protocolli applicativi che danno la possibilità di scegliere.
I protocolli che lavorano su TCP sono quelli che hanno come requisito fondamentale la sicurezza del trasferimento dell'informazione, e dove vengono mandati messaggi molto grandi, mentre i protocolli che lavorano su UDP sono quelli basati su un modello richiesta-risposta molto semplice, e dove i messaggi scambiati sono piccoli, e l'affidabilità non è il requisito fondamentale (è inutile pagare tutto l'overhead del TCP). Quando i meccanismi di recupero dell'errore sono semplici si preferisce lavorare su UDP (ad esempio il DNS lavora su UDP). Ci sono protocolli che lavorano su

entrambi: questi protocolli scelgono UDP quando le due entità che parlano vivono nella stessa rete locale, dove si assume che la connessione sia più affidabile. **NFS** (condivisione di filesystem in ambiente distribuito) usa entrambi i protocolli.

Il protocollo UDP.

Il protocollo UDP è molto semplice perchè, al di sopra di IP, offre solo il concetto di multiplexing e demultiplexing, cioè duplica il servizio best-effort nel livello rete di IP. I messaggi possono subire perdite, arrivare non ordinati. La comunicazione è semplice, perchè non c'è una fase di inizializzazione, e ogni segmento è inviato indipendentemente dagli altri. Tipicamente, quando si manda un messaggio su un port-number non indirizzato, nella maggior parte dei casi viene generato un messaggio di ICMP di port-number non valido, non utilizzato. E' usato UDP quando non è necessaria la fase di inizializzazione che introduce delay e voglio una comunicazione veloce. È semplice da implementare perchè sender e receiver non devono conservare informazioni di stato. Ci C'è un basso overhead di gestione ma c'è un controllo della congestione assente. E' visto male dagli ISP, che vedono flussi di traffico non regolabili. Ci sono source port number, destination port number, la lunghezza in byte del pacchetto. A livello datalink si parla di frame, a livello logico di datagramma, a livello trasporto di segmento.
Laddove ci siano esigenze di affidabilità, servono meccanismi implementati a livello applicativo.

Reti di Calcolatori - 20/11/2012

Lezione 22. Il protocollo UDP

Spesso gli errori si manifestano a burst (consecutivamente), quindi la possibilità che ci sia più di un bit errato non è trascurabile. Per questo si usa ridondanza: a parità di bit di informazione si usano più bit di controllo.

Checksum.

Nell'header di un pacchetto UDP, oltre ai porti di sorgente e destinazione c'è la checksum, che contiene informazioni di controllo, e viene calcolata dall'entità che trasmette il pacchetto in questa maniera. Il mittente tratta il contenuto del segmento (sinonimo di pacchetto a livello trasposto) come una sequenza di interi di 16 bit. Viene calcolato come complemento ad 1 della somma dei numeri a 16 bit che costituiscono il segmento UDP. Questo incluso uno pseudo-header che include gli indirizzi IP sorgente e destinazione. Questa quantità viene scritta nel campo checksum del segmento UDP. Prima del calcolo della checksum, nel campo viene messo 0, e dopo aver fatto il calcolo viene messo il risultato nella ckecksum. Il ricevente calcola la somma in complemento da 1 dei campi del segmento ricevuto compresa la checksum e, se il risultato è composto da tutti 1 vuol dire che non ci sono stati errori, se invece la checksum non è tutti 1 vuol dire che ci è stato un errore. La condizione che non c'è stato errore non è del tutto vera perchè il calcolo della checksum non mi assicura che non ci sono stati errori non rilevati. Semplici manipolazioni del pacchetto conservano la checksum. Il fatto che sia tutto a posto nel campo della checksum non ci da la garanzia che ci siano stati errori. Anche nel calcolo della checksum si possono verificare errori che non sono rilevati da quest'algoritmo.

Aritmetica in complementi ad 1.

Somma in complementi ad 1. Nella rappresentazione in complementi diminuiti un numero si rappresenta con se stesso se è positivo, col complemento 2^n - 1 se negativo.
Se si deve fare la somma in complementi ad 1 si può usare l'algoritmo di somma modulo 2^n (e non $2^n - 1$), e così si ottiene il risultato della somma, ma in alcuni casi è necessaria una correzione del risultato. Se ad esempio n=8, se devo fare la somma in complementi diminuiti posso usare un addizionatore modulo 256 e poi corregere il risultato. Non devo usare un addizionatore modulo 255 che è difficile da costruire.

In alcuni casi il risultato dell'addizione va corretto:

- se entrambi gli addendi sono negativi la correzione consiste nel sommare +1 al risultato,
- se uno è negativo e uno è positivo allora la somma è positiva.

Si deve fare la correzione perchè, se sommo numeri negativi, un numero si rappresenta con se stesso se è positivo e se è negativo viene preso il complemento 2^n-1 del suo valore assoluto, e quindi rispetto al complemento alla base il complemento diminuito c'è un -1. Sto sottraendo l'unità 2 volte, quindi va compensata la doppia sottrazione dell'unità sommando 1.
Quando la checksum non è tutti 1 allora c'è stato un errore. Se è tutti 1 potrebbe comunque esserci stata una probabilità piccola di errore. L'entità UDP, se vede che c'è un errore, scarta il pacchetto e non lo invia ai livelli superiori. Il controllo viene fatto sull'header, non su tutto il pacchetto,e se vede l'errore butta il pacchetto. Se ci sono errori, UDP non fa altro perchè il modello di servizio che offre è un modello di consegna non affidabile. Quindi le applicazioni sanno che se usano UDP i pacchetti possono non arrivare o arrivare errati. UDP non è affidabile.

Tecniche di recupero dell'errore a livello trasporto.

Sicuramente questa problematica non è affrontata a livello rete per scelte di progetto. Tutte queste problematiche sono tenute in conto a livello trasporto, e sono affrontate a livello TCP.

Se la rete presenta errori di trasmissione (come la perdita di pacchetti) il ricevente, se vuole recuperare l'informazione persa, può fare 2 cose a valle del rilevamento degli errori.
Rileva gli errori con la checksum, e poi può:

- correggere l'errore (più o meno magicamente),
- notificare al mittente che c'è stato un errore nella trasmissione e richiede una ritrasmissione.
    - Il ricevente deve segnalare l'errore e il mittente deve ritrasmettere.

Tutte le tecniche che consentono di rilevare errori si basano sull'aggiunta controllata di **ridondanza**: se devo trasmettere una stringa di bit, trasmetto n+k bit dove i k bit non sono l'informazione ma sono bit di controllo che consentono al ricevitore di verificare la presenza di errori. Si può vedere che, se si aggiunge un certo grado di ridondanza sufficientemente elevato, si può provare a capire, una volta ricevuti dei bit dove c'è un errore, qual'è la configurazione dei bit di origine, e quindi si può ricostruire la sequenza che contiene degli errori. E' possibile grazie a dei codici che aggiungono un grado di ridondanza sufficientemente elevati. Supponiamo che ogni 8 bit dato vengono aggiunti 3 bit di parità calcolati opportunamente, per cui due sequenze valide devono differire sempre in almeno 3 bit. Se in questa sequenza di 11 bit uno solo è errato, vedrò la configurazione che differisce per il minor numero di bit da quella ricevuta.
Allora, in questo contesto, si parla di **distanza di ending**: è il numero di bit per cui due sequenze differiscono. E' più probabile che questa sequenza sia stata derivata da un errore su singolo bit a partire da questa sequenza. Ogni bit si codifica con un certo numero di bit di ridondanza e si cerca di capire, nello spazio n dimensionale delle sequenze, qual è la sequenza valida più vicina. È una scelta che viene fatta sempre in **probabilità**, e si basa sul fatto che è piccola la probabilità che ci sia un errore e si assume che i bit siano indipendenti tra di loro, quindi le probabilità di errore sui singoli bit è indipendente. Se ci sono 3 bit errati, allora la sequenza valida viene trasformata in un'altra sequenza valida, ma la probabilità di trasmettere 3 bit errati suppongo sia bassa. Queste sono tecniche di correzione degli errori, che sono fatte in base a tecniche di codifiche opportune e una correzione degli errori controllati: le **ECC** ***(Error Correcting Code)*** sono il tipo di memorie che usano questi meccanismi.

Recupero dell'informazione mediante richiesta di ritrasmissione.

Protocollo di Stop & Wait senza perdite. (*slide 5*)

- **ACK**: Notifica di corretta ricezione, così il mittente invia altri dati.
- **NAK**: Segnala al mittente che c'è stato un errore e bisogna ritrasmettere

Abbiamo un mittente che trasmette una sequenza di dati, e ogni volta che il ricevente riceve una sequenza valida invia al mittente una ricevuta di ritorno (ACK) , e autorizza il mittente di inviare nuovi dati e di dimenticare i dati trasmessi precedentemente. Il mittente invia una nuova sequenza di dati, che vengono alterati. Allora il ricevente calcola la checksum e rileva l'errore, e segnala al mittente che c'è stato un errore mediante un messaggio di notifica negativa (NAK), e il mittente ritrasmette il dato trasmesso in precedenza. Una volta che i nuovi dati arrivano al ricevente, lui non può sapere se la richiesta di trasmissione è arrivata al sender, quindi non sa se sta rinviando un nuovo pacchetto o è quello ritrasmesso. Questo è il caso di un canale con errori ma senza perdita di pacchetto.
Per risolvere il problema dei duplicati occorre inserire nel pacchetto che contiene i dati un'intestazione che ha un **numero di sequenza**. Quando il ricevente riceve i dati, sa se sono dati nuovi o sono dati ritrasmessi. Lo schema di trasmissione che vediamo è uno schema molto

semplice, in cui chi invia il messaggio aspetta sempre di ricevere un riscontro dalla controparte, quindi un protocollo che viene detto di Stop & Wait. Chi invia questo protocollo, come numero di sequenza usa un solo bit. Il primo segmento viene inviato con n=0. Il secondo segmento dati ha n=1, i dati vengono alterati, quindi il ricevente aspetta di ricevere di nuovo gli stessi dati con n=1. Stavolta i dati ricevuti sono corretti e sono proprio quelli che il ricevente si aspetta di ricevere, allora li prende, li consegna ai livelli superiori, manda l'ACK, e si va avanti.

Protocollo senza NAK.

Se non ci piace mandare il NAK si possono inviare messaggi di ACK, però bisogna fare l'ACK più complicato: bisogna prevedere un messaggio di riscontro che dica il numero di sequenza dell'ultimo messaggio ricevuto correttamente. Se il ricevente non vuole inviare NAK può inviare un ACK ma deve mettere nell'ACK anche il numero di sequenza del pacchetto precedente.
Quando arriva un ACK duplicato, il mittente si vede arrivare due volte ACK (0), e quindi il mittente prende la ricezione di un ACK duplicato come la segnalazione di un ACK e quindi come la necessità di ritrasmettere il pacchetto.

Protocollo di Stop & Wait con perdita di pacchetti. (*slide 9*)

Se i pacchetti si possono perdere del tutto, si gestisce la cosa con un **timeout**. Quando il mittente invia un segmento dati, fa partire immediatamente dopo un timer, che è stabilito su un certo valore. Se, prima che scada il timer arriva l'ACK, tutto a posto. Se invece, allo scadere del timer, non è arrivato il messaggio di riscontro, questo viene visto come un evento di perdita, che si può essere verificato o perchè si sono persi dati o perchè si sono persi riscontri nel viaggio di ritorno. Il comportamento che ne scaturisce è la ritrasmissione dell'ultima sequenza di dati trasmessi. In questa circostanza, il segmento dati con seq=0 viene ricevuto dal mittente 2 volte, e il mittente quando riceve i dati con numero di sequenza 0, li scarta e ritrasmette l'ACK, perchè quello che è stato perso prima non sono stati i dati ma l'ACK, che ritrasmette al mittente.
Se il timeout è troppo piccolo, non si perde nulla ma c'è comunque una ritrasmissione. Un valore ragionevole per il timer è funzione del **Round Trip Time (*RTT*)**: il tempo che impiegano i pacchetti per fare il viaggio di andata e ritorno. Posso porre il timeout un po' più grande del RTT. Questo protocollo è detto **ad alternanza di bit**.

Aumentare l’efficienza: Pipeling. (*slide 12*)

Questo tipo di tecnica porta a delle inefficienze. Se la comunicazione tra due end-system geograficamente lontani tra di loro, il RTT è abbastanza elevato (100-200 ms), e il tempo di attraversamento della rete è determinato dal tempo di propagazione dei segnali su mezzi fisici. Il canale di comunicazione di andata potrebbe trasmettere tanti altri dati, ma sta in attesa che arrivi il riscontro, e quindi non viene impegnato.Questo vuol dire che per aspettare l’ACK è in attesa (idle) per troppo tempo. L'idea base per aumentare l'efficienza è quella di poter **trasmettere più pacchetti prima** di fermarsi in attesa **di riscontro**, cioè inviare tanti dati in attesa del riscontro del primo dato trasmesso. Se trasmetto 10 pacchetti, prima che sia arrivato il riscontro del primo, quando scade il timer, devo poter ritrasmettere questi 10 pacchetti. Stavolta non devo mantenere in memoria un pacchetto ma 10, quindi devo aggiungere dei buffer in sender e receiver. Queste tecniche vengono dette di pipelining. Sono due possibili tecniche.

Coefficiente di utilizzo del link di trasmissione. Supponiamo di avere 1Gbps link, 15 ms di propagazione di ritardo, e un pacchetto di 1 KB.
Con la tecnica Stop&Wait, il tempo di trasmissione è 8 microsecondi. Impegno la linea per 8 microsecondi per trasmettere il pacchetto. Con la tecnica Stop&Wait, solo quando arriva l'ACK il mittente può ritrasmettere il pacchetto. Nell'esempio abbiamo

$T_{trasmissione} = \frac{numero\ di\ bit\ trasmessi}{rete\ di\ trasmissione} = \frac{8\frac{kb}{pkt}}{10^9\frac{b}{s}} = 8\,ms$. Dividiamo il tempo di trasmissione della rete con il tempo totale, e vediamo così in percentuale quanto è impegnata la rete. $\frac{0.008}{30.008} = 0.027$ .Abbiamo una percentuale di utilizzo dello 0,027%, che è un'efficienza non accettabile.
Con la tecnica del pipeline, invece, l'efficienza aumenta di un fattore 3, in assenza di perdita. Viene fatta una **trasmissione back-to-back**: quando finisce un pacchetto subito ne viene inviato un altro. Stiamo anche supponendo che i pacchetti non vengano sfasati tra loro, il che è vero nei collegamenti "in un tubo", ma in realtà è difficile. Il problema è che, quando si verifica una perdita, devo ritrasmettere, quindi devo avere un buffer più grande.

Algoritmo Go Back-N. (*slide 16*)

Non posso usare un numero di sequenza binario, ma devo aumentare il range di valori possibili nel numero di sequenza, e quindi userò un intero. Questo numero di sequenza viene messo nell'intestazione dei pacchetti trasmessi e negli ACK. Questo numero di sequenza è espresso come un numero su k bit. Supponiamo che il protocollo abbia una finestra di trasmissione di N pacchetti trasmissibili senza riscontro. Vengono trasmessi i pacchetti 0,1,2,3.... La sequenza dei numeri di sequenza e le condizioni in cui si possono trovare i pacchetti associati a questa sequenza è questa: in un certo momento, supponiamo che il primo pacchetto che è stato trasmesso che non ha ancora ricevuto un ACK è il send_base. In questo momento, i pacchetti con numero di sequenza da 0 a 5 sono stati riscontrati. I pacchetti con numero di sequenza 6..13 sono stati trasmessi ma il mittente non ha ancora ricevuto riscontri dal destinatario. La finestra N mi limita il numero di pacchetti che posso inviare nell'ipotesi ottimistica. La finestra in questo caso vale 14: le due parti hanno convenuto che possono essere trasmessi 14 pacchetti. Il mittente non si è ancora bloccato anche avendo trasmesso 8 pacchetti, e ce ne ha altri 6 che può trasmettere. Dopo aver inviato gli altri 6 pacchetti, deve aspettare un riscontro, e quando arriva il riscontro, la finestra scorre verso destra. Così come sono numerati i pacchetti, sono numerati anche gli ACK. C'è l'ACK cumulativo: ricevere un ACK(n) significa che anche i pacchetti precedenti sono stati ricevuti correttamente. Se scade il timeout del pacchetto N, il mittente ritrasmette tutto il pacchetto N e tutti quelli successivi al pacchetto N.

(*slide 17*) Quando i dati arrivano fuori sequenza al destinatario, vengono scartati, non è dovere del destinatario memorizzarli Ad esempio il pacchetto 3 viene scartato se non viene inviato il pacchetto 2, quindi manda l'ACK fino a 1 e chiede la ritrasmissione del 2. Ci sono state una serie di trasmissioni inutili.

Ripetizione selettiva. (*slide 18*)

Occorre che gli ACK siano specifici. Occorre inviare un riscontro specifico per ogni singolo pacchetto ricevuto correttamente. Il ricevente deve anche mantenere nel buffer dei pacchetti per un'eventuale consegna in sequenza al livello superiore. Il mittente ritrasmette solo i pacchetti per cui non ha ricevuto un riscontro. Stavolta c'è una necessità di accumulare pacchetti dal ricevente e si ha una vista differente lato mittente e lato ricevente. Per il mittente la cosa è simile a quella di prima, solo che adesso ci sono pacchetti che non sono stati riscontrati in mezzo a pacchetti che sono stati riscontrati. Per quanto riguarda il mittente, quando scatta un timer ritrasmette solo i pacchetti per cui non ha ricevuto un ACK. Il numero di pacchetti che si possono ricevere senza ACK va concordato col ricevente, che ha comunque un buffer limitato. Di fatto, il mittente deve sapere che può trasmettere N pacchetti, ma non di più, perchè se ne trasmettesse di più rischierebbe di sforare

il buffer del ricevente.
Siccome il range di numeri possibili è limitato, quando un contatore di sequenza fa il wrapping, quando la finestra è grande rispetto al valore di sequenza si possono creare delle ambiguità. Se la finestra è 3, il mittente può trasmettere 0 1 2 e il ricevente fa una cosa simile. Ci sono relazioni tra spazio dei numeri di sequenza e la dimensione della frequenza. Il range è maggiore della lunghezza della trasmissione. Questi problemi si risolvono usando un numero di sequenza con un numero di bit elevato rispetto alla lunghezza della trasmissione.

Reti di Calcolatori - 22/11/2012

Lezione 23. Il protocollo TCP *(Transmission Control Protocol)*

E' un protocollo di trasporto end-to-end e offre a livello sovrastante un servizio che consiste nel riportare un flusso di dati bidirezionale tra i due end-point. I due end-point sono identificati dagli **indirizzi IP** delle due macchine corrispondenti **A e B** e anche dal port number, e quindi esiste il concetto di **port number sorgente** e **port number destinazione**. Ho una **quintupla** come nell'UDP. Mittente e destinazione sono da intendersi sulla base dell'entità che prende l'iniziativa di comunicazione, ma la comunicazione è bidirezionale, quindi non c'è un verso specifico di flusso di dati. Questa connessione trasmette il flusso di byte senza errori e in maniera ordinata, sapendo che il protocollo IP non garantisce l'assenza degli errori. Il flusso è **byteoriented**: l'unità trasmissiva fondamentale è l'ottetto di bit, il byte.

Formato del pacchetto TCP.

C'è un **header** e un **payload**. Il payload contiene dati applicativi e l'header contiene informazioni che servono per mantenere l'***affidabilità di comunicazione***.

L'header è fatto da una parte fissa e da una parte opzionale. La parte fissa è fatta da righe di 32 bit, e quindi occupa 5 righe da 4 byte. La lunghezza minima dell'header è di 20 byte. I primi 2 campi sono i numeri di porta sorgente e destinazione. Gli altri campi sono specifici del protocollo TCP. Inoltre, lo spazio di inirizzamento dei porti TCP è logicamente distinto dallo spazio dei porti UDP: la porta 5000 TCP è diversa dalla porta 5000 UDP.
Abbiamo i seguenti campi:

- **Lunghezza dell'intestazione (*HLEN*)**: è di 4 bit, e contiene un numero intero che è la lunghezza dell'intestazione TCP in parole da 32 bit.
- **Numeri di Porta**: contengono i due numeri di porta **sorgente e destinazione**.
- **Numero di sequenza**: dà la posizione dei dati contenuti nel segmento. Se il pacchetto è stato trasmesso da A a B, contiene dei dati e il primo byte del campo dati è identificato dal numero di sequenza. Questo numero viene incrementato nella trasmissione successiva della lunghezza del segmento dati trasmesso precedentemente.
- **Numero di riscontro**: si riferisce al flusso di dati che viaggia nella direzione opposta. Il riscontro contiene il numero sequenziale (il byte successivo) rispetto a quello di sequenza. Se contiene X vuol dire che tutti i byte fino a X-1 sono stati ricevuti correttamente, e quindi il ricevente si aspetta il byte X. Non si riscontra il pacchetto, ma il byte, e c'è un riscontro cumulativo.
- Ogni pacchetto contiene un valore del numero di riscontro, e la sua significatività dipende dal flag di **ACK**. Se il flag ACK è pari a 1, allora il campo relativo al numero di riscontro diventa significativo, altrimenti non è significativo. I riscontri di ACK possono viaggiare dal destinatario al mittente e non contengono dati, oppure a volte c'è la tecnica del ***piggy-backing***: l'ACK viaggia "*a cavalluccio*" dei dati. Nel momento in cui B trasmette dati ad A, insieme ai dati manda anche l'informazione di riscontro. I dati potrebbero anche non esserci, perchè il pacchetto di riscontro potrebbe non contenere nessun messaggio.
- Nel calcolo del numero di riscontro, oltre a considerare il campo payload, bisogna anche considerare la presenza dei campi **SYN** e **FIN**. Quando entrambi sono alti, il numero di sequenza aumenta di 1.
- **Bit di codice:**
    - **ACK**: serve a indicare il fatto che il numero di riscontro nell'header è valido.

    - **RST**: effettua il reset della connessione.
    - **SYN**: sincronizza i numeri di sequenza.
    - **FIN**: indica che il trasmettitore ha raggiunto la fine del suo stream di byte.
    - **PSH**: se settato a 1, indica che i dati in arrivo non devono essere bufferizzati ma passati subito ai livelli superiori dell'applicazione.
    - **URG**: se settato a 1, indica che nel flusso sono presenti dati urgenti alla posizione (*offset*) indicata dal campo *Urgent pointer,* che punta alla fine dei dati urgenti.
- **Finestra di ricezione**: è un numero intero di 16 bit che indica la dimensione del buffer che l'entità ricevente ha a disposizione per immagazzinare i dati in arrivo. Serve per la gestione dinamica della dimensione della finestra scorrevole (***pipeling***). Si cerca di gestire la memoria in maniera dinamica. I dati, una volta arrivati, devono essere messi nel buffer del ricevitore, e non deve mandarlo in overflow. La receive window non è costante durante la comunicazione, ma varia progressivamente e si adatta alla comunicazione stessa.
- **Puntatore urgente**: contiene un puntatore nel flusso dei dati a una parte di dati che è necessario consegnare urgentemente all'applicazione ricevente. Consente la trasmissione di dati ad alta priorità. Questo campo è reso valido dal fatto che il flag URG è a 1.
- **Checksum**: è un campo di 16 bit che serve alla macchina ricevente per verificare la correttezza dei dati e dell'intestazione. Questo calcolo della checksum viene fatta su un pacchetto in cui viene aggiunta una pseudo-instestazione dove compaiono degli IP di provenienza e di destinazione. La pseudointestazione viene calcolata facendosi passare dall'entità IP che ha ricevuto il datagramma gli indirizzi di mittente e destinatario.
- Le **opzioni** mi consentono di allungare la lunghezza dell'header, e vengono utilizzate per scopi particolari. Quando ci sono le opzioni, la lunghezza dell'intestazione aumenta.Serve un identificatore dell'opzione e una lunghezza dell'opzione stessa. Le opzioni vengono usate per 3 circostanze:

1. **Maximum TCP payload**: Durante la fase di connessione quando le due entità vogliono conoscere la massima dimensione di pacchetti che sono in grado di uscire non frammentati. Questa attività viene detta **MSS *(Maximum Segment Size)***, che rappresenta la massima dimensione del payload TCP, che aggiunta alle due intestazioni rende il pacchetto di dimensione minore o uguale della massima dimensione trasmissibile dalle due parti. Ciascuna delle due parti che si collegano annunciano questo parametro. L'obiettivo è quello di non fare un segmento che poi, passato in IP, deve essere spezzato in più pacchetti. Questo non impedisce che il pacchetto non venga frammentato dai router più interni. Le due parti possono fare un'ulteriore attività di **path MTU discovery**, che serve a capire, sulla sequenza che unisce le entità A e B, qual'è la più piccola MTU, così se tutti i pacchetti vengono inviati con una grandezza minore o uguale di questa MTU, in modo tale da evitare la frammentazione.

Reti di Calcolatori - 23/11/2012
Lezione 24. Il protocollo TCP

Opzioni dell'header:

- **Windows Scale (*Fattore di scala*)**: serve quando abbiamo la possibilità di avere finestre grandi. Se 64 kB non sono sufficienti, occorre moltiplicare questo numero per un fattore di scala. Questo si deve fare su un ***collegamento long-fat pipe***, che serve per indicare connessioni lunghe e grosse di spessore, cioè con una larga banda. Quando abbiamo una comunicazione a larga banda abbiamo un transito sulla linea un numero elevato di byte e se vogliamo utilizzare efficientemente questa connessione dobbiamo trasmettere molti byte

prima di fermarci e aspettare il riscontro. I buffer di trasmittente e ricevitore devono essere grandi.

- **Selective Repeat**: consente di fare un acknowledge selettivo: dare dei riscontri quando nel flusso ricevuto ci si accorge che ci sono dei buchi. Si può richiedere una trasmissione selettiva solo di alcuni byte mancanti, a costo di una maggiore complessità nella gestione, e a patto che questa connessione supporti la trasmissione selettiva. L'affidabilità avviene mediante un meccanismo di ritrasmissione che viene attivato in certe circostanze.

Ciò che da luogo a una ritrasmissione è la scadenza di un **timer**, che viene fermato nel momento in cui si arriva l'ACK dalla controparte. Se invece scatta il timer e scade prima che sia arrivato in tempo utile un riscontro positivo allora parte la ritrasmissione.
Il numero di sequenza è associato al flusso di byte trasmessi, quindi TCP non prevede una strutturazione del colloquio in messaggi, ma offre un **flusso indistinto di byte**, che dovrà essere tagliato per separare un messaggio dal positivo, e questo affettamento del flusso di byte è lasciato al protocollo applicativo. Tipicamente si usano due tecniche: o si mette da qualche parte la lunghezza della stringa oppure una sequenza di caratteri che fanno da tappo. In HTTP ci sono entrambe. Come questi messaggi applicativi vengono spezzati in segmenti dipende dall'azione di una serie di meccanismi che decidono in ogni momento quanti byte è opportuno trasmettere.
La connessione gestita dal TCP è una **connessione full-duplex**, che viaggia in entrambi i versi. Questi due flussi di byte sono indipendenti dal punto di vista dei numeri di sequenza. Il numero di riscontro che A scrive nei propri segmenti è il numero di sequenza del byte successivo che A attende da B, e questo numero di riscontro deve essere inviato in un segmento che ha il bit ACK posto ad 1. Lo stesso messaggio può contenere dati da A a B e anche riscontri.

Telnet consente di aprire una sessione di lavoro testuale su una macchina remota B. L'applicazione, per dare la sensazione che stiamo interagendo con B, fa in modo che i caratteri scritti su A vengono inviati verso l'host B e lui a sua volta ne invia la copia dietro verso A. Tutto questo si realizza con una comunicazione TCP tra A e B.

Se si riceve un riscontro, è del primo byte non ancora ricevuto. Vuol dire che tutti i precedenti sono stati ricevuti correttamente. Questo metodo ha il vantaggio che una perdita di riscontro non blocca la trasmissione se confermata dal segmento successivo.
Quando la destinazione invia un numero di riscontro, questo numero indica quale byte si aspetta di ricevere successivamente: tutti i byte fino a quel numero sono stati ricevuti correttamente. Qual'è il valore più corretto per impostare il timer? Di sicuro deve essere maggiore dell'RTT, perchè si hanno dei timeout prematuri e quindi delle ritrasmissioni ridondanti. Ma l'RTT non è sempre costante, perchè a parità di percorso ci sono componenti fisse e componenti variabili. C'è questo problema di stima del valore corrente di RTT. Questa stima la posso fare se conosco la PDF della variabile aleatoria. In realtà oltre che il **valore medio** occorre anche considerare la **varianza**. Se voglio fare una stima che varia meno rapidamente nel tempo, posso fare una stima del valore medio. **(*slide 17*)**
Si usa una **media esponenziale mobile**. Si ha memoria del valore precedente stimato dell'RTT e si fa in modo tale che l'ultimo valore stimato aggiorni il valore stimato precedentemente ma mantenendo quel valore ancora in memoria. In questa maniera si ha un processo simile a un filtro passa-basso. Con un coefficiente α si considera l'ultimo campione stimato e con un coefficiente $1-\alpha$ si tiene memoria del campione precedente. α lo si prende piccolo, e in questa maniera, ogni volta che si ha una stima, si fa il calcolo dell'RTT. Per settare il timer non solo bisogna considerare il valore medio, ma devo stimare anche la varianza, con lo stesso criterio: il valore di deviazione riscontrato sull'ultimo campione (che è la differenza tra il campione stimato e il precedente) viene pesato con un fattore β e contribuisce alla nuova stima della deviazione ma mantenendo in memoria il valore stimato precedentemente. Tipicamente si usa un fattore β un po' più grande di α. A che valore stimo il timeout? Lo stimo come valore stimato dell'RTT + 4 volte la deviazione (per motivi

di sicurezza).

Come si stabilisce la connessione. (*slide 19*)

Ci sono dei pacchetti preliminari che si scambiano mittente e destinazione per il three-way-handshake. Il mittente invia un segmento di controllo SYN, dove il flag SYN viene messo a 1. Quando il server riceve questo SYN, riceve il segmento come una richiesta di connessione,e quindi risponde con un segmento di riscontro SYN/ACK, dove l'ACK è un riscontro al SYN, e il SYN serve a dire che il server può stabilire la connessione. Nel messaggio di risposta il server indica il primo numero di sequenza della connessione dal server al client. Il primo numero di sequenza dal client al server viene inviato nel primo messaggio di SYN, inviato precedentemente. Questo accade perchè la comunicazione avviene con numeri di sequenza che non partono da 0, ma che partono con un numero di sequenza generato arbitrariamente. Chi inizializza il numero di sequenza è il client. La connessione logica si ritiene stabilita nel momento in cui il client riceve SYN/ACK e invia a sua volta un ACK al server. Al terzo passaggio il numero di sequenza viene aumentato da x a x+1 per convenzione, perché il SIN viene considerato incremento unitario del numero di sequenza, come se contasse un byte.

Scenari patologici del three-way handshake.

Immaginiamo che la connessione che ha fatto la sua vita ed è ancora attiva. All'host 2 poi, arriva un messaggio duplicato di CR (*richiesta di connessione*) con numero di sequenza x. L'host 2 si vede arrivare questa connessione e la prende per buona. Però, l'host 1 è come se non l'avesse mandato questo messaggio, e allora l'host 1, che non si aspetta il messaggio di risposta, si rifiuta con un messaggio di **reject**.
Oppure, se come duplicato arriva un segmento dati con numero di sequenza x e con ACK=z, mentre ACK è y, allora, host 2 non manda il riscontro e alla fine c'è il reject con ACK=y.
Per questo motivo, per evitare situazioni del genere, i numeri di sequenza partono casualmente: è meno probabile che possa arrivare un messaggio duplicato riferito a connessioni stabilite precedentemente. Facendo partire i numeri di sequenza casualmente, si distribuiscono distanti e quindi è più facile che chi riceve questi messaggi capisca che non sono corretti.

Chiusura di una connessione. (*slide 22*)

Un problema analogo si verifica nel momento della chiusura della connessione. La tecnica con cui si risolve al problema è simile alla tecnica TCP: si fa una procedura di **four-way handshake**: un handshake simmetrico. Un client invia un messaggio di volontà di chiusura al server, e si aspetta l'ACK dal server, dopodichè il server stesso invia un messaggio in cui manifesta la volontà di chiusura di connessione e aspetta l'ACK. A volte concretamente si può realizzare con 3 passi se ACK e FIN dal server sono inviate in un unico segmento. Nel momento in cui si chiude la connessione, uno non ha intenzione di inviare altri dati, ma non sa se l'altro ha dati da inviargli. Il client non vuole inviare altri dati, ma, purchè la comunicazione funzioni bene, il client deve ancora poter ricevere dati dal server, e finchè non li ha ricevuti tutti non può andarsene. Quando il server riceve il FIN, risponde con un ACK, ma fa partire un timer perchè non è sicuro di sapere se l'ACK arriva realmente a destinazione. Le entità TCP funzionano comportandosi in maniera diversa a seconda dello stato in cui si trovano: se il client manda il FIN sta in timewait(1). Se riceve l'ACK dal server sta in timewait(2) e quando riceve il FIN sta in timewait(). Poi parte un timer, e quando finisce rilascia le risorse.
Come variante, al passo 2 il server manda ACK e FIN all'interno dello stesso segmento. *DR = disconnection request.* Un caso patologico avviene se si perdono dei messaggi. Anche il server dall'altra parte fa partire un timer, dopo il quale se non riceve l'ACK comunque rilascia la

connessione.
Rilasciare la connessione significa rilasciare tutte le risorse del sistema operativo che servono per gestire la connessione TCP. Il puro fatto di provare a stabilire una connessione TCP può essere usato a scopo di attacco.
Il sistema operativo, infatti, oltre a creare la socket crea una struttura dati con tutte le risorse utili per la connessione. Per fare un attacco DoS si mandano tanti SYN lasciando tante connessioni appese: per questo il sistema operativo ha un limite di connessioni da una stessa macchina.

Per **DoS (*Denial of Service*)** si intente la negazione del servizio. Si tratta di un attacco informatico in cui si cerca di portare il funzionamento di un sistema informatico che fornisce un servizio, ad esempio un sito web, al limite delle prestazioni, lavorando su uno dei parametri d'ingresso, fino a renderlo non più in grado di erogare il servizio.

Diagramma degli stati del TCP. (*slide 29*)
I diagrammi degli stati sono diversificati a seconda del ruolo client/server giocato.

TCP: trasferimento dati affidabile. (*slide 30*)
Gestisce un solo timer di trasmissione alla volta: quando scade viene generato un evento di ritrasmissione. Le ritrasmissioni sono generate da diversi eventi.

Ritrasmissioni.
Gli ACK sono cumulativi. Quando vogliamo fare delle comunicazioni selettive usiamo dei NAK.
TCP usa un solo timer di trasmissione alla volta. Quando scade questo timer viene generato un evento di ritrasmissione. Le ritrasmissioni sono determinate anche da ACK duplicati. Quando ho due segmenti di riscontro con ACK uguali, questa cosa può essere vista come sintomo di perdita.
L'entità TCP deve gestire eventi che gli arrivano dall'entità applicativa che sta sopra, poi ci sono altri eventi che sono: timeout e ACK ricevuti. Ciascuno di questi 3 eventi può essere descritto come un loop nel quale si fanno 3 cose differenti a seconda dell'evento.

Il timer lo possiamo associare al più vecchio segmento non riscontrato. Quando si verifica l'evento timeout viene ritrasmesso il segmento che ha causato il timeout e si riavvia il timer. Se vengono ricevuti degli ACK, si vede se questo ACK riscontra segmenti non riscontrati, aggiorna il numero di sequenza dell'ultimo byte correttamente trasmesso e riavvia il timer se ci sono altri segmenti da completare.

Allo scadere del timeout, si imposta il prossimo intervallo al doppio del valore precedente. In questo modo, gli intervalli di timeout crescono. In questa maniera si ritrasmette anche in maniera meno aggressiva.

Ritrasmissione veloce.
A volte si può anche far scatenare una ritrasmissione prima che scada un timeout per cercare di reagire più rapidamente. Si ha questo comportamento quando, per esempio, arrivano degli ACK duplicati, e vengono inviati indietro quando arrivano dei nuovi segmenti dato, ad esempio. Se arrivano ACK sempre sul segmento precedente, anche senza aspettare un timeout, se ho ACK duplicati devo ritrasmettere. Se ho degli ACK che non avanzano come dovrebbero si prende questa informazione come perdita e si ritrasmette prima delle sequenza del timeout.
Quando mi arrivano un certo numero di ACK duplicati, posso prendere quest'informazione come una necessità di ritrasmissione. Quando un receiver rileva un buco nei segmenti ricevuti, mette i dati nel buffer e comunque manda un riscontro dicendo che deve ricevere un pezzo che ancora non

è arrivato, e siccome vengono inviati segmenti consecutivamente, se ci sono ACK che non avanzano come vorrebbero, si prende l'informazione come segnalazione di perdita. Quando ci sono 3 ACK duplicati, questo è, con buona probabilità, dovuto a un vero e proprio evento di perdita, e quindi si può procedere ad una ritrasmissione senza aspettare il timeout.

In alcuni casi si può anche non inviare subito l'ACK, ma si può ritardare il suo invio fino a 500 ms, fino all'arrivo del prossimo segmento. Oppure si potrebbe inviare un ACK al contrario, per evitare di inviarli inutilmente. Se il ricevente ha dei dati da inviare indietro lo associa ai dati. Altrimenti si può provare comunque a ritardare l'ACK. Tanto gli ACK sono cumulativi, quindi quello che conta è l'ultimo. Allora se si potesse proprio evitare di inviare un ACK si può fare: si aspetta un po' e se nel frattempo arriva un altro segmento lo invio e evito una trasmissione inutile (a meno che non scada il timeout).

Se c'è una macchina di cui conosco solo l'IP posso capire che macchina è vedendo come reagisce ai protocolli standard.

Se arriva un segmento non ordinato viene inviato un ACK duplicato indicando il numero di sequenza del byte atteso. Se c'è un buco e il pacchetto lo lascia a sinistra allora il numero di sequenza dell'ACK non può aumentare. Altrimenti faccio scorrere la finestra di ricezione.

Reti di Calcolatori - 27/11/2012

Lezione 25. TCP: controllo di flusso e di congestione

Il TCP necessariamente deve implementare un meccanismo di **bufferizzazione** dei dati ricevuti, che vanno mantenuti in un buffer e che vanno poi consegnati al livello applicativo superiore. Questa consegna dei dati è possibile solo quando l'applicazione è disponibile a ricevere dati e quando i dati sono stati accumulati in ordine nel buffer. Se sono ricevuti senza ordine, vengono mantenuti nel buffer e si aspetta che i buchi vengano riempiti in seguito a ritrasmissione.
Il meccanismo di controllo di flusso ha il compito di proteggere il ricevitore. Questo meccanismo convive con il meccanismo di controllo di congestione che ha il compito di non proteggere il ricevitore ma la rete: evitare che alcuni router siano essi messi in crisi, e quindi col buffer in overflow, dovuto al fatto che molti flussi stanno attraversando quel router e il router non riesce a smistarli in uscita. Questi due meccanismi (controllo di flusso e controllo di congestione) sono entrambi implementati negli end system e si basano sulla limitazione della trasmissione, del rate, negli end system.
In ogni momento uno solo dei due meccanismi è quello che realmente è efficace, perchè i due agiscono contemporaneamente, non sapendo a priori quale dei due sta agendo effettivamente limitando in maniera più stringente la ricezione dei byte. Agiscono limitando la dimensione della finestra. Quella di ricezione viene inviata dal ricevente al trasmettitore con un messaggio di riscontro, dove c'è un campo dell'header determina quanti byte il ricevitore è in grado di ricevere senza andare in trabocco. Questa finestra annunciata dal ricevitore è dimensionata sulla base dell'azione di due meccanismi.

Controllo di flusso.

Supponendo che il ricevitore scarti segmenti fuori sequenza, può accadere che il buffer si riempie a sinistra, e viene svuotato da destra, attraverso l'azione di receive dell'applicazione. Mano a mano che il buffer si riempie, la coda si riempie, e mano a mano che l'applicazione riceve i dati, il grado di riempimento del buffer diminuisce. **Receive Buffer** è la dimensione del buffer, e possiamo immaginare che si mantenga costante. Invece il **Receive Window** è una quantità dinamica che viene annunciata dall'entità TCP e che è variabile nel tempo. Se le frequenze con cui i dati arrivano e con cui vengono estratti fossero uguali, il buffer non andrebbe mai in overflow, ma queste velocità possono essere differenti. Mediamente, sono possibili piccoli disallineamenti per intervalli di tempo limitati. Il buffer a quello serve. Però, quando si media su un'intervallo lungo, queste due velocità devono coincidere, altrimenti il buffer non ce la farebbe mai a compensare una differenza di velocità persistente. Il controllo di flusso serve a evitare che il trasmettitore invii i dati troppo velocemente, e consumi il ricevitore. Il ricevente comunica la dimensione corrente dello spazio libero del buffer, attraverso il campo **RecWindow**, che viene riempito con *RecBuffer-{LastByteRevd-LastByteRead}*, dove la differenza rappresenta la dimensione dei dati TCP del buffer. Il mittente fa in modo tale da on inviare mai più dati contenuti nella RevWindow, cioè la differenza *LastByteSent – LastByteAcked* (dati che sono in transito nella rete o stanno accumulati nel buffer) deve essere minore o uguale dell'ultima RcvWindow annunciata dal ricevitore.

Esempio.

Il mittente riceve dall'applicazione dei byte da trasmettere attraverso una primitiva di write di una socket che passa 2k byte, che vengono inviati in un segmento dal mittente al ricevente con numero di sequenza pari a 0. Inizialmente il ricevitore ha un buffer vuoto di 4k. I 2k riempiono il buffer a metà. Il ricevitore manda indietro un ACK=2048 (perché i byte da 0 a 2047 sono arrivati correttamente), in cui dice che si aspetta di ricevere il byte 2048 e la sua finestra disponibile è 2048.

Il mittente ha quest'informazione, e manda 3k di dati, che non possono essere inviati al buffer perché avremmo overflow, e allora di questi 3k, solo 2k sono inviati con numero di sequensa 2048. Quando arrivano questi 2k, riempiono completamente il buffer. Allora l'entità TCP manda il riscontro all'entità mittente per dire che si aspetta il byte 4096 e la RecWindow è 0, e in questo modo stoppa il sender. Successivamente l'applicazione del ricevitore legge 2k, e quindi svuota il buffer per metà. Quando si fa spazio libero, l'entità TCP annuncia questa cosa, e manda un messaggio al trasmettittore in cui dice che la sua Recieve Window è 2048. Il trasmettitore prende quest'informazione, e poiché deve inviare 1k lo invia in un altro pacchetto che è grande 1k con numero di sequenza 4096, che si andrà a mettere nel buffer. Questo messaggio blocca il timer del mittente e preserva il ricevente da un buffer overflow.
Questo meccanismo si presta a comportamenti particolari.

Silly Window. (*slide 7*)

Quando l'applicazione mittente o l'applicazione ricevente inviano o leggono pochi byte alla volta, si possono verificare situazioni di utilizzo non efficiente della comunicazione che si cerca di risolvere: questi problemi sono detti di silly window è il fenomeno più essere dovuto a mittente (trasmette pochi dati alla volta) o ricevente.
Quando il problema della comunicazione non efficiente è dovuto al trasmettitore perchè trasmette pochi byte alla volta, allora, se si operasse in questa maniera si avrebbero tanti segmenti dati che viaggiano avanti e indietro, con 40 byte di header e 1 byte di dato. Per evitare questo viene implementato l'**algoritmo di Nagle**, che serve ad avere una trasmissione più efficiente. La prima volta, se la finestra di trasmissione è maggiore o uguale della MaximumSegmentSize, quindi posso inviare più MSS in sequenza e ci sono dati maggiori di MSS, allora viene inviato un pacchetto della dimensione MMS. Se ci sono dati già trasmessi in attesa di ricevere degli ACK nella coda di trasmissione, i dati sono trattenuti nel buffer di trasmissione e non sono trasmessi subito, fino a quando non si riceve un ACK. Se invece non ci sono dati in attesa di riscontro, questi dati vengono inviati immediatamente. Cioè, il primo byte viene inviato, e il mittente lo bufferizza fino a quando non riceve l'ACK. Tutti i dati vengono inviati in un unico segmento, oppure vengono trattenuti finchè non arriva un ACK.

Il problema di silly window si può verificare anche quando è il ricevitore che estrae pochi dati alla volta. Immaginiamo di avere un ricevitore che ha riempito tutto il buffer, e quindi c'è la situazione di buffer pieno, e l’applicazione è lenta (ad esempio, l'applicazione legge un solo byte). Se l'entità TCP mandasse subito un segmento di aggiornamento per 1 byte, forzeremmo la situazione di inefficienza, perchè il segmento ha 1 byte utile e tanti byte di intestazione. Il buffer si riempie e si ha la stessa situazione di prima. Per evitare questo fenomeno, si implementa l'**algoritmo di Clark**: il ricevitore indica finestra nulla finché il buffer non si riempie almeno per metà. Bisogna sempre trovare un compromesso ed evitare che scadano i timeout, perchè bisogna sempre inviare messaggi di riscontro. L'idea è quella di evitare di mandare questo invio alla trasmissione di un solo byte, per evitare la trasmissione di dati utili ma pochi.

Controllo di congestione.

La congestione è dovuta a molti motivi. In presenza di questi fenomeni, si può produrre un sovraccarico della rete, che si manifesta come sovraccarico di uno o più router, e me ne accorgo dal fatto che i link di uno o più router funzionano al 100% e le code vanno in overflow. Quando ho un'entità servente, perchè la coda abbia una lunghezza che non cresca a dismisura occorra che il servente lavori in una frazione di tempo minore dell'unità. Il rapporto tra il tasso di utilizzo e il

valore globale deve essere all'80% circa. Finchè il tasso di utilizzo, che è il rapporto tra tempo in cui l'operatore lavora e il valore globale è minore di 1 allora si lavora bene. Quanto più il valore è prossimo a 1, tanto più la coda aumenta. Questa teoria si può usare per modellare la coda del router. Il router serve ogni pacchetto e serve un certo tempo per trasmettere il pacchetto. Tutti i pacchetti che arrivano, mentre il servente è impegnato, sono memorizzati in un buffer interno del router. La coda è finita, quindi **sistema** è **con perdita**: oltre un certo limite i pacchetti sono persi. In generale bisogna sempre evitare che un'entità che gestisce dei clienti abbia un tasso di utilizzo che si approssima all'unità.
(*slide 5*) Si procede fino a quando non ci sono situazioni si **congestione**, cioè finchè arrivano tot pacchetti al secondo, escono tot pacchetti al secondo, e il tempo di attraversamento del pacchetto tende ad aumentare sempre di più. Quando si arriva ad un certo punto, la lunghezza della coda diverge, e anche il tempo di ritardo di attraversamento dei pacchetti diverge. C'è R/2 perchè è il caso in cui ci sono 2 flussi indipendenti che attraversano tutti e 2 lo stesso router, e se il link di entrata e il link di uscita hanno una capacità trasmissiva di R, fintantochè i due flussi di uguale velocità hanno un tasso inferiore a R/2, questi due flussi riescono a passare attraverso il router, ma quando i due flussi superano la velocità di R/2, succede che questo link viene usato al 100% del suo utilizzo e quindi si ha la situazione di crisi, che fa sì che i pacchetti arrivino con tempi sempre maggiori. E stiamo immaginando il buffer di capacità infinita, e quindi senza perdite. Se ci fossero perdite, la situazione peggiora, perchè in TCP c'è la ritrasmissione.
Quando il buffer del router ha capacità finita, la situazione peggiora, quindi bisogna accontentarsi di un valore di **throughtput** inferiore perché la congestione non è migliorata da una dimensione potenzialmente infinita. Se ci sono anche ritrasmissioni la situazione peggiora ancora di più.
Bisogna assolutamente evitare che parti della rete si trovino in congestione, perchè quando si verifica, di fatto tutte le performance osservate dai singoli flussi degradano in maniera significativa, e allora bisogna proteggere la rete da queste situazioni, rallentando le sorgenti di traffico: mettere un rubinetto controllato dalla rete stessa che faccia uscire meno flusso dalle sorgenti, che rallenti la velocità con le quale il flusso di byte viene iniettato nella rete. Questo è l'obiettivo delle tecniche di controllo della congestione.

Nelle reti a pacchetto sono disponibili due approcci. Fermo restando che l'entità ultima su cui agire è la sorgente, chi è che dice alla sorgente di rallentare?

1. Può essere il ricevitore che, in qualche maniera, si accorge che c'è un problema sulla rete, lo segnala, e il mittente rallenta sulla base della segnalazione diretta o indiretta, trasmettendo meno byte.
2. Richiede il coinvolgimento esplicito, attivo della rete: i router della rete si accorgono di stare andando verso la congestione, trovano le sorgenti che buttano più traffico e avvisano questi mittenti di rallentare. Questo avviene attraverso l'invio di un bit in alcuni protocolli che specificano il fatto che c'è congestione. Un meccanismo di questo tipo è anche implementato in alcune entità TCP che specificano l'**ECN (*Explicit Congestion Notification*)**.
   L'approccio direttamente implementato nel TCP è tale che il mittente si accorga della congestione (attraverso un analisi di ritardi e perdita di pacchetti).

Con il primo approccio, l'entità mittente capisce della congestione attraverso un'informazione data dal ricevitore. Sono meccanismi di controllo,e prevedono un feedback: un loop di reazione. Il problema è capire chi da questo feedback, rete o ricevitore.
In TCP, il feedback agisce sulla RecWindow. Si fa in modo tale che il ricevitore in ogni momento annuncia un valore di RecWindow che è il minimo tra la Receive Window implementata dal controllo flusso e una Congestion Window calcolata con degli algoritmi particolari.

Esempio. **(*slide 9*)**

Il meccanismo di controllo di congestione limita la differenza tra ultimo byte trasmesso e ultimo

byte per cui si è ricevuto un riscontro.

Quanto deve essere grande la Congestion Window? Inizialmente non si ha idea se c'è o meno una congestione nella rete. L'approccio utilizzato è quello di trasmettere alla massima velocità possibile senza che si verifichino perdite. Se non ho informazioni che facciano presupporre una perdita nella rete, non vale la pena limitare la velocità e quindi si usa la massima velocità possibile. Il meccanismo di congestione non deve limitare il traffico nella rete. Non devo limitare il traffico, se il ricevutore non mi limita col meccanismo di flusso. Non devo però cominciare a trasmettere con una quantità di byte estremamente elevata, ma devo essere prudente, perchè posso causarla io la congestione. L'idea è quella di arrivare al livello ottimo con degli aggiustamenti successivi, sapendo che il valore ottimo non è una quantità statica, anche perchè ogni sorgente è parzialmente responsabile del fenomeno di congestione. Questo meccanismo funziona solo se tutte le sorgenti lavorano nello stesso modo. Se un router non rallenta in congestione, lavora bene ma rallenta tutti gli altri. Si raggiunge un obiettivo di **fairness**: *equità*. Tutti quanti vengono premiati o penalizzati alla stessa maniera. L'approccio usato è quello di partire da un certo valore di Congestion Window, e provare ad aumentarla fino a quando non si rileva la perdita di un segmento. Quando c'è un evento di perdita, che può essere rilevato da un timeout, questo può indicare la congestione. Allora, si trasmette una quantità di byte minore, diminuendo la Congestion Window, e si ricomincia da capo. I meccanismi di aggiustamento sono 2 differenti, e agiscono in 2 momenti differenti della congestione.

Fasi del controllo della congestione

Ogni volta che c'è una perdita, si parte lentamente, cioè da un valore basso di Congestion Window, che si cerca di far salire il più velocemente possibile. Quando ci si accorge di essere arrivati ad una stima realistica, si ragiona con un comportamento **AIMD (*Additive Increase, Multiplicative Decrease*, *incremento additivo*, *decremento moltiplicativo*)**: vai piano piano a salire, e quando ti sei accorto che hai superato giusto, per cui stai trasmettendo troppo, allora abbassa velocemente. Si genera un **andamento** nel tempo **a dente di sega**: si cresce linearmente e poi si abbassa rapidamente. Si parte da un **algoritmo di slow start**. Si parte da 1 MSS (si trasmette in un segmento con bassa quantità di dati) e si aspetta un riscontro. Un riscontro positivo arriva dopo un RTT. Quando arriva il riscontro, per ogni segmento per il quale si riceve il riscontro (ACK) si aumenta la Congestion Window di 1 MSS. Si inviano ora quindi 2 segmenti, e si aspettano i 2 ACK. Ogni ACK mi porta a trasmettere un altro segmento, e quindi al terzo tentativo i segmenti diventano 4, poi 8, ecc.. La velocità aumenta velocemente ma si parte da un valore basso. Si parte lentamente, ma si cerca di recuperare rapidamente. Questa fase termina ad un certo punto.

Per TCP, gli **eventi di perdita** sono: o ***timeout*** (niente riscontro) o può accadere che mi arrivano 3 ***ACK duplicati consecutivi***: invio un segmento e poi o si è perso un pacchetto, o c'è stata una perdita non grave, quindi c'è una situazione di "quasi" congestione. Chiaramente sto immaginando che non sia scaduto un timeout. In un certo momento, questa fase slowstart viene abbandonata, in quanto fa salire troppo velocemente la Congestion Window, per cui se continuo a trasmettere così molto presto vado in congestione. In un certo momento, al raggiungimento di un **valore di soglia THR (*Threshold*)**, si abbandona questa fase e si entra nella fase di AIMD. In questa fase, ogni volta che arriva un ACK, viene incrementata la Congestion Window di una quantità pari a $\frac{MSS * MSS}{Congestion\ Window}$. In questa fase, detta di **congestion avoidance**, l'andamento nel tempo è lineare, e si cerca di evitare la congestione. Il mittente aumenta il rate, ma è pronto a riabbassarlo se c'è congestione. Quando si verifica un evento di perdita, la finestra di congestione viene dimezzata *(multiplicative decrease)*, e si prosegue di nuovo linearmente. Per questo si ha una situazione simile ad un dente di sega. **(*slide 14*)**

Ci sono 2 implementazioni TCP che si comportano in maniera differente quando abbiamo 3 ACK duplicati:

- **TAHOE**: riparte con lo slow start, la crescita è esponenziale a partire dal valore 1.
- **RENO**: mette come valore di finestra di congestione la metà del precedente, e questo valore è impostato anche come nuovo valore di soglia.

Supponiamo che la perdita sia verificata al valore corrente della finestra di congestione 12. Con il TCP Reno viene impostato come nuovo valore di congestione la metà di 12, e questo valore viene impostato anche come nuovo valore della soglia. In generale, quando c'è l'evento di timeout comunque si riparte da 1.

Fast Recovery.

Quando ci sono vari flussi in competizione, chi è più aggressivo tende ad avere la meglio. Chi è meno democratico sono i flussi UDP e molti ISP li limitano a prescindere con dei filtri nella rete per rallentarne il rate. Per TCP l'approccio RENO è più aggressivo perché tende a recuperare subito la connessione alta, il TAHOE no. Nei flussi multimediali, ritrasmettere i pacchetti non è una cosa furba.

Fearness. (*slide 18*)

È un meccanismo di ripartizione. Il throughtput che si riesce ad ottenere, dipende anche dall'RRT. Se la distanza è grande, il flusso è più svantaggiato, perchè adatta la sua finestra più lentamente. Per cui in una situazione di competizione sono avvantaggiati i flussi con RTT più piccolo. Un altro parametro che gioca a favore è l'MSS: una connessione con un MSS più grande ha un throughtput più grande. A parità di MSS e RTT mediamente trasmettono allo stesso modo mentre in realtà a volte trasmette più uno e a volte più un altro. La banda non si ripartisce mai ugualmente a metà, ma il certi momenti trasmette di più, in altri momenti di meno, e si ha un equilibrio di throutput pari a R/2. Dipende anche da chi parte prima e chi parte dopo.

[Applet: A Tour Around TCP – The Sliding-Window Flow-Control Algorithm]

Reti di Calcolatori - 29/11/2012

Lezione 26. Simulation of Computer Networks

Possiamo costruire un prototipo su piccola scala del testbed da valutare, che replica il sistema reale. Cioè si crea una piccola rete prototipale all'interno di un laboratorio. Più su larga scala si usano dei nodi che si usano per esperimenti, costruiti dall'Università di Princeton (PlanetLab). Si può emulare il sistema, introducendo anche elementi non reali simulati da sistema software: ho programmi che interfacciano elementi reali. L'emulazione consiste in elementi reali e elementi simulati: il tempo è ancora quello reale. Se ci spostiamo ancora verso elementi non reali abbiamo la simulazione: si basa tutto su software che lavorano su modelli matematici: non ci sono elementi reali. Possiamo anche costruire dei simulatori ad hoc invece di usare quelli già disponibili: sono dei programmi scritti ad hoc per simulare il sistema. Infine, si può lavorare direttamente col modello matematico cui si da un input per ricevere un determinato output in base al sistema da testare.

Per la simulazione abbiamo un parallelo tra sistema reale e simulato. Ogni sistema reale ha delle regole che ne definiscono il comportamento. Il sistema reale riceve input dall'ambiente esterno e da determinati output all'osservatore. Il contorno del sistema reale è l'ambiente stesso. Il simulatore è un programma che parte da degli input (dati proprio dal calcolatore, genericamente sono numeri casuali). In questo caso il sistema under test è quello che gira nel computer. Non si misura il sistema reale direttamente perché o è troppo costoso o non ce l'abbiamo ancora perché siamo in fase di progettazione. Cambiando un parametro nel simulatore comporta un istantaneo cambiamento nel comportamento, cambiare un sistema reale è molto più complesso. Se non esiste un modello matematico in forma chiusa col simulatore possiamo dividere il sistema in più sottosistemi e approssimare una soluzione. Infine, il sistema reale è fatto in modo da avere determinate variabili indipendenti dalle altre, si fa variare esattamente quello che vogliamo: abbiamo un controllo completo sugli input del sistema.

Il simulatore astrae il sistema: il suo difetto è che è un'approssimazione. Può essere vicino al sistema reale in alcuni casi, in altri no. Dobbiamo capire quando può essere usato. E' in grado di generare topologie e scenari complessi: cioè può generare una rete complessa secondo parametri che definiamo noi. Deve essere estendibile e modulare, perché quando usiamo un simulatore possiamo voler provare un nuovo protocollo. Dobbiamo quindi estendere il simulatore (non è scontato, l'ns-2 simulava tutti i protocolli esistenti ma era complicato aggiungerne altri perché bisognava agire sul core del sistema). Deve funzionare anche l'emulatore: il simulatore deve interfacciarsi anche con un sistema reale applicando il modello con cui è stato configurato.

Il simulatore consiste di tre elementi:

- **Tempo simulato**: variabile globale che mantiene il tempo della simulazione. Varia in maniera discreta.
- **Stato**: ci saranno componenti, variabili (una che rappresenta una coda, il numero di pacchetti...)
- **Eventi**: il simulatore va avanti simulando degli eventi che provocano una modifica dello stato e degli output in uscita, come gli automi a stati finiti.

Il simulatore mantiene una lista di eventi e fa quest'algoritmo: esegue degli eventi in successione, da quello con un tempo minore a quello con un tempo maggiore. Gli eventi modificano uno stato del sistema e creano nuovi eventi secondo un principio causale (gli eventi sono creati nel futuro, ovviamente). Se un evento è definito al tempo 10 vuol dire che sono arrivato a misurare al tempo 10.

Quando prendo un evento ci sarà una procedura nel simulatore che lo identifica e si comporta di conseguenza. Ad ogni evento è associata una struttura che è tipicamente un evento. Gli eventi sono processati da uno scheduler. Questo è uno strumento del simulatore che schedula gli eventi uno dopo l'altro. Gli eventi sono processati attraverso una coda o heap. Il tempo di simulazione non equivale al tempo reale: a seconda del numero di eventi il tempo cambia. Il tempo di simulazione dipende dalla densità degli eventi: tipicamente è elevato, e quindi si cerca di fare in modo che la simulazione si concentri sui parametri di interesse.

Esempio all'arrivo di un pacchetto.

Abbiamo un router da cui arrivano certi pacchetti con una frequenza λ. Questi vengono messi in due code e dopo un tempo μ_1 e μ_2 vengono mandati fuori. In questo caso, lo stato è la lunghezza delle due code (gli altri parametri sono tutti fissi, può variare solo questo).
Gli eventi sono: l'arrivo dei pacchetti e il loro invio.

Il simulatore agisce sull'arrivo di un pacchetto: quando arriva si decide quale coda scegliere. Poi si valuta lo stato della coda, a seconda della quale il comportamento è diverso: se il link è libero lo elabora altrimenti lo mette in coda. In questo caso cambio anche lo stato. Invece se il link è pieno incremento una variabile (dropped packets) e ignoro l'arrivo. Se non è né pieno né vuoto incremento il numero di pacchetti nella coda. Infine, a prescindere dallo stato della coda, metto nella lista degli eventi il prossimo evento. In base al valore estratto, secondo una certa distribuzione di probabilità, installo l'arrivo di un nuovo evento.

Esempio alla partenza di un pacchetto.

Rimuovo l'evento, aggiorno il tempo di simulazione e le statistiche. Modifico il numero dei pacchetti in coda e faccio in modo da creare un altro evento che lavori anche sul pacchetto successivo. Si processano gli eventi finché non sono finiti tutti gli eventi in lista.

Tutorial Ns-3.

Ns-3 cerca di aderire il più possibile agli standard di fatto dei sistemi reali. In output abbamo dei file .pk, dei binari usati anche nei sistemi reali. Le simulazioni sono in C++ o Python. Per installarlo oltre al configure si può usare anche il `./waf`. Le documentazioni sono in Doxygen.

Configure -> `./waf -d [optimized|debug] configure`
make -> `./waf`
make test -> `./waf check (run unit tests)`

Per far girare il programma si usa sempre `./waf`.

E' organizzato secondo una rete reale: si usano le applicazioni e c'è un'interfaccia simile alle socket tra applicazione e stack protocollare. Poi ci sono i nodi tra stack protocollari e NetDevice e per l'interfacciamento tra questi due ci sono i Packet Socket (interfacce a basso livello per non usare il TCP). Gli oggetti principali sono: nodi, pacchetti e canali.

Prima si genera il canale (perché l'interfaccia dipende dal canale) e poi il resto. C'è una stretta relazione tra interfaccia e canali. Si mantengono due astrazioni: Socket e Linux Packet Socket tra applicazione e stack e tra stack e net devices.

I pacchetti sono costituiti da un buffer, tag e metadati. Il buffer è una stringa di bit che descrive il

contenuto del pacchetto. Poi abbiamo dei tag, che sono informazioni addizionali collegate al pacchetto. Un modulo TCP aggiunge un tag e poi va a leggere l'IP: il tag ha una visibilità locale al nodo, quando lo invio si legge solo il buffer. I metadati sono creati da ns3 e sono dati aggiunti per il controllo (sono opzionali, controllano che non si facciano errori sugli header). E' un check che però rallenta la simulazione. All'interno del buffer dei pacchetti abbiamo anche gli header (rappresentati dalla classe Header, usata per scrivere o leggere dal buffer). Questo si fa con due operazioni dette Serialize (si legge dal pacchetto e si scrive nell'oggetto header) e Deserialize (per leggere l'header dal pacchetto).

Per iniziare e terminare una simulazione si chiama `Simulation::Run()` e si usa `Simulation::Stop()` se si vuole fermare la simulazione ad un dato istante (tipicamente finisce da sola all'esaurirsi degli eventi senza chiamare la stop).

La memoria può essere gestita in maniera semiautomatica. Quando la memoria in un dato oggetto è a zero questa viene deallocata dinamicamente. Invece di partire da una classe base con una gerarchia si usa l'aggregazione: aggiungo alla classe base altre funzioni in maniera dinamica. Cioè istanzio un oggetto base e a runtime aggiungo funzionalità e altri oggetti. Non faccio subclassing ma aggregazioni di classi in classi.

C'è un sistema di attributi che consente di accedere a delle proprietà della simulazione e/o settarle.

Reti di Calcolatori - 04/12/2012

Lezione 27. Il livello datalink della rete

Il livello 2 ha come obiettivo quello di realizzare una trasmissione di pacchetti tra un'interfaccia di un dispositivo ad un'interfaccia di un altro dispositivo. I dispositivi possono essere sia end-system che router.
Internet è fatto dal concatenarsi di tanti collegamenti fisici: nel momento in cui si lavora in questa logica, a livello data link appaiono una serie di **tecnologie differenti**. Mentre dal livello rete in su le tecnologie sono nascoste, invece a livello data link vediamo tutte le caratteristiche specifiche di una tecnologia. Nessuna caratteristica di IP (al massimo l'MPU) è caratteristica della tecnologia. Ci sono tecnologie di rete locale e tecnologie che consentono di collegare router lontani tra di loro su scala geografica. La tecnologia determina le modalità dei trasmissione dei bit sul supporto fisico. C'è una caratteristica comune come problematica a quella del collegamento fisico: come si partiziona la capacità trasmissiva tra le varie stazioni? Servono protocolli ad accesso multiplo. E' un problema con soluzioni differenti a livello LAN e su scala geografica.
Da un punto di vista dello stack di protocolli, il livello datalink è il livello più basso in cui si vede ancora il pacchetto, perchè nel livello fisico c'è proprio uno stream di bit. Il pacchetto a questo livello viene chiamato **frame**, e contiene al suo interno il pacchetto di livello rete, a cui è anteposto un header, e spesso c'è un'informazione di controllo aggiunta in testa ma anche in coda (*trailer*). Il livello datalink è il livello più alto implementato nel livello hardware delle schede di rete, mentre le funzionalità del livello rete sono implementate nel sistema operativo del dispositivo, quindi in software.

Funzioni del livello datalink.

Costruisce il pacchetto (***framing***), il che consiste nell'aggiungere campi di intestazione, header e trailer, gestisce l'accesso al canale in caso di mezzo condiviso e gestisce il problema dell'indirizzamento, in quanto viene fatto utilizzo di indirizzi fisici all'interno delle frame per identificare protocollo e destinazione. Gli **indirizzi fisici** sono diversi dagli indirizzi di rete e sono indirizzi **di 48 bit** che identificano l'hardware di rete e vengono cablati nella scheda di rete del router. Ci si pone come far viaggiare l'informazione in maniera sicura nel canale, anche se meccanismi di recupero di informazione non sono implementati a livello datalink, però occorre accorgesi quando una frame arriva corrotta. In alcuni casi si possono implementare altri meccanismi come ridondanze, ma questa tecnica non si usa. Il controllo che si fa è il **controllo di flusso**: viene implementato non per l'intero percorso, ma per la singola tratta (ad esempio, tra computer e router che sta ricevendo il pacchetto), e serve a regolare il flusso tra mittente e destinatario.Si deve cercare di evitare duplicazioni di funzionalità tra i vari strati, quindi tipicamente le funzionalità di controllo sono implementate nella maniera più completa possibile a livello trasporto.

Scheda di rete.

L'**adattatore di rete** è la scheda di rete implementata nei computer, e consente al computer di inviare o ricevere dei pacchetti che sono passati dal sistema operativo in dei buffer e inviarli sul collegamento fisico, e in ricezione viene ricevuto il pacchetto e generato un interrupt per prelevare il messaggio dalla scheda e metterlo in un buffer del sistema operativo. Queste schede sono sempre più intelligenti: le funzionalità sono implementate in maniera tale da scaricare quanto più possibile la CPU da compiti di elaborazione. Si mettono più risorse possibili nella scheda di rete (buffer sufficientemente grandi da poter accumulare un certo numero di pacchetti), per generare quante meno interruzioni possibili.

Se per costruire il pacchetto occorre calcolare il registro checksum, lo si calcola sulla scheda di rete direttamente, invece di usare la CPU del computer. Se la trasmissione del segnale necessita di una conversione in una sequenza di bit lo si fa con un **DSP (*Digital Signal Processor*, *processore di segnale digitale*)** all'interno della scheda. Sempre più sono i dispositivi I/O intelligenti: hanno capacità di DMA (accesso diretto alla memoria: possono prelevare direttamente i dati dalla RAM senza scomodare la CPU) e le capacità di elaborazione sono supportate da un processore che sta a bordo della scheda. E' un minicalcolatore dedicato che comunica e interagisce con la CPU principale del calcolatore.

Rilevazione degli errori.

Per la rilevazione degli errori si aggiungono dei bit di controllo, calcolati con una certa funzione (tipo parità), che vengono aggiunti ai bit di dato. Alla base di questi schemi vale l'ipotesi che se è piccola la probabilità di errore su un solo bit, la probabilità di errore su più bit è ancora minore. Ma, su molti canali trasmissivi, gli errori si propagano a treni: se un bit è errato, quelli successivi è molto probabile che siano errati.
Con il **CRC (Cyclic Redundancy Check**, *controllo a ridondanza ciclica***)** le sequenze di bit devono essere interpretabili come il coefficiente di un polinomio. Il calcolo della CRC viene fatta con la divisione per polinomi. Se ottengo un resto che non è zero ottengo l'indicazione che ci sia stato un errore. Queste operazioni vengono fatte direttamente dall'ALU della scheda di rete.

Link di rete.

Ci sono due tipi di data-link:

- **Link punto-punto**: per collegare dispositivi in maniera esclusiva,
- **Link broadcast**: per collegare un numero n di stazioni e farle parlare tra di loro secondo l'occorrenza.

I collegamenti punto-punto vengono utilizzati anche in ambito locale, per far comunicare due computer con cavi seriali mediante le opportune interfacce, ma non sono proprio tecnologie di rete. Queste tecnologie hanno senso se sono organizzate in maniera geografica. E' un ottimo modo per collegare router lontani tra di loro. Su queste tecnologie per realizzare la comunicazione, occorrono dei protocolli che servono a dire come una stazione deve costruire le frame e quali informazioni di controllo devono essere inserite in queste frame. In questi tipi di collegamenti viene meno la necessità dell'indirizzamento: se c'ho un collegamento solo tra A e B, tutto quello che parte da A arriva a B necessariamente. Quando invece abbiamo un mezzo condiviso, cioè un canale che consente di comunicare due stazioni qualsiasi tra n, nasce un **problema di indirizzamento**.
Tecnologie a mezzo condiviso possono essere utilizzate per creare collegamenti su scala geografica o LAN (tecnologie wireless e cablate), ma più frequentemente vengono utilizzate per creare reti locali. Quando ho tante stazioni che possono trasmettere, come faccio a dire quando una stazione può trasmettere e cosa succede se due stazioni trasmettono contemporaneamente? Per questo nascono i **protocolli di trasmissione ad accesso multiplo**. Quando le stazioni trasmettono contemporaneamente, c'è un conflitto tra le stazioni e l'informazione viene danneggiata e nessuna delle stazioni riesce a trasmettere i dati che voleva trasmettere. Occorre risolvere il problema del partizionamento del canale di trasmissione tra i vari trasmettitori. Si può, ad esempio, agire sul tempo: una sola stazione alla volta può trasmettere alla volta, e bisogna decidere quale stazione è destinata a trasmettere e quale no. Quando c'è questo tipo di approccio, c'è un problema di coordinamento tra le trasmissioni.
Gli schemi fondamentali sono 3:

- **Channel Partitioning**: schemi che suddividono la capacità trasmissiva del canale in porzioni più piccole, su cui non esistono contese. Questo partizionamento assegna ad ogni

stazione la porzione della capacità trasmissiva globale. La porzione di capacità trasmissiva è assegnata in maniera esclusiva ad una stazione. Questo partizionamento porta a dei problemi.

- **Random Access**: Schemi che partizionano dinamicamente il link sulla base delle esigenze correnti. Ma, generando una contesa nella capacità trasmissiva, incorrono nella possibilità di trasmissioni simultanee che il channel partitioning evita. C'è un problema: non si impediscono trasmissioni simultanee. Gli schemi ad accesso casuale sono più efficienti ma devono risolvere il problema del controllo delle collisioni-
- **Taking turns**: Lo schema del partizionamento è definito dinamicamente, ma evita la possibilità che si generino queste collisioni attaverso un meccanismo di coordinamento, che dice chi è abilitato a trasmettere e chi no. La divisione è sempre dinamica.

Lo schema ideale è quello che assegna la capacità trasmissiva (il canale) alla stazione che deve trasmettere se è l'unica che deve trasmettere, e, invece, se tutte le stazioni vogliono trasmettere, deve essere equo, perchè la capacità trasmissiva deve essere divisa equamente tra le varie stazioni.

Protocolli per il partizionamento del canale.

La tecnica **TDMA (*Time Division Multiple Access*)** fa il partizionamento sull'asse dei tempi: il tempo viene suddiviso ciclicamente in slot, ciascuno dei quali è assegnato univocamente ad una stazione. E' una situazione ripetuta per ogni stazione. Se ho 4 stazioni, ho 4 slot, e in ogni slot di tempo può trasmettere una sola stazione. Questo schema prevede una sincronizzazione temporale tra le varie stazioni, perché le stazioni si devono sincronizzare precisamente su un unico blocco temporale comune, e per questo si usa un clock comune tra le quattro stazioni, che devono essere sincronizzate sull'unico clock. Si usa per le linee telefoniche: su un unico supporto trasmissivo viaggiano più telefonate. C'è un problema di efficienza: ci possono essere parti del canale che non vengono utilizzate perchè in quel quanto di tempo il canale non viene utilizzato dalla stazione che lo dovrebbe utilizzare. Una stazione deve sempre aspettare il proprio turno per trasmettere anche se in quel momento il canale non è usato dal nessuno. La rigidità del partizionamento è anche uno svantaggio perché porta inefficienza.

La tecnica **FDMA (*Frequency Division Multiple Access*)** divide la capacità trasmissiva sulle bande di frequenza: il canale trasmissivo ha una banda passante, divisa in sottobande e ciascuna banda trasmissiva è usata per far viaggiare il segnale in una coppia trasmettitore-ricevitore. Su ogni portante c'è il segnale che porta i dati (con una modulazione). Cambia solo la tecnica implementativa, ma non cambia il concetto: partizionamento statico della risorsa. C'è lo stesso vantaggio (niente contesa) e lo stesso svantaggio (banda inutilizzata) del TDMA, basato sul partizionamento statico.

La tecnica **CDMA (*Code Division Multiple Access*)**, invece, utilizzata soprattutto per la comunicazione cellulare, non partiziona il canale trasmissivo né nell'asse dei tempo né nell'asse delle frequenze. Sta per multiplazione a divisione di codice. Il canale non è partizionato, tutte le stazioni trasmettono in banda base generando sequenze di bit a partire dai dati. Viene trasmesso il prodotto tra il bit dato e la sequenza base. Quando ho 1, viene trasmessa la sequenza. Quando è -1 ho la sequenza ribaltata. La sequenza base del codice è specifica di un mittente. Il ricevente condivide col mittente la conoscenza a priori della sequenza base (il ricevente conosce la sequenza base su cui sta trasmettendo il mittente). Il mittente moltiplica la trasmissione per la sequenza base.

Se ho due mittenti questi devono usare due codici differenti. Le sequenze dato devono essere ortogonali e le sequenze così trasmesse indipendentemente dai due mittenti si compongono in

maniera additiva. I due mittenti per poter lavorare direttamente devono essere sincronizzati. Ci deve essere un sincronismo di clock. Quando si sovrappone la sequenza generata dal primo e dal secondo mittente si ottiene un segnale sommato. Se il ricevitore fa la stessa operazione per tutti e due i mittenti riesce a isolare le informazioni trasmesse dal primo e dal secondo mittente. Applicando sulla sequenza ricevuta lo stesso algoritmo di prima il ricevitore riesce ad estrarre gli stessi dati generati in origine.
Questo funziona se le sequenze codice sono ortogonali, il canale è additivo, c'è sincronismo tra i mittenti e il ricevente conosce le sequenze base.

Protocolli ad accesso casuale.

Tutte queste tecniche consentono di condividere la capacità trasmissiva del canale secondo uno schema prefissato, e quindi, quando viene dato un permesso di trasmissione ad un nodo, esso può usare il canale in tutta la sua capacità. Di contro, sono inefficienti perché potrebbe esserci spreco di canale. Le tecniche a random access decidono dinamicamente una divisione, per non avere sprechi. Non esiste nessun coordinamento, e quindi può accadere che si generino delle collisioni: trasmissioni simultanee tra due o più nodi. E' un prezzo da pagare: la collisione è distruttiva, ma si prevedono dei meccanismi di recupero per rimediare a queste collisioni. Il primo problema è capire quando si generano queste collisioni e poi bisogna decidere come eliminarle.

Ci sono 3 tecniche:

- **Slotted Aloha.**

C'è una sicnronizzazione temporale tra le stazioni, e immaginando che tutti i pacchetti abbiano la stessa lunghezza in termini di bit, vengono trasmessi in slot tutti di uguale durata. Uno slot dura quanto la lunghezza del pacchetto / la velocità di trasmissione. Il nodo ha dei dati e trasmette, ma per fare questo si possono generare delle collisioni. In caso di collisione, il nodo si accorge della collisione, e allora ritrasmette negli slot successivi con probabilità p, finchè la trasmissione non va a buon fine. Non c'è coordinamento tra le stazioni, ma l'unica forma di coordinamento è quella temporale, cioè tutte condividono la stessa divisione del tempo a slot. Quando si genera la collisione bisogna aspettare la fine dello slot, ma è andata male. La probabilità che si verifichino collisioni a mano a mano che aumentano i nodi che devono trasmettere pacchetti.
Dopo la collisione tra più pacchetti, allo slot successivo “si lancia la moneta”, per vedere se bisogna ritrasmettere o no. Supponiamo ci sono tre collisioni: lancio la moneta, nessuno trasmette, spreco lo slot. Lancio la moneta, due trasmettono, collisione. Lancio la moneta, trasmette uno. Lancio la moneta per i rimanenti, non trasmette nessuno. Lancio la moneta, collisione. Lancio ancora la moneta e finalmente trasmette solo uno. Poi trasmette l'altro.
Supponiamo di avere N stazioni che devono trasmettere, ognuna trasmette con probabilità p.
La probabilità s che una trasmissione abbia successo, cioè che solo uno degli N nodi trasmetta, è $S = p(1-p)^{N-1}$ p(1-p)^(N-1). Siccome ci sono N nodi, la probabilità che sono un nodo trasmetta è N volte la funzione. . Il valore ottimo di s per n che tende all'infinito si trova calcolando il massimo di questa funzione al variare di p. Al limite si ottiene il 37%.

- **Aloha puro.**

Non richiede sincronizzazione. La durata dello slot è nota ad ogni stazione. Quando c'è stata una collisione, sugli slot successivi (basati sul suo clock, non sincronizzati) decide se trasmettere o meno sempre con modalità p. La probabilità di collisione raddoppia, c'è una sovrapposizione possibile di frame che è il doppio di quella di prima..La funzione è la metà della precedente, e ha un esponente diverso e il suo massimo cel’ha prima: anche il suo valore massimo è più basso e corrisponde al 18%.

Il goodput è la frazione di slot che si riesce a trasmettere con successo, è la frazione di capacità trasmissiva che si riesce ad utilizzare efficacemente. Il canale è usato efficacemente solo per una frazione del tempo complessivo, perchè per il resto del tempo il canale o non è usato o è c'è collisione e quindi non viene utilizzato.

Per **rilevare un conflitto**, il nodo confronta il segnale che sta inviando al canale e il segnale che gli ritorna dal canale e, inoltre, il trasmettitore ha attive un'entità trasmissiva e una ricevente. Fintantochè non ci stanno altri segnali sovrapposti, ciò che trasmette e ciò che riceve sono uguali, a meno di un rumore e un'attenuazione in una fascia di tolleranza. Quando c'è una collisione, il segnale ricevuto è diverso dal segnale trasmesso. Si usa un'amplificatore differenziale, in cui si fanno entrare due componenti frequenziali, e se il segnale differenza è in una soglia, la differenza è solo rumore, mentre se il segnale differenza ha una potenza sufficientemente maggiore di 0, allora c'è collisione. Nelle reti wireless è più complicato.

- **CSMA (Carrier Sense Multiple Access)**

Risolve i problemi dell'ALOHA: le due aloha hanno una bassa efficienza perchè le stazioni trasmettono senza controllare il canale. L'idea è ancora quella di evitare le collisioni quando possibile. Questa tecnica usa un approccio "ascolta prima di inviare".Il problema è capire rispetto all'evento canale occupato quale approccio usare.
Se la stazione trova il canale occupato, non trasmette. Ma quando ricomincia a trasmettere? Si può usare come persistente o non persistente:

- **CSMA persistente**: Riprova immediatamente con probabilità p quando il canale si libera,
- **CSMA non persistente**: Si fanno scattare una serie di timer con un intervallo temporale casuale.

Questa tecnica non è sufficiente per evitare le collisioni.Si genera una collisione quando una stazione vede un'altra stazione se è occupata o meno. Nello stesso istante di tempo, se la stazione D prova a trasmettere, non vede ancora il segnale inviato da B prima di lei, e quindi può accadere che la stazione D, al tempo t_1, decide di trasmettere, e lo può fare. I due segnali poi vanno a collidere. Il primo punto fisico in cui si genera la collisione è tra C e D, e l'area a scacchi è l'area spazio-temporale in cui si genera la collisione. Perchè B si possa accorgere che si sia verificata una collisione, deve trasmettere un numero minimo di bit tale che la trasmissione del pacchetto sia ancora in corso nel momento in cui un'altra stazione fa arrivare il bit trasmesso. Se trasmette un pacchetto molto piccolo e chiude la trasmissione non ce ne si accorge: quindi deve trasmettere un numero minimo di bit tale per cui la trasmissione è ancora in corso in caso di collisione. C'è un legame esplicito (negli standard) tra la dimensione massima in termini fisici della rete (lunghezza del filo) e la dimensione minima in bit, fissata la velocità trasmissiva in bit/s e la dimensione minima del pacchetto. Quando uno trasmette un pacchetto su una rete Ethernet, c'è un limite massimo (MTU), ma anche un limite minimo (58 byte), e questo limite non è fissato in assoluto, ed è strettamente collegato alla massima dimensione. C'è un vincolo tra rate trasmissivo, massimo diametro fisico della rete e dimensione minima del pacchetto. Quando si verifica una collisione, i pacchetti sono pacchetti persi. Per evitare le collisioni bisogna fare una tecnica di **Collision Detection**: appena si rileva una collisione, una trasmissione interrompe la trasmissione, e così anche l'altra stazione, dopo un tempo di guardia minimo. Usando questa tecnica, la stazione B interrompe la trasmissione dopo un po' di tempo, e la stessa cosa accade per D. Questa cosa rende le stazioni più "pronte" per inviare di nuovo l'informazione.

Protocolli di tipo "Taking Turns" *(a circolazione di turno)*.

Sono ibridi tra i primi due, e cercano di prendere il meglio degli approcci precedenti: evitano le collisioni, ma non usano uno schema di partizionamento rigido. Si vuole consentire, con qualche

regola, ad una sola stazione alla volta di trasmettere. C'è un permesso di trasmissione che viene assegnato ad una trasmissione secondo una data funzione. Il permesso non è rigido, ma viene assegnato a chi ne ha bisogno. Una stazione può avere più permesso perchè ha più dati da trasmettere, e quindi può avere più tempo a disposizione per trasmetterli. Si può immaginare che ci sia un nodo coordinatore che interroga periodicamente i vari nodi, chiedendo se hanno dati da trasmettere. C'è una tecnica di **polling**. Questo tipo di schema viene implementato con messaggi di tipo **RTS (*Request To Send*)** e **CTS (*Clear To Send*)**. C'è un overhead dovuto alla richiesta e un tempo di latenza dovuto a questa richiesta circolare. Inoltre il nodo master può essere un punto debole. Un'altra tecnica, detta **Token passing**, fa a meno del nodo master e c'è un token che gli stessi nodi si passano come autorizzazione allaa trasmissione. Chi lo possiede in quel momento ha il permesso di trasmettere. Questo gettone di permesso di trasmissione viene passato, ad esempio, in maniera sequenziale. Questo token gira in maniera circolare, e siccome il permesso gira circolarmente, prima o poi tutti i nodi vedono passare il token e nessuno dovrebbe avere il token per tanto tempo.
C’era una tecnologia prima di Ethernet, detta **Token ring**, in cui i nodi fisicamente erano cablati in modo da formare un anello. Lo stesso schema è stato implementato anche in una rete che ha la topologia di un bus. E' la tecnologia del **Token bus**.

Reti di Calcolatori - 06/12/2012

Lezione 28. Reti locali Ethernet

L'indirizzo fisico che identifica univocamente la scheda di rete è una serie di 48 bit rappresentata con sei coppie di cifre esadecimali. Esistono indirizzi IP a 32 bit e indirizzi **MAC** usati per le schede. Per la distribuzione degli indirizzi si usa un protocollo MAC. Gli indirizzi non sono indicativi della posizione della scheda nella rete in generale, ma sono indicativi del costruttore che ha realizzato quella scheda. Il fatto che gli indirizzi siano collocati in uno spazio che non ha nessun riferimento fisico consente agli indirizzi di restare inalterati quando la scheda di sposta da un computer ad un altro computer, oppure se sta su un portatile che viene trasportato. Gli IP, invece, hanno una parte che identifica la posizione geografica.
I 48 bit sono divisi in parti: i primi tre byte (**OUI**) servono ad identificare il costruttore; gli altri tre byte sono una numerazione progressiva decisa dal costruttore. Se ci mettiamo in ascolto sulla rete e osserviamo i pacchetti che girano, possiamo capire i pacchetti da che tipo di macchina provengono dal loro indirizzo MAC.
L'indirizzo speciale **broadcast**, con 48 bit tutti 1, è l'indirizzo che identifica una frame destinata a tutte le schede nella rete locale. Quando faccio un frame con questo indirizzo destinazione la devono ricevere tutti i router. C'è anche la trasmissione **multicast**: gruppi di stazioni che vogliono ricevere quel pacchetto. Quando non si considerano gli speciali OUI per la trasmissione multicast, stiamo osservando pacchetti destinati ad un **unico destinatario**. Quando accade, la scheda di rete confronta il destinatario del pacchetto con il proprio indirizzo, scritto o nella ROM del dispositivo o in un buffer. Se i due indirizzi coincidono, il pacchetto viene catturato e passato agli strati superiori. Da un punto di vista numerico, è un vantaggio di velocità.

La tecnologia dominante per le reti locali è la tecnologia **Ethernet**. L'idea fondamentale è realizzare la rete locale con un bus condiviso sul quale si agganciano le varie stazioni. Il **bus** è un **cavo coassiale** che aveva due tappi di terminazione per impedire la riflessione dei segnali, e su questo cavo coassiale veniva realizzato l'aggancio delle stazioni. Il cavo, in punti opportuni, veniva forato, e in questi fori veniva inserito il **transceiver**, che trasforma il segnale elettromagnetico in un segnale elettrico che trasporta l'informazione come insieme di bit. Il transceiver porta il segnale elettrico che porta la sequenza di bit ricevuti e trasmessi. Questo cavo di interfacciamento è un cavo che collega il transceiver con la scheda di rete, che, montata all'interno del calcolatore, non possedeva la tecnologia per l'accoppiamento al cavo coassiale. Le onde elettromagnetiche si propagano attraverso questo mezzo trasmissivo (da etere, per questo si chiama Ethernet).

Formato dei pacchetti. (*slide 11*)

Il formato dei pacchetti trasmessi è rimasto, entro certi limiti, inalterato. Questo è uno dei vantaggi che ha consentito una buona evoluzione tecnologica.
C'è un **preambolo** iniziale che ha uno scopo di sincronizzazione. E' una sequenza di 7 bit alternati con un ultimo bit. Serve ai ricevitori per agganciarsi al flusso di bit, e per poter interpretare correttamente quello che viene dopo. Riconosciuta questa sequenza, garantisce al ricevitore che parte tutto il resto.
Dopo il preambolo seguono l'**indirizzo** di **destinazione**, l'**indirizzo** di **sorgente**, un campo **protocol type** che serve ad identificare il tipo di pacchetto che c'è dentro e poi chiude il pacchetto il **CRC (*Cyclic Redundancy Check*)**, che serve a chi riceve il pacchetto di verificare che sia arrivato inalterato.

Tecnica di accesso. (*slide 13*)

La tecnica usata è derivata dal **CSMA/CD**. Il transceiver ascolta il canale, e fintantochè il canale è

impegnato, cioè non è idle, allora aspetta di vedere che il canale sia libero. Quando è libero, la stazione trasmette il pacchetto, ma mentre lo trasmette, ascolta il canale, perchè durante la trasmissione si può verificare una collisione. Se si verifica, allora interrompe la trasmissione e invia su questo bus condiviso una sequenza di bit speciale detta **sequenza di jam**, che serve ad assicurarsi che tutte le altre stazioni sappiano che si è verificata una collisione. Aumenta poi il numero di collisioni che si sono verificate in precedenza di un'unità. Prima di ritornare al punto di partenza ,cioè a tentare una nuova trasmissione, si mette in un'attesa che dura una quantità di tempo regolata da un **algoritmo di exponential backoff**. Se c'è stata una collisione, quindi, si aspetta un po' di tempo. Se non è stata rilevata nessuna collisione, allora la frame appena trasmessa si ritiene correttamente ricevuta e si mette il numero di collisioni a 0.

Per essere sicuri che non si verifichino questi eventi, occorre che la trasmissione duri un tempo sufficientemente lungo per fare in modo che l'altra stazione che ha trasmesso il segnale abbia ricevuto una trasmissione che ho inviato io. C'è un legame tra trasmissione inviata da me e diametro della rete. I pacchetti trasmessi non possono essere arbitrariamente piccoli, ma devono durare un certo numero di byte proprio per questo motivo, e oltre a questo vincolo cen'è un altro che dice che la massima lunghezza tra la stazione e l'oggetto centrale della rete può essere massimo 100 metri. La sequenza di jam dura 48 bit.

Exponential Backoff.

Determina il tempo di attesa che occorre attendere prima di riprovare la connessione. Adatta il tempo di attesa in funzione del numero di collisioni verificate fino aquel momento. L'idea è: se la collisione che si è verificata è la prima, si può riprovare in tempi brevi, ma se la collisione che si è verificata è l'n-esima di una sequenza di n-1 precedenti, è indicatore sempre più forte che sulla rete c'è un forte traffico, e per evitare che si verifichi un'altra collisione devo distribuire i tentativi di trasmissione in un intervallo grande per evitare una collisione in una trasmissione successiva. Quando si verifica una collisione, alla prima collisione si sceglie K tra 0 e 1. Il ritardo di trasmissione è pari ad un intervallo K*512 bit (pari a 51 microsecondi in una Ethernet a 10 Mbps). Si individua uno slot di 51,2 microsecondi e alla prima collisione si decide se ritrasmettere subito o dopo 51,2 microsecondi. Dalla seconda collisione in poi si sceglie K in un intervallo tra 0 e 3. Al terzo tentativo si sceglie tra 0 e 7, e così via… L'intervallo di valori in cui possiamo scegliere K **aumenta esponenzialmente**, perchè si vuole evitare che due terminali comincino la trasmissione nello stesso slot, per evitare il verificarsi di nuove collisioni. E' un margine di sicurezza che serve a verificare che tutte le stazioni coinvolte nella collisione stiano al sicuro. Questo aumento esponenziale nel quale si sceglie la durata dell'attesa prima della trasmissione arriva fino a un **valore limite** di **1023**. Arrivati a 10 collisioni non ci si ferma nei tentativi ma, ad un certo punto, ci si ferma. Viene fatta una tecnica di **codifica Manchester** che, se la sequenza di bit da trasmettere è di un certo tipo, utilizza fronti di salita e fronti di discesa. Questo serve a fare una variazione di livello almeno per ogni bit trasmesso, in maniera tale da facilitare la sincronizzazione dei clock.

Tecnologia 10Base2. (*slide 18*)

Cambia il supporto fisico: era sempre un cavo coassiale più sottile di prima e quindi più flessibile. Questo cavo coassiale veniva collegato ai transceiver attraverso **connettori BNC**, e ogni stazione aveva, nell'estremità che esce dal computer, un **connettore a T**, che serviva ad agganciare il cavo coassiale da entrambe le parti, quindi non c'è più il tranceiver esterno rispetto alla stazione, ma stava internamente nel computer e la connessione avveniva direttamente con dei connettori. Questo tipo di cablaggio era più comodo perchè non richiedeva il tranceiver.

Tecnologia 10BaseT. (*slide 20*)

Questa tecnologia è stata rapidamente superata dall'implementazione 10baseT (**T** sta per ***Twister pairs***, *doppino*). Stavolta il mezzo trasmissivo non è più un cavo coassiale, ma un **doppino**, che tiene delle coppie dei fili di rame appositamente intrecciate. Ogni coppia porta un segnale da un'estremità all'altra, il cavo possedeva 4 coppie. L'intrecciatura tra i due fili della stessa coppia serve a ridurre la capacità dei fili di essere influenzati da interferenze per induzione dai campi elettromagnetici esterni. C'è un abbattimento dei costi, perchè questa coppia di fili è tipicamente molto più economica del cavo coassiale. Cambia molto anche dal punto di vista della topologia, perchè mentre prima c'era una topologia a bus, ora si passa una **topologia a stella**. La connessione avviene tra stazione e oggetto centrale, l'**hub**, che è un oggetto che ha tante porte e ad ogni porta dell'hub si collega un filo. Questi cavi vengono detti **UTP (*Unchieved Twister Pair*)**. Altri cavi di questo tipo sono gli STP (shielded), con una schermatura esterna, oppure FTP (folded), che hanno una schermatura esterna in alluminio. Più è schermato, migliori sono le caratteristiche elettriche del cavo per trasmettere il segnale senza distorsioni e interferenze ad alte frequenze. Questi cavi vengono identificati da un numero di categoria, che cresce con la miglioria delle caratteristiche elettriche del cavo. Il cablaggio di tipo 10BaseT e 100BaseT avviene secondo una topologia a stella, che è organizzata gerarchicamente: ci sono collegamenti di backbone tra hub di livello più basso e hub di livello superiore che li collegano tra di loro. **(*slide 21*)** I collegamenti di backbone vengono realizzati con tecnologie a maggiore bitrate.
Gli standard prevedono dei link: la massima distanza tra un nodo e un altro è **100 metri**.

Interconnessione di LAN: Hub, Bridge e Switch.

Non si può creare un'unica grande LAN. Un primo motivo è quello del **problema fisico dell'estensione**. Ci sono motivi geografici. Se le stazioni stanno troppo distanti, si può verificare che la stazione termina la trasmissione e la collisione si verifica ad una distanza non rilevabile. Se devo cablare zone troppo distanti devo trovare delle tecniche di interconnessione.
L'altro motivo è quello dei **problemi di collisione**. Se la rete ethernet è realizzata con un bus, per come funziona il CSMA/CD, questa tecnica di condivisione della capacità trasmissiva funziona anche in caso di collisioni. Se aumenta il numero di stazioni aumenta la probabilità di collisioni, quindi bisogna separare i domini di collisioni. Quando si verificano collisioni non viaggia niente di utile e quindi è traffico inutile. Se ho una rete a 100 Mbit e la uso al 10% della sua capacità per le collisioni non va bene, e allora è utile non mettere su un unico grande bus 100 stazioni, ma partizionare le stazioni che collidono tra di loro.
Il problema è quello di trovare un modo efficiente di dividere la banda tra le varie stazioni. Gli **Hub** mi consentono di passare da un cablaggio lineare al cablaggio a stella, che è comodo da realizzare, perchè metto l'apparato in un punto baricentrico dell'ufficio e faccio in modo che tutti i cavi convergono verso questo punto centrale. L'hub lavora a livello fisico.
Quando arriva a una porta un segnale trasmesso da una stazione, l'hub replica il segnale sulle altre porte (come se fosse un ripetitore di segnale). Se, contemporaneamente, arrivano due segnali provenienti da due stazioni differenti, mentre si fa questo lavoro, si genera una collisione su tutte e 4 le porte, quindi l'hub non riduce le connessioni, ma le genera lui. L'hub, con questo tipo di comportamento, realizza internamente quel bus che possiamo immaginare che ci sia. Da questo punto di vista, quando c'è l'hub, tutte le porte appartengono allo stesso dominio di collisione, che è un insieme di porte dell'hub che può generare collisione. In questa maniera, se questi oggetti sono tutti hub, tutte queste macchine formano tutte un unico dominio di collisione. Le porte dell'hub possono creare impostazioni tra di loro.
Questo tipo di collegamento non mi consente di far convivere reti locali con tecnologie differenti o reti ethernet con tecnologie differenti. Se voglio realizzare questo tipo di problematica, può avere senso mettere un dispositivo di interconnessione non sofisticato quanto un router ma non semplice

quanto un hub. Si chiama **Bridge**. E' un dispositivo che nasce per l'interconnessione di reti locali, che opera a livello 2, nel senso che, per ogni pacchetto che gi arriva, vede l'indirizzo e sulla base di questi indirizzi prende delle decisioni di inoltro. Ha un numero di porte limitato e quindi, quando arriva un pacchetto ethernet sulla porta, il bridge non trasmette un pacchetto su tutte le altre porte, come l'hub, ma lo trasmette solo sulla porta dove è necessario inviare il pacchetto, riducendo le probabilità di collisione.
A differenza di un hub, che è un puro ripetitore di segnale, un bridge riceve un pacchetto, lo mette in un buffer, osserva questo pacchetto e sulla base di quello che sta scritto nel pacchetto decide dove ritrasmetterlo. Prende questa decisione sulla base di informazioni a livello 2. Di fatto, un bridge, nel momento in cui ha tutto il pacchetto, lo può modificare almeno per quanto riguarda l'intestazione, e quindi un bridge potrebbe collegare reti con tecnologie differenti. Conoscendo la differenza di struttura della frame, può adattare nei limiti del possibile la frame, per adattarla alla tecnologia. Il bridge isola i domini di collisione, il che significa che se il pacchetto deve andare sulla rete 2 allora viene trasmesso su una sola linea. Il bridge vede anche che i pacchetti devono restare sulla parte giusta. Come fa a sapere sua quale segmento deve essere inoltrata? L'approccio utilizzato è guidato da un'**esigenza di trasparenza**. Questa configurazione deve avvenire in maniera automatica. Il bridge non deve essere configurato dall'amministratore della rete.
Un bridge utilizza un **algoritmo di autoapprendimento** per scoprire a quali interfacce sono collegati gli host. Ogni volta che il bridge vede una frame, prende nota dell'indirizzo MAC sorgente, e quest'informazione non è univoca. Queste tabelle, mantenute nel bridge, devono mantenere un numero elevato di entry. Nelle tabelle di filtering si salvano informazioni per scoprire a quali interfacce sono collegati gli host. Quest'informazione opera secondo lo schema: quando vede arrivare un pacchetto livello data-link con un indirizzo MAC di destinazione che è sulla stessa LAN ricevuta, il bridge ignora il pacchetto. Se lui già sa che l'host sta sulla rete da dove è stato inviato il pacchetto non lo deve inoltrare. Se invece il pacchetto è collegato su una porta diversa da cui proviene, fa una ricerca nella **tabella di filtering** costruita precedentemente. Se trova una entry perchè ha già appreso che la destinazione è implementata nella rete 2, allora la inoltra in quella interfaccia. Se invece la destinazione è sconosciuta, allora fa il ***flooding***: su comporta come un hub, e inoltra il pacchetto su tutte le porte tranne quella dalla quale il frame è arrivato. Da una parte c'è l'attività di apprendimento, dall'altra c'è l'attività di inoltro. All'inizio si comporta come un hub, poi apprende le porte cui sono collegati i terminali e crea la sua tabella. Ha un timeout per la tabella, venogono risolte problematiche sulla posizione delle entry (che vengono cancellate se dopo un po' non ci sono più invii verso un certo host).

Bridge Learning.

Quando A trasmette a C, tipicamente, C trasmette qualcosa indietro ad A. Questo frame con destinazione A è associato ad una destinazione che il bridge sa che sta sulla porta 1, e quindi sa che non deve fare nulla. Quando il pacchetto viaggia da C ad A, il bridge apprende qualcosa di nuovo. Se successivamente A invia un pacchetto a C, il bridge non ritrasmetterà il pacchetto sopra, perchè sa che A e C stanno sulla porta 1.

I bridge sono fonte di problemi. Se ho più bridge, nella topologia potrebbero generarsi percorsi chiusi, in quanto la rete avrebbe degli anelli. Se il bridge non si comporta come un router, che decrementa il TTL, quando c'è un percorso chiuso allora il pacchetto può viaggiare all'infinito.
La rete realizzata attraverso bridge a livello due può creare situazioni molto dannose che vanno evitate. Se ho questo problema o faccio attenzione e non metto fili in più (ma non ho neanche il vantaggio della ridondanza topologica) o faccio in modo che i bridge riconoscano i loop, si organizzino tra loro e coprano la rete attraverso uno spanning tree.
I bridge dialogano tra loro attraverso gli **spanning tree protocol**: infatti l'albero è loop free. Così si evita il problema di loop in una rete di bridge. C'è bisogno di opportuni protocolli di ordinamento

tra i bridge.

Il router opera a livello 3, il bridge a livello 2, l'hub a livello 1.
Questi oggetti a fine anni '80 sono serviti per interconnettere reti locali, poi quest'esigenza di connettere LAN con tecnologie differenti è venuta meno (l'ethernet si è imposto) e l'esigenza di ridurre l'ampiezza dei domini di collisione è rimasta, e quindi i bridge vivono come funzionalità all'interno dei switch. Nel caso di un pacchetto broadcast (ad esempio un ARP request broadcast), passa attraverso un hub, passa attraverso un bridge, ma viene fermato da un router (che non inoltrano pacchetti con indirizzo broadcast) e quindi, rispetto al traffico broadcast, il pacchetto si forma al router, e questo è un vantaggio: : se faccio una connessione con bridge il traffico broadcast di una rete trabocca sull'altra, mentre se l'interconnessione tra reti locali si fa con router il traffico broadcast rimane confinato con la rete. Con i router possono essere realizzate diverse tecnologie. Nel bridge le operazioni sono più semplici dei router ma hanno topologie limitate e possono realizzare più tecnologie. Richiedono una connessione IP (non sono plug and play) etc. Ma comunque oggi i bridge non si usano più.

Switch Ethernet.

Gli switch ethernet funzionano come i bridge. Dal punto di vista fisico, sono due oggetti che hanno delle porte (sono simili agli hub). Lo switch nasce per interconnettere stazioni o segmenti di tecnologia omogenea (si basa sull'ethernet). Data la minore complessità dell'hub rispetto allo switch l'hub ha meno porte. Oggi non è più vero. Lo switch ethernet serve a separare i domini di collisione. Apprende gli indirizzi MAC dei pacchetti che arrivano e cerca di inon inoltrarli […]. La differenza rispetto al bridge è che uno switch nasce come un oggetto che serve ad interconnettere stazioni di topologia ethernet. Gli switch isolano le reti, ma usano tipicamente degli **approcci di cut-through** (invece di *store-and-forward*). Non si bufferizza tutto il pacchetto ma, appena arrivano i primi byte, viene presa la decisione di inoltro, e il pacchetto viene inoltrato nella porta di destinazione. Mentre dalla porta esce la testa, la coda sta ancora entrando, quindi il pacchetto non vive interamente nello switch, anche per ridurre la latenza del dispositivo e per ridurre il buffer. Da un certo punto di vista, siccome riducono i domini di collisione, quando si usa uno switch all'interno di una rete ethernet si riduce il numero di collisioni ma non si azzera mai. Spesso i collegamenti di backbone sono realizzati con fibre ottiche. Ethernet funziona anche sulle fibre ottiche in realtà, usate per i backbone tra gli switch.

Topologie complesse per LAN: Il cablaggio strutturato.

Esistono degli standard che regolano le modalità, gli strumenti e i materiali che vengono utilizzati per realizzare una rete locale dal punto di vista del cablaggio. Questi standard sono derivati e nascono da standard analoghi concepiti per il mondo della telefonia. Nel corso del tempo si sono venuti a creare una serie di sistemi di cablaggio che sono proprietari, per poi essere soppiantati da tecnologie dettate da standard internazionali. Lo standard più rilevante è il **TIA/TIA 568**. Oggi sempre di più si tende a portare tutto sulla rete dati (cavi). In ambito degli edifici, occorre che ci siano delle predisposizioni infrastrutturali anche per quanto riguarda i locali adibiti ad ospitare gli apparati.
Gli standard definiscono il cablaggio fino ad un livello massimo di comprensorio. Servono per “situazioni di campus”. Comunque un complesso proprietario che non deve attraversare un suolo pubblico (lì cambiano gli standard in base a decreti legislativi).
Si adottano topologie di **cablaggio stellare**, che riproducono la struttura stellare realizzata su **livelli gerarchici**. Il cablaggio di ogni piano si attesta su un centro stella di comprensorio (MC, main cross

connect),), di edificio (IC, intermidiate) e uno a piano (TC o HC, telecommunication closed), in corrispondenza del quale tutti i fili del cablaggio di piano si attestano e dove è ospitato uno switch. Questa è la scala gerarchica. Può accadere che il centro stella di un piano particolare rappresenti quello di un edificio e se l'edificio è il più importante rappresenti quello di comprensorio. I collegamenti intermedi sono collegamenti di dorsale creati attraverso fibra ottica, ad esempio. All'interno del centro di piano c'è un armadio di piano, in cui c'è lo switch ed eventuali archivi.

Il cablaggio all'interno di un piano è **cablaggio orizzontale**, mentre il cablaggio che connette i vari centri stella di piano e il centro stella dell'edificio è detto **cablaggio verticale**. Termina in una placchetta utente (presa al muro), e il collegamento è detto **cavo di distribuzione di piano**. Tutti i fili del cablaggio orizzontale vengono attestati su pannelli detti **patch pannel** dove, a ciascuna estremità, si deve realizzare un collegamento con una porta di un hub o di uno switch, che avranno una porta per ciascun utente. Il cablaggio non si realizza mai facendolo terminare dentro l'hub o dentro lo switch, ma si mette un punto di permutazione distinto attraverso un cavo, che ha una certa lunghezza. Sostanzialmente lo standard ethernet prevede che la distanza tra hub e computer sia al massimo di 100 metri, e lo fraziona in 3 pezzi, uno che va dal patch dell'armadio alla presa dell'utente (90 metri), e poi 5 metri vengono usati per collegare il computer con la presa utente, e poi ci sono 5 metri che utilizzo per collegare il patch pannel con l'hub o lo switch che sta nell'armadio.

Reti di calcolatori - 07/12/2012

Lezione 29. Reti Wireless

Per un certo tipo di cavo ci sono due aspetti fondamentali che vanno tenuti in conto:

- Il **minimo raggio di curvatura**: il cavo non deve mai essere piegato eccessivamente. Il cavo nelle canalette deve sempre fare delle curve in maniera tale che il raggio di curvatura deve essere sempre grande.
- La **massima forza di trazione** che si può esercitare sul cavo. Il cavo va tirato per farlo passare nelle canalette ma non deve essere esercitata una forza eccessiva per non rovinare l'abbinatura delle coppie, ecc..

Se si vuole aggiornare la propria infrastruttura non è sufficiente cambiare gli apparati ma bisogna fare attenzione a verificare che la propria infrastruttura sia adeguata e non vada ricablata completamente.

[Roba inutile sul collegamento di cavi...]

Il cablaggio orizzontale è massimo di 90 metri, il cavo utente deve essere massimo 6 metri, la dimensione per il collegamento dall'armadio di piano al centro stella di edificio è massimo 500 metri e ci sono varie dimensioni che dipendono dal supporto.

Gli 8 fili che si trovano all'interno del cavo UDP hanno un significato:

- Colore e Bianco/Colore sono le coppie.
- Quando si vanno a connettorizzare i 4 fili vanno disposti secondo quell'ordine. In realtà non tutte le coppie stanno vicine nel connettore.

Se una coppia va in un senso e l'altra va nell'altro, si può avere un'inversione. Un cavo di questo tipo non va bene per una rete punto-punto. Questi connettori sono racchiusi in una guaina esterna di plastica, e per poterli connettorizzare bisogna avere una pinza crimpatrice.

Reti Wireless.

Le tecnologie wireless sono utilizzate in vari contesti. Il vantaggio delle reti wireless è quello di liberare l'utente dai cavi. L'uso prevalente è quello di realizzare punti di accesso senza fili: i terminali sono collegati alla rete senza fili di collegamento ma poi la rete raccoglie i segnali trasmessi o inviati dai terminali e li inviano agli access point, che sono cablati con tutto il resto della rete internet. Ci sono due problemi:

- Il **raggio di copertura** di un access point è **limitato**, e date le caratteristiche della propagazione delle onde elettromagnetiche gli ostacoli fisici sono un limite per la propagazione del segnale.
- Diventa possibile trasportare un apparato al di fuori di un access point e immesso nel raggio di copertura di un altro access point. Bisogna gestire **problematiche di hand-over**, cioè interruzione della connessione con un access point e continuo della connessione su un altro. L'hand-over, o hand-off, è la procedura che si verifica quando una connessione mobile cambia da una base station ad un altra.

Senza fili non implica automaticamente mobilità.

La **base station** è quel dispositivo collegato ad un plesso di rete cablato, e assume la funzione di **relay**: è responsabile non solo di inviare pacchetti tra terminale e rete ma, laddove debbano comunicare due terminali tra di loro deve far viaggiare il pacchetto tra i due. I terminali non dialogano tra di loro direttamente, ma sempre attraverso la base station. Il termine base station deriva dalle reti di telefonia mobile che sono organizzate secondo uno schema simile (stazioni trasmittenti di radio base che coprono col loro segnale una cella in cui vi sono i telefoni cellulari). Router e access point sono due funzionalità diverse che a volte sono messe nello stesso dispositivo: l'access point opera a livello due.
I **link wireless** sono usati tipicamente per collegare stazioni terminali alla base station, ma possono essere utilizzati anche nelle parti di backbone, per connettere varie stazioni intermedie. Questo può essere utile nelle zone nelle quali la rete infrastrutturata non è ancora arrivata (ad esempio, nelle zone rurali).

Caratteristiche.

Le **frequenze** e i **data-rate** dipendono dalle tecnologie. Si diversificano nel raggio di copertura e per il data rate che offrono ai terminali di utente. Ce ne sono varie che si distinguono per scopi e raggio di copertura, quindi anche per data-rate. Esistono tecnologie per coprire piccole aree (**PAN**), quei collegamenti in wireless tra telefono e computer o bluetooth etc. Poi in ambienti outdoor si vogliono coprire distanze maggiori, quindi si usano tecnologie nate per il cellulare come l'**UMTS** o in alternativa il **WiMAX**. Le tecnologie cellulari di prima generazione hanno basso datarate, mentre le tecnologie più recenti hanno un data-rate più elevato. Le ultime competono con le **WLAN**, standardizzate da IEEE e 802.11, che definiscono varie tecnologie per la realizzazione di reti wireless.

Reti non infrastrutturate.

La **tecnologia wireless** può essere anche non infrastrutturata, cioè **ad-hoc**: consente ai vari terminali di comunicare tra di loro. Nelle reti ad-hoc si prevede la possibilità di gestire il **multi-hop**: consentire ai pacchetti inviati da una stazione di arrivare ad un'altra attraverso dei computer che fanno da router per far arrivare il pacchetto a destinazione. Queste si chiamano reti ad-hoc o **MANET (*Mobile Ad-hoc NETworks*)**.
Quando nelle reti senza infrastruttura un nodo abbia la capacità di inoltro dei pacchetti verso gli altri nodi, si parla di reti MANET, nel quale c'è un protocollo di routing in esecuzione sui nodi.

Con **single hop** intendiamo che i collegamenti wireless fanno un solo salto tra stazione e base station (e questo può avvenire con reti infrastrutturate o reti ad hoc senza relay, le PAN). Con **multi hop** intendiamo che nelle reti ad hoc ogni terminale, per aumentare la copertura della rete wireless, possa far passare pacchetti e rispedirli. Sempre in assenza di infrastruttura, non c'è access point. Ma ci sono anche reti infrastrutturate che gestiscono multi hop, ed è il caso delle reti MESH. Nelle reti MESH l'access point non è dotato di un collegamento cablato nell'infrastruttura.
L'access point è collegato al gateway che connette la rete wireless all'interno dell'altra rete cablata. Oppure, eventualmente, l'access point può essere collegato alla rete attraverso una serie di router wireless, organizzati a formare tra loro una rete con tutti collegamenti wireless con una topologia prestabilita.
In questo modo posso offrire una copertura wireless con dispositivi per la cui installazione non serve un cablaggio. Purché l'oggetto funzioni basta mettere alimentazione elettrica (si mettono su tetti o lampioni). Le reti wireless MESH orbane si stanno diffondendo in varie città perché offrono un servizio di comunità offerto dal comune stesso agli abitanti. Sono anche partiolarmente utili per offrire connettiività in zone rurali dove non ho ADSL in tempo breve.
Attraverso una serie di hop posso offrire una connettività wireless in un quartiere o in una città. Sono anche utili per zone rurali.

Caratteristiche.

I collegamenti wireless presentano caratteristiche differenti dai collegamenti cablati.
La prima differenza è il fatto che il **segnale si attenua** in maniera molto più significativa **in funzione della distanza**. Viene trasmesso con una certa potenza dalla stazione ma, viaggiando in aria, arriva attenuato al ricevente, e questo ha delle conseguenze sulle tecniche di condivisione del mezzo.
Un'altra problematica è il problema che nella porzione di spettro utilizzato in questo tipo di trasmissione si generano segnali sia prodotti da altri dispositivi ma sia segnali di **distrurbo** prodotti da dispositivi che non appartengono alla rete. Ci sono una serie di dispositivi che possono interferire con i segnali trasmessi. Nella banda di frequenza del wifi arrivano interferenze da microonde e motori elettrici, ad esempio. Riguardo l'uso delle bande, la tecnologia utilizza lo spettro a disposizione: non tutti possono trasmettere onde liberamente. Per poterlo fare occorre autorizzazione perché la risorsa è condivisa. Esistono bande di frequenza dedicate. Le porzioni di spettro non vincolate sono ridotte. In tutti i paesi si lasciano libere determinate aree di frequenza. Una banda centrata sui 2.4 GHz è quella tipicamente usata, ultimamente si usa a 5 GHz.
La risorsa spettro è una risorsa preziosa, e non tutti possono trasmettere onde elettromagnetiche direttamente, e per poter trasmettere segnali occorre un'autorizzazione, perchè è una risorsa condivisa da tutte le stazioni.
Un altro problema è la **propagazione multipath**: il segnale è riflesso. Quando una stazione trasmittente trasmette il segnale al ricevente, questo segnale non arriva mai in un unica versione ma ne arrivano altre opportunamente ritardate.

WLAN/802.11.

C'è un **BSS (*Basic Service Set*)**, su cui si poggia l'infrastruttura. Si parla di **spread spectrum**: la trasmissione di un flusso di dati generato da una singola stazione interessa una porzione di spettro più ampia della necessaria allo scopo di minimizzare i disturbi esterni. Se spalmo la mia trasmissione su una porzione di spettro ampia, spero di avere un effetto più limitato di un'altra causa interferente. Sono sequenze di trasmissione che ricordano la tecnica CDMA. Queste tecniche di trasmissione spread spectrum sono usate nel **direct sequence** e nel **frequency hopping**.

Nel **direct sequence**, l'intero spettro di frequenza è diviso in 11 canali, dove su ogni canale è possibile mandare un flusso di bit. Questi canali sono parzialmente sovrapposti. Il massimo numero di canali che non si sovrappongono è 3. Altre coppie di canali interferiscono sicuramente tra di loro. In questo tipo di tecnica, per realizzare la protezione di informazione si genera un numero di bit maggiore di quello strettamente legato ai dati, per proteggere l'informazione dalle interferenze. Se l'interfereza è eccessivamente elevata, gli access point si mettono a lavorare su un altro canale.
Una tecnica diversa è il **frequency hopping**, che sta nel saltare sempre da un canale all'altro, e questi canali vengono utilizzati da una coppia di stazioni secondo una sequenza di salto di pseudo canale generato da un opportuno seme noto sia a ricevitore che a trasmettitore. Trasmettitore e ricevitore devono essere sincronizzati per funzionare questa operazione. In questa maniera, saltando continuamente, se c'è possibilità di interferenza su una frequenza saltando su un'altra frequenza si spera che non ci sia.

Questa sequenza è generata da un generatore di **numeri pseudo-casuali**. Nel caso dell'utilizzo della sequenza diretta, per proteggere l'informazione ogni volta che il mittente vuole inviare un bit, invia il risultato dell'OR esclusivo tra questi bit e N bit scelti in maniera casuale e che sia il mittente che il ricevitore conscono (questo è il **chipping sequence a 4 bit**). Il ricevitore prende la sequenza ricevuta e ne fa l'OR esclusivo con la stessa sequenza di chipping e ottiene un segnale che è quello giusto. Se c'è stato un bit sporadico, in 4 bit consecutivi posso avere tre 1 e uno 0, e allora posso dedurre che sono quattro 1. Si sceglie quindi il bit in base alla probabilità, e si prende quello a maggioranza.

La connessione avviene attraverso uno scanning attivo o passivo.

Quando usiamo la tecnologia 802.11 il terminale per poter comunicare con un access point deve fare un'associazione, per rendere noto all'AP vuole inviare e ricevere dati. La connessione avviene attraverso uno scanning attivo o passivo.

- **Scanning passivo**: il terminale si mette in ascolto su frame speciali che sono legate a tutti gli altri access point. Si mette in ascolto e riceve due ***beacon frames***, e poi prende qualche decisione. L'associazione avviene secondo una richiesta esplicita. Invia una frame di richiesta di associazione all'AP selezionato, e riceve un frame di risposta e può cominciare a comunicare. Dopo questo il terminale ancora non è connesso al livello IP. Successivamente viene inviata una richiesta di DHCP, per ottenere un indirizzo IP. Se il terminale non ottiene un indirizzo IP non può comunicare. In questa fase di associazione c'è anche una procedura di autenticazione, che è incorporata negli standard.
- **Scanning attivo**: il terminale, anziché aspettare i frame di beacon, invia lui una richiesta di associazione mediante una ***frame di probe request***, che viene inviata in broadcast. Gli AC rispondono con una frame di risposta. Il terminale sceglie l'access point a cui si vuole associare, invia un messaggio di richiesta di associazione e riceve un messaggio di risposta di associazione.

C'è il **problema del terminale nascosto (*slide 21*)**. Supponiamo di avere due terminali A e C che sono in grado di raggiungere la stazione B, ma non riescono a vedere i segnali generati dall'altro perchè tra di loro c'è un ostacolo. In una situazione di questo tipo, quando A e C trasmettono a B, se trasmettono insieme generano una collisione di segnali, e B non è in grado di ricevere correttamente né il segnale di A né il segnale di C e si genera un'interferenza, che rende la comunicazione non possibile. In questa situazione né A né C sanno che si sta verificando un'interferenza.
Un altro problema è il **problema dell'attenuazione del segnale (*fading*)**: il canale radio ha una caratteristica di attenuazione che fa sì che il segnale mantenga una potenza in ricezione adeguata per una certa distanza per poi crollare bruscamente. Secondo questa caratteristica, se A, B e C sono

disposti in questa maniera **(*slide 23*)** quando a trasmette a B, arriva potente, però il segnale che A trasmette non arriva ad una potenza adeguata a C. Lo stesso accade per il segnale che C trasmette verso B. A e C non vedono i segnali inviati da C ed A stessi. I loro segnali arriverebbero singolarmente con una potenza adeguata a B, ma si genera un'interferenza che rende le due trasmissioni inutili. Nè A né C sanno che le loro trasmissioni hanno generato interferenza.
Per questi motivi, il protocollo di accesso a mezzo condiviso nelle reti wireless non usa la tecnica di collision detection. La tecnica di accesso al mezzo condiviso (etere), è una tecnica detta **CSMA/CA (*Collision Avoidance*)**: si cerca di evitare il verificarsi di collisioni, perchè non si è in grado di rilevarle. Per questo problema è anche difficile confrontare segnale trasmesso e segnale ricevuto usando l'approccio per le ethernet, cioè confrontando la potenza del segnale trasmesso e ricevuto. I due segnali sono a livelli di potenza diversi quindi il confronto non si fa facilmente, e, nel caso, non è affidabile.
C'è poi il **problema del nodo esposto (*slide 24*)**. Immaginando di avere 4 nodi, e B sta trasmettendo ad A. Il raggio di trasmissione include anche C e quindi il segnale arriva anche a C. C vuole trasmettere a D, ma il segnale arriva anche a B. Se B e C trasmettessero contemporaneamente, i due segnali nella zona verde sarebbero sovrapposti, ma all'esterno della zona verde, i segnali non sono sovrapposti, perchè A non riesce a ricevere il messaggio trasmesso da C,e quindi non c'è interferenza. Se C trasmette, D è in grado di ricevere. Il problema è che, quando B trasmette, C se ne accorge, e evita di trasmettere, perchè pensa che ci sia interferenza, e quindi in questa trasmissione, le trasmissioni contemporanee di B e C si potrebbero fare ma non si fanno.
Prima di trasmettere si cerca di sentire la situazione sul canale e si evita di generare trasmissioni quando il canale è già occupato. Il fatto che non si effettui collision detection fa sì che […].
Se il canale è inattivo, prima di trasmettere si aspetta un tempo di **DIFS (*Distribuited Inter Frame Space*)**. Se durante quell'intervallo di tempo il canale è libero si comincia la trasmissione dell'intera frame. Se invece durante quel tempo di attesa il canale è occupato, si usa una tecnica di timer casuale, incrementato quando si è trovato il canale occupato e decrementato quando si riesce a trasmettere con successo. Una volta inviata la frame il mittente si aspetta di ricevere un segnale esplicito di ACK dal ricevente, che potrebbe essere l'access point. Una volta ricevuta la frame, il ricevente calcola la CMC e ancora una volta invia al mittente un segnale di riscontro dopo aver aspettato un tempo di attesa. Il numero casuale serve per evitare che ci siano collisioni. Se c'è collisione non si verifica l'ACK e la trasmissione va ripetuta. Le altre trasmissioni, quando vedono la trasmissione in corso, si mettono in uno stato di attesa. C'è un **campo duration** che dà informazione sulla durata della connessione, e le altre stazioni prima di provare a vedere se il canale è libero o meno si mettono in un'attesa la cui durata è dettata dal mittente. Non si può utilizzare una pipeline, ma l'ACK viene dato pacchetto per pacchetto.
(*slide 28*) Il protocollo prevede una forma di trasmissione attraverso una prenotazione del canale. Il mittente trasmette una **frame RTS (*Request To Send*)**, che ha una probabilità di collisioni abbastanza bassa. A questa richiesta c'è un **messaggio** di risposta in broadcast detto **di CTS (*Clear To Send*)**, frame che viene data in risposta a una specifica richiesta in broadcast, in modo tale che tutti quanti sentano. Quando il mittente riceve la frame CTS, trasmette la frame dati e le altre stazioni aspettano. In realtà è un eventao di collisione di pacchetti RTS con probabilità piccola. Se c'è la collisione le due stazioni si mettono in attesa per un tempo casuale per evitare che alla seconda trasmissione si possa verificare una nuova collisione.

802.11 frame: addressing. (slide 30)

Il pacchetto viaggia in aria su una certa struttura, e sul segmento di rete ethernet viaggia con un'altra struttura, e questa differenza è dovuta al fatto che nella struttura del pacchetto 802.11 ha 4 MAC address.

1. Primo destinatario in aria del pacchetto.
2. MAC address dell'access point che trasmettono il frame.

3. MAC address dell'interfaccia del router a cui si collega l'AP. In questa maniera, l'AP può formare facilmente la frame sul pacchetto ethernet prendendo come indirizzo del destinatario il 3 e quello del mittente il 2.
4. Usato all'interno dell'infrastruttura in certe situazioni.

Distribution System. (*slide 36, slide 41*)

Se A vuole trasmettere ad F, c'è il percorso A->AP1->AP3->F. In questa situazione serve il quarto indirizzo MAC. In questo tipo di situazione, vengono usati tutti e 4 gli indirizzi, e nelle modalità infrastruttura, degli indirizzi 2 e 1, uno dei due campi contiene l'identificare il **BSSID** (identificativo della rete, indirizzo dell'AP). Nel viaggio della parte cablata, gli indirizzi contengono, oltre a stazione mittente e destinatario, anche gli indirizzi dei due access point che si trovano in mezzo.

Reti di Calcolatori - 11/12/2012

Lezione 30. Protocolli per flussi multimediali: RTP ed RTCP

Il problema sta nel trasferire dei flussi multimediali su una rete basata su protocollo non affidabile UDP. Questo flusso di informazioni, è un'informazione che viaggia in maniera continua da una sorgente verso uno o più ricevitori che devono ricevere. L'informazione multimediale porta informazioni trasmesse in forma codificata, cioè si tratta di informazioni digitalizzate, rappresentate mediante sequenze di bit, che non viaggiano sulla rete così come sono state ottenute dalla sorgente ma a monte dell'operazione di trasmissione c'è un'operazione di codifica che serve a rappresentare la stessa informazione con un minor numero di bit (diminuire il bit-rate). Senza un'operazione di conversione ci sarebbe un flusso di bit molto elevato.
Quest'operazione di codifica viene detta di **compressione**. Si cerca di rappresentare la stessa informazione con un numero minore di bit, tenendo conto del fatto che la sorgente produce questi bit con un certo grado di ridondanza.
Viene utilizzato da programmi per produrre archivi ridotti. Strumenti di questo tipo rappresentano una certa informazione con un numero minore di bit. Questo tipo di operazione è senza perdita, nel senso che non c'è alterazione dell'informazione. Possiamo dire che l'operazione di conversione può introdurre un certo grado di perdita, purchè questa perdita non sia percettibile ai nostri sensi, che hanno una capacità di rilevare l'informazione in maniera limitata, e possono tollerare leggere distorsioni dell'informazione.
Il concetto fondamentale che si utilizza è quello di utilizzare i **codici a lunghezza variabile**. I **codici a lunghezza fissa** sono i più semplici, ma questa cosa è giusta quando la probabilità che io scelga un codice invece di un altro è sempre la stessa. Quando invece un colore si presenta con maggiore frequenza rispetto agli altri, è più conveniente rappresentare i codici che si presentano maggiormente con stringhe di bit più brevi e i codici che si presentano in maniera minore con stringhe di bit più lunghe. Si studia la **frequenza statistica**. Se faccio questo tipo di scelta, uso un numero minore di simboli per trasmettere il dato. Questo tipo di tecniche vengono dette di **compressione senza perdita**. Altre tecniche introducono un certo tipo di perdita, ma non è grave perchè sono perdite che i nostri sensi non riescono a percepire.
Il ricevitore, dalla sequenza di pacchetti che riceve, deve fare un'operazione inversa di **decodifica**.
Ci sono flussi di informazione generati in tempo reale (live), catturati dalla sorgente, e flussi di informazione pre-registrati, prelevati da file già memorizzati su una memoria di massa.
Le operazioni di codifica possibili sono differenti, nel senso che un'operazione di codifica su un flusso video live deve essere più veloce della codifica di un video pre-registrato su un disco.
Quando l'informazione è preregistrata, potrei immaginare di fare il **transfert**: prendo tutto il file e lo mando dalla sorgente alla destinazione. Questo tipo di tecnica va bene solo per documenti di piccole dimensioni, che si possono trasferire in un tempo breve. Se il file è grande, allora si deve fare lo **streaming**: cominciare a riprodurre l'informazione mano a mano che viene trasferita attraverso la rete. Con questa tecnica non è neanche necessario che tutto il file originale sia memorizzato interamente dal destinatario, e questa cosa è idonea quando il ricevitore è un dispositivo che non ha una grande capacità di memoria. Se la rete fosse in grado di fornire un modello di servizio a commutazione a circuito, allora al ricevitore non sarebbe necessario mantenere grosse quantità di informazione. Ogni volta che un campione audio viene inviato al destinatario, così come arriva può essere riprodotto. Se il flusso fosse continuo non sarebbe necessario memorizzare una grossa quantità di dati, ma questa cosa non è possibile perchè la rete trasferisce l'informazione in maniera discontinua.
Ciò che dà fastidio è il **jitter**, la variabilità del tempo di attraversamento dei pacchetti nella rete. Se due pacchetti partono a 1/25 di secondo uno dall'altro, e arrivano a distanza di 100 ms, avremo un

buco di di tempo durante il quale il ricevitore non vede ancora arrivare il secondo pacchetto. Se questa cosa è variabile, la qualità dell'informazione percepita dall'utente viene degradata. Oltre alla perdita di pacchetti la variabilità del tempo di latenza tende a far perdere la continuità del flusso d'informazione che arriva al destinatario.
Se potessimo avere un modello di servizio della rete ottimale, allora potremmo realizzare questo servizio senza problemi. Però, il **modello di servizio best-effort** (offerto dal protocollo IP) non ci dà nessuna garanzia, e allora bisogna intervenire con dei meccanismi di recupero, che ci consentano di recuperare le perdite. Sulla perdita di pacchetti o si ritrasmette o si codifica l'informazione con un certo grado di ridondanza. L'approccio basato sulla ritrasmissione non è idoneo, perchè la ritrasmissione introduce un ritardo, ma è meglio la **ridondanza**. Il tempo che si aggiunge rende il flusso d'informazione particolarmente irregolare, e quindi andiamo contro all'esigenza di continuità, e i meccanismi di recupero da perdita non sono idonei per questo tipo di informazioni. Si ritiene che il protocollo di trasporto più idoneo è il protocollo UDP, che non implementa meccanismi di ritrasmissione.

Quando si tratta di flussi audio, perdite di pacchetti o jitter che diventa eccessivo, rendendo irregolare il flusso audio riprodotto, vengono percepiti come dei disturbi. Nel caso dei flussi video si hanno dei disturbi localizzati nel tempo e nello spazio. Queste tecniche di modifica sfruttano il fatto che la sorgente ha un certo grado di correlazione tra un pixel e i pixel adiacenti nello spazio e nel tempo. Per ridurre il bit rate c'è collegamento tra la luminosità di un pixel e punti adiacenti nel tempo. Il fatto che la codifica sfrutti questa correlazione fa sì che quando viene alterato un pixel, l'effetto di questa alterazione si produca non solo per i pixel vicini nello spazio, ma anche quelli vicini nel tempo.

Contromisure.

O si usano tecniche di **forward error correction**, che mi consentono di recuperare l'informazione dai bit che mi sono già arrivati, o si implementa un **buffer nel ricevitore**: il ricevitore non riproduce l'informazione appena è disponibile, ma viene consumata dal ricevitore prelevandola da un buffer, nel quale si accumula l'informazione progressivamente. I pacchetti sono accumulati in una coda e quando c'è un grado di riempimento adeguato si consumano i pacchetti. C'è un meccanismo di compensazione rispetto a variazioni di ritardo limitate nel tempo, a patto di introdurre un ritardo aggiuntivo perchè possa fare questo. Si introduce un ritardo iniziale che ha un effetto benefico sul flusso. Senza buffer quando un pacchetto arriva troppo tardi di fatto il ricevitore non lo trova e passa al successivo, quindi lo salta.
I ricevitori dimensionano il delay rispetto alla variazione di jitter e perdite. Quando si verificano eventi di perdita tipicamente si aumenta il buffer e ce ne si accorge perché la riproduzione si arresta e poi prosegue finché il jitter non dovesse ulteriormente aumentare a seconda dell'ultima stima fatta. Non possiamo bufferizzare troppo perchè è poco opportuno chiedere al ricevitore memoria troppo elevata. Un altro fattore limitante è il ritardo: noi desideriamo non passi troppo tempo dall'avvio della riproduzione in poi.
Ci sono altre applicazioni nelle quali il tempo di latenza non può essere aumentato in maniera indefinita, ad esempio le applicazioni interattive. Se c'è troppa latenza non si riesce a giocare, ad esempio. Anche l'interazione tra persone: se si fa una telefonata e il tempo di latenza è troppo grande l'interazione diventa difficile. Quando il ritardo supera i 500ms diventa difficile interagire. Allora, per questi fatti, il flusso audio che stiamo bufferizzando non si può bufferizzare troppo altrimenti si va incontro a questi problemi.

Protocolli di trasporto più idonei: Protocollo RTP.

Il protocollo più idoneo è l'UDP arricchito: l'**RTP**. E' abbastanza utilizzato, ma non lo è

universalmente. Oggi per fare streaming di flussi video si fa ampio ricorso ad HTTP, che funziona su TCP (ad es.: youtube). L'HTTP si cerca di usarlo in maniera abbastanza furba, cioè non viene trasferito l'intero file e poi comincia la riproduzione, ma si fa una specie di streaming, nel senso che il ricevitore cerca di correlare quanti byte sono arrivati in quel momento e quando si accorge che il buffer si sta svuotando interrompe la connessione e ne avvia un'altra invece di chiedere la ritrasmissione. Non è un vero e proprio file transfer. Si fa questo per due motivi: il primo motivo è che HTTP è naturalmente implementato nei browser, e poter lavorare su un protocollo già implementato nei browser non richiede che siano installati altri plugin. Ma un altro buon motivo è più legato alla rete. Gli ISP non vedono di buon occhio i flussi UDP, e quindi tipicamente tendono a bloccare o almeno limitare i flussi UDP. Per cui, tendono a interferire negativamente con questo tipo di flussi, soprattutto quando sono voluminosi. Allora si cerca di utilizzare il più possibile il TCP sulla porta 80, che gli ISP fanno passare tranquillamente.

Real-time Transport Protocol.

È stato definito in una serie di RFC dell'89. È un protocollo di trasporto che si monta sopra UDP, nel senso che sono incapsulati in datagrammi UDP, che hanno un'intestazione specifica del protocollo UDP.
È stato progettato sul modello **ALF (*Application Layer Framing*)** che dice come un flusso di dati debba essere frammentato in segmenti, cosa che non dev'essere decisa da un modulo software che è completamente non consapevole dell'informazione (come avviene con HTTP: i segmenti TCP che viaggiano sulla rete contengono dati definiti da meccanismi da controllo di flusso e congestione, c'è poca comunicazione con quello che è stato generato a livello applicativo).
Se i due pacchetti hanno vite differenti, e uno si perde e l'altro no, il fatto che sia arrivato un pacchetto su 2 rende inutili tutti e 2 i pacchetti. L'idea è che siano le applicazioni a definire ciò che va in un'unità di trasmissione. Per questo motivo, questo **protocollo** è **implementato nelle applicazioni**, e non nel sistema operativo. Questo protocollo è neutrale rispetto alla codifica utilizzata: può essere utilizzato per trasportare flussi di qualunque tipo di codifica.
L'informazione fondamentale dell'header RTP è il **time-stamp**: l'istante di tempo in cui l'informazione è stata generata, che ha un significato relativo. Non conta tanto che quel pacchetto sia stato generato ad una certa data e ora in senso assoluto, ma interessa il time-stamp per poter collocare sull'asse dei tempi in maniera adeguata un pacchetto rispetto ai successivi. Questo serve a poter riprodurre i pacchetti negli istanti di tempo più opportuni.
Sono meccanismi che funzionano bene per flussi in **multicast**, dove alcuni meccanismi sono pensati per scalare quando il numero di ricevitori diventa elevato. Separa la trasmissione dei dati dalla trasmissione delle informazioni di controllo. C'è un protocollo compagno **RTCP (*Real-time Transport Control Protocol*)** che serve a scambiare informazioni di servizio e di controllo sulla qualità della trasmissione.
Il pacchetto RTP ha un header IP, un header UDP che contiene l'informazione di port number sorgente e destinazione (il multiplexing sta su UDP), un header RTP e il payload (che contiene i dati). RTP usa numero di porto di destinazione pari.
Nell'header RTP c'è un campo di versione, un numero di sequenza e un time-stamp.
Il **numero di sequenza** è un numero progressivo, che viene incrementato ogni volta che viene generato un pacchetto. Il **timestamp** serve a rimuovere il jitter prodotto dalla rete tramite bufferizzazione. Il numero di sequenza serve ad evitare la perdita dei pacchetti. Il campo **payload type** serve a capire il tipo di flusso, e quindi la codifica dei dati, all'interno del payload. Il **numero di identificatore di sorgente** è scelto casualmente dalla sorgente stessa e serve a non dover fare affidamento sull'IP per identificare la sorgente, perchè è un'informazione di livello rete non significativa, in quanto potrebbe cambiare. C'è poi il CSRC che è una sequenza di n campi, dove ciascuno dei quali serve a identificare la sorgente originaria in un flusso prodotto dalla fusione di flussi diversi mediante un mixer software. In questa situazione il campo SSRC identifica il mixer e

CSRC la sorgente.

Sessione RTP.

Una sessione RTP è un'ssociazione tra un gruppo di entità che comunicano trasmite RTP. Abbiamo anche **sessioni multicast**, un gruppo di ricevitori a cui si trasmettono un certo numero di sorgenti, e la trasmissione dei pacchetti avviene attraverso le tecniche di multicast IP. Quando ho diversi flussi audio/video che devono viaggiare sulla stessa coppia di PC, possono generare anche due sessioni diverse RTP (audio e video), che si differenziano per il numero di porta utilizzato.

Non possono mai essere generati due pacchetti diversi con lo stesso numero di sequenza, però possono essere generati due o più pacchetti diversi con lo stesso timestamp, a causa di una possibile frammentazione dei pacchetti (ad esempio, stiamo trasmettendo un frame video molto grande, di 8000 byte, che vanno frammentati in tanti pacchetti). Questi pacchetti sono trasmessi come RTP con sequenza diversi ma tutti con lo stesso timestamp e il destinatario deve metterli tutti insieme e riprodotti allo stesso istante. Può accadere che, di questa frame spezzata in più pacchetti, uno non arriva e gli altri che arrivano li devo buttare tutti, perchè manca il pacchetto che completa il file, perchè non sono previsti meccanismi di recupero, in quanto si ritiene che ci sia minore impatto nel non riprodurre l'informazione mancante piuttosto che ritrasmettere l'informazione.

Il protocollo RTCP.

Mittente e ricevente si scambiano **informazioni sulla qualità dei pacchetti ricevuti**. In questa maniera, il trasmettitore è in grado di capire se la trasmissione sta andando a buon fine o meno. Questo va bene soprattutto per le trasmissioni multicast. Posso immaginare che c'è una congestione e, se dello stesso flusso video cen'è uno a più bassa risoluzione e uno a più alta risoluzione, posso passare alla bassa risoluzione per diminuire ancora di più il bit-rate. Il problema è che se arriva l'80% dell'informazione, la tolleranza di perdita è ad un valore di soglia, e quello che arriva al di sotto del valore di soglia è pattume. Laddove è possibile, meglio passare ad un flusso che in partenza sia generato ad una qualità più bassa ma che sicuramente arriva a destinazione. Ci sono anche **informazioni di identificazione**, e in questa maniera il trasmittente riesce a capire chi sono i ricevitori, sopratutto in comunicazioni multicast.
Ci sono cinque tipi di messaggi nel protocollo:

- **SR**
- **RR**
- **SD**
- **BYE**: indica che sto uscendo dall'applicazione
- **APP**: tipo di messaggio le cui funzioni sono definibili in funzione della necessità

Banda usata da RTCP.

Siccome questi messaggi sono inviati da ogni ricevitore verso la sorgente, questa cosa come funziona quando ho un multicast con un milione di ricevitori? Per evitare questo fenomeno di affogamento della rete, i report non sono inviati in unicast al mittente, ma sono anch'essi inviati in multicast, e, in questa maniera, ogni appartenente al gruppo multicast può monitorare la quantità di messaggi RTCP generati nel gruppo, e può regolare il periodo con il quale invia i suoi propri messaggi. Se si accorge che vicino a lui già c'è chi manda report non lo manda e aspetta. L'idea è che i vari ricevitori debbano regolare il periodo medio di invio dei report in modo tale che la banda consumata da RTCP non superi il 5% della banda usata dalla sessione multicast. Questo avviene regolando i timer che regolano l'invio dei report successivi.

Sicurezza: tecniche crittografiche.

La sicurezza si può intendere da vari punti di vista.

1. La comunicazione tra due persone come si può rendere sicura? Si deve assicurare che le informazioni che le due persone si scambiano non siano rese disponibili a terze parti. Questo si realizza mediante **codifica con crittografia**, e si fa in modo da proteggere l'informazione, in modo tale che solo destinatario e mittente siano in grado di decodificare l'informazione. Questo è un **problema di riservatezza**.
2. Siccome questi messaggi viaggiano attraverso la rete, su cui possono operare terze parti, si vuole essere sicuri che un messaggio inviato da un certo mittente arrivi inalterato, e se viene fatta quest'**alterazione** deve essere **rilevabile dal destinatario**.
3. La terza problematica è l'**autenticazione**: il ricevitore deve essere sicuro che il messaggio gliel'abbia mandato proprio il mittente da cui lo deve ricevere. Bisogna proteggere i servizi offerti da attacchi detti tipicamente di **DoS (*Denial of Service*)**.

Si usano tecniche di **crittografia**. Tecniche di rappresentazione dell'informazione mediante un codice che trasforma il testo in chiaro in una sequenza di simboli che costituiscono il testo cifrato, che contengono un'informazione in una forma non immediatamente estraibile. Le teniche si dividono in due famiglie: chiave simmetrica e chiave pubblica.

- **Chiave Simmetrica**: l'informazione è protetta a seconda di una conoscenza segreta condivisa tra mittente e destinatario.
- **Chiave Pubblica**: si usano due chiavi differenti per le operazioni di cifratura e decifratura.

Cifrario per sostituzione.

Posso usare tecniche efficienti che scartano a priori chiavi di cifratura impossibili ma mi guidano verso la soluzione, facendo un'analisi statistica delle frequenze di occorrenza e usando pattern ricorrenti. Se conosco le frequenze di occorrenza posso facilmente decrittografare le chiavi simmetriche con un cifrario monoalfabetico. Le lettere hanno anche una ridondanza intrinseca (dopo la q la u, prima della q la c). Quindi posso fare meno tentativi di un brute force attack.

Si può anche spezzare il testo da decodificare in k bit tutti della stessa dimensione, e l'operazione di cifratura trasforma un testo di k bit in chiaro in un testo di k bit cirfrati.

Reti di calcolatori - 13/12/2012

Lezione 31. Esercitazione su QUAGGA

```
cat proc/sys/net/ipv4/ip_forward

echo 1 >> /proc/sys/net/ipv4/ip_forward
```

Reti di Calcolatori - 14/12/2012

Lezione 32. Sicurezza nella comunicazione in rete

Gli attacchi si possono fare da due punti di vista:

- **host bersaglio dell'attacco**: realizzato attraverso il punto di vista dell'analisi del sistema operativo dell'host.
- **guardare il traffico di rete**: si osserva l'insieme dei pacchetti che viaggiano sulla rete e si cerca di capire se ci sono pattern anomali di traffico che sono tipicamente noti per essere associati a degli attacchi.

Ci sono anche **attacchi relativamente a macchine**, sfruttando il fatto che il three-way-handshake se non si completa, lascia delle risorse allocate e quindi, se uno fa un grosso numero di tentativi di connessione senza mai portarli a termine può mettere in difficoltà la macchina server (*attacchi DoS*).
Tipicamente si sfruttano non tanto le falle dei protocolli, che non dovrebbero esserci, ma falle legate alla non perfetta implementazione dei protocolli stessi, o comunque spazi lasciati da specifici protocolli che sono incompleti.
L'operazione inversa di estrazione del testo in chiaro dal testo cirfato è fatta dal ricevitore sfruttando una chiave di cifratura. Se le due chiavi sono uguali, la **crittografia** è **a chiave simmetrica**.
Occorre disporre un canale sicuro che, però, potrebbe non esistere, per il trasferimento della chiave.
Per venire incontro a questo tipo di limitazione, è nata la **tecnica della crittografia a chiave pubblica**, nella quale la chiave utilizzata nella fase di cifratura è una chiave pubblica associata al destinatario. Il destinatario utilizzerà la sua chiave privata, segreta, nota solo a lui.

Tipicamente si usano **tecniche di cifratura a blocchi**: il messaggio da cifrare non viene preso nella sua interezza, ma la lunghezza del messaggio da mandare può essere variabile. Il messaggio da cifrare viene sempre spezzato in blocchi piccoli di dimensione fissa. Se diventano troppo piccoli, l'operazione di attacco attraverso tutte le modalità di cifratura diventa realizzabile, e quindi questi blocchi devono essere di lunghezza sufficientemente alta.
Per venire incontro alle 2 esigenze contrastanti si fanno cifrature prendendo un blocco grande, decomponendolo in blocchi piccoli, dove su ognuno si opera una trasposizione secondo una tabella, dopodichè gli 8 blocchi di codice vengono scambiati secondo un sistema di mischiatura e vanno in un nuovo blocco di 64 bit su cui viene operata una trasformazione simile, per un certo numero di volte. In questa maniera, se vediamo uno specifico bit nella sequenza in origine, questo bit, attraverso quest'operazione di miscelatura e codifica fatta per n volte, finirà con l'influenzare, nella parola codice finale, un certo numero di bit opportunamente distribuiti nei 64 bit finali. In questo modo diventa più difficile fare quegli attacchi basati sull'analisi statistica del testo cifrato.

DES (*Data Encryption Standard*).

Non sono noti approcci di decifratura con **backdoor**: nel mondo della cifratura, siccome i codici nascono in applicazioni militari, c'era sempre il sospetto che esista un meccanismo noto nell'algoritmo che consenta di superare in parte, o in tutto, le procedure di sicurezza.
Nel corso degli anni, è cambiato l'approccio alla sicurezza della cifratura. Negli ultimi anni, gli algoritmi sono noti, e quindi tutta la sicurezza sta nel mantenimento nella sicurezza della chiave stessa. Per questo si fa la chiave sempre più lunga.
La chiave da 54 bit è sempre mantenuta con 8 bit di parità.

AES (*Advanced Encryption Standard*).

Usa chiavi più lunghe, ed è stato stimato che un calcolatore in grado di decifrare DES in 1 secondo, impegherebbe 149 trilioni di anni per decifrare AES. A quel punto, il rapporto di proporzione dovrebbe mantenersi uguale nel tempo, ma comunque col passare del tempo si riduce il tempo che serve a decifrare la chiave. L'attacco a forza bruta, però, si presta molto ad essere partizionato.

CBC (*Cypher Block Chaining*).

Cifrare blocchi nei quali il testo cifrato ritorna indietro per un numero di volte. Cifrare una volta e poi farlo rientrare un'altra volta nell'algoritmo di cifratura serve a difendersi contro tecniche di decodificazione di crittoanalisi.

Crittografia a chiave pubblica.

Nella crittografia, per la cifratura si usa la chiave pubblica del destinatario, ma per la decifratura il destinatario solo è in grado di decifrare il testo a seconda della sua chiave privata. Quest'operazione è stata fatta da A sfruttando la chiave di cifratura pubblica di B. B fa una decifratura con la sua chiave privata e l'algoritmo è tale che si tiene il testo in chiaro usando le due chiavi insieme. Si può utilizzare: occorre trovare una coppia di chiavi tale per cui $+¿(m)$ $K_B^¿$ $¿$ $-¿¿$ $K_B^¿$, e, nota la chiave pubblica, dovrebbe essere impossibile ricavare la corrispondente chiave privata.

Algoritmo di RSA.

1. Si prendono due numeri primi grandi p e q
2. Si calcola, a partire da questi numeri, il prodotto $n=p\cdot q$ e poi si calcola l'altro numero $z=(p-1)(q-1)$. p e q, siccome sono numeri primi grandi, saranno sicuramente dispari, quindi p-1 e q-1 saranno pari, e z sarà sicuramente pari.
3. Si sceglie un numero $e<n$ che non abbia fattori comuni con z, che sia relativamente primo rispetto a z.
4. Si sceglie un numero d tale che $e\cdot d-1$ sia esattamente divisibile per z.
5. La chiave pubblica è la coppia **(n,e)**, mentre la chiave privata è la coppia **(n,d)**.

La parte n è nota, è d che protegge l'informazione. P e Q non escono esplicitamente, ma compaiono come loro prodotto.
L'operazione di cifratura si fa partendo dal testo in chiaro m e facendo l'operazione $m^e \bmod n$. Il testo cifrato c si invia al destinatario, che calcola $m=c^d \bmod n$. Questo perchè si può dimostrare che se uno fa quest'operazione ottiene un'altra volta il messaggio m originario, avendo fissato n,e e d nelle modalità precedenti.

Esempio: Quando si fanno queste operazioni di elevamento a potenza si ottengono numeri grandi e richiedono un certo lasso di tempo per essere calcolati. Cifrare e decifrare testi molto lunghi richiede molto tempo.

Un'altra proprietà notevole è che le coppie pubblica-privata si possono scambiare tra di loro nelle operazioni di cifratura e decifratura. Viene usata questa tencnica per verificare l'identità del mittente. La **robustezza** è dovuta al fatto che si parte dalla conoscenza di n, e se dato n riuscissi a decomporre n nei due numeri p e q avrei risolto, ma il problema è che non sono noti. Gli algoritmi DES sono 100-1000 volte più veloci di RSA. Nelle comunicazioni su rete, **RSA** è spesso usato in

combinazione con **DES** e **3-DES**. Sostanzialmente si fa generando la chiave di sessione. Le due parti che vogliono comunicare, comunicheranno attraverso una tecnica di cifratura a chiave simmetrica, quindi sicura. Però, come si fanno a scambiare questa chiave? Uno dei due decide qual'è la chiave, e poi ha il problema di comunicare la chiave a B, e allora usa per la comunicazione della chiave l'algoritmo di cifratura a chiave pubblica, cioè invia la chiave a B usando l'algoritmo a di cifratura chiave pubblica conoscendo la chiave pubblica di B, utile per la sessione. La chiave vale per un certo tempo, e per la sessione in cui serve. La chiave si mantiene per un certo lasso di tempo per non rendere semplice trovarla da terze parti.

Riservatezza: Problema dell'integrità

Bisogna assicurarsi che il messaggio che A invia a B non sia manipolato da una terza parte che si introduce nella comunicazione. Ci si assicura anche che chi ha inviato il messaggio a B è veramente A e non uno che si finge A. Le soluzioni che sono state immaginate si utilizzano su **funzioni hash crittografiche**. Una funziona hash è una funzione che prende un testo m e produce una stringa di lunghezza prefissata H(m). La funziona hash deve fare in modo che sia computazionalmente impossibile riuscire a ottenere che questi due messaggi abbiano lo stesso valore della funzione di hash. Fa sì che due stringhe differenti producano la stessa stringa, ma deve essere impossibile riuscire a ricavare, nota la funzione hash, le due stringhe x e y. Quando i due testi originali producono lo stesso valore della funzione hash producono una collisione. In termini equivalenti, significa dire che se mi è dato il valore della stringa hash H(x) non sono in grado di ricavare il testo x di partenza. Per esempio, la **funzione** internet **checksum** non soddisfa questo requisito, in quanto la funzione checksum non può essere definita una hash perchè, se si calcola su un testo, testo (i bit che compongono il costo del pacchetto sono messi su righe di lunghezza prefissata e la funzione è calcolata disponendoli per righe e applicando l'algoritmo), basta invertire due byte su righe differenti e sulla stessa colonna perché il testo rimanga lo stesso.

L'allineamento viene fatto su righe di 16 bit. È vero che la lunghezza dell'hash sarà sempre di 16 bit, indipendentemente dal numero di byte che sto calcolando, ma non gode della proprietà dell'irreversibilità o meglio di non saper generare collisioni (necessaria affinchè una funzione sia hash). Si usano delle funzioni calcolate con degli algoritmi che sono MD5 e SHA-1. MD5 non è perfetto perchè è stata trovata una tecnica che mi consente di trovare due messaggi che siano in grado di generare delle collisioni.
Come funzione di hash si usano **algoritmi MD5** e **SHA-1**. Il primo calcola un valore hash di 128 bit in 4 passi. Non è perfetto perché recentemente è stata trovata una tecnica per trovare due messaggi con lo stesso valore di hash, quindi che possono creare collisioni. SHA-1 usa un algoritmo differente.

Utilizzo pratico.

Rispetto al problema dell'integrità, la funzione di hashing ppuò essere usata così: supponiamo di avere A e B, e A invia un messaggio m a B e B vuole essere sicuro che il messaggio non venga alterato. A trasmette non solo il testo in chiaro m, ma una stringa che è prodotta da una funzione hash H(m+s) e da una stringa segreta s condivisa tra A e B. Questo hash viene aggiunto al messaggio m. Il destinatario B separa m e H(s) ricevuta. Sul messaggio ricevuto calcola la H(m,s), e dopodichè fa il confronto tra queste due: tra il valore di hash calcolato e il valore di hash ricevuto. Se questi due sono uguali il messaggio non è stato alterato, mentre se sono differenti il messaggio è stato alterato da qualche parte. Il problema non sta nel proteggersi rispetto a errori casuali, ma proteggersi rispetto a attacchi maliziosi. Se uno lo sapesse fare dovrebbe alterare m e anche la stringa di controllo, detta **MAC (*Message Authentication Code*)**. Un'evoluzione di questo problema della verifica dell'integrità è il problema della **firma digitale**. La forma di protezione

descritta prima presuppone la conoscenza della stringa s. Alice potrebbe dire che qualcuno ha alterato il messaggio o che essa stessa l'ha alterato. I requisiti che si chiedono ai meccanismi di firma digitale sono due requisiti di natura legale, che devono consentire al meccanismo di conferire al messaggio ricevuto le stesse proprietà di cui gode il messaggio quando viene inviato: deve essere verificabile e non falsificabile.

Si usa la crittografia a **chiave pubblica (asimmetrica)**: sostanzialmente B firma il messaggio m crittografandolo con la sua chiave privata, creando un messaggio firmato $-¿(m)_{K_B^¿}$, quindi c'è il messaggio in chiaro a cui si aggiunge un MAC ottenuto facendo l'operazione di cifratura sul testo m sfruttando la chiave di B. Alice riceve il messaggio e può verificare con la chiave pubblica di bob $+¿(m)_{K_B^¿}$ che è stato effettivamente B a comporre quel messaggio, decifrando con la chiave pubblica di B il testo cifrato. Verifica che il messaggio ricevuto che è stato trasmesso a parte e la sua firma decifrata coincidono. In questo modo, A verifica che il messaggio l'ha firmato B e nessun altro e che ha firmato quello e non m'.

Nella pratica, per i motivi di prima, non viene mai cifrato tutto il messaggio, ma viene cifrato un hash del messaggio. B produce il messaggio grande, dopodichè applica a questo messaggio una funzione di hash, e su questa hash applica la funzione di cifra digitale. Dopodichè ricava questa stringa, che è il codice di autenticazione. Questo messaggio così composto viaggia su un canale non sicuro, che può essere manipolato, e arriva al destinatario che separa messaggio e codice, sul messaggio calcola H(m) e sul codice di autenticazione applica l'algoritmo di decifratura sfruttando la conoscenze della chiave pubblica di B.

In realtà, ci si basa sul fatto che le chiavi pubbliche, di chiunque voglia firmare il documento, siano distribuite a tutti in maniera affidabile. Queste chiavi pubbliche vengono distribuite secondo **chiavi di distribuzione**, e perchè questo meccanismo sia realmente robusto e serva a scopi legali occorre che si costruisca un'infrastruttura di costruzione di chiavi pubbliche secondo un'autorità di certificazione affidabile (**CA, *Certification Autority***). Questa rilascia il certificato digitale attraverso procedure che seguono standard internazionali.

Come si usano le CA?

Lo scopo di una CA è quello di associare in modo sicuro una chiave pubblica e una particolare entità E (che può essere una persona, un ente, ecc…). B si presenta a quest'ente dichiarando di essere lui e presentando la sua chiave pubblica. La CA deve fidarsi dell'identità di B. La CA produce un certificato, che contiene la chiave pubblica di B, ma non da sola, firmata con la chiave privata della CA (che ha a sua volta una chiave privata e una pubblica, distribuita). Si trova un modo affidabile di distribuire la chiave pubblica: viene costruito questo documento che mette insieme la chiave pubblica di B e la sua firma prodotta con la chiave privata della CA. A chiede il certificato e applica l'algoritmo di cifratura usando la chiave pubblica della CA. Così ottiene la chiave pubblica di B.
Materialmente il **certificato** è un file che alcuni browser sono in grado di trattare, codificato secondo dei formati standard (X-509), che contiene varie informazioni.

Autenticazione.

Due parti si mettono in contatto e chi è contattato desidera che chi contatta provi la sua identità.
[Bob è un credulone. Alice dice che lei è Alice, ma c'è anche Trudy che si mette in mezzo!]
Si procede secondo un **protocollo ap3.0**. Si usa una password condivisa tra Alice e Bob. Il

problema è che un malintenzionato può sniffare i pacchetti e fare lo stesso meccanismo di autenticazione. Alice si presenta con la password e se è quella nota a priori a Bob lei viene autenticata. Ma Trudy può sniffare i pacchetti di Alice, isola la password e la usa per presentarsi maliziosamente a Bob come Alice!
La password è cifrata ma anche questa tecnica non è robusta: la cifratura dipende da una conoscenza a priori tra Alice e Bob ma c'è l'**attacco playback**. Per impedirlo si usano password usa e getta, e questo **meccanismo** viene detto **di once** *(nonce??)*.
Si usano allora le chiavi simmetriche: Bob manda un challenge ad Alice, che deve definire il messaggio codificato con la sua chiave. Challenge accepted! Alice gliela rimanda crittografata. Ora il problema è lo scambio delle chiavi.
Si può usare la crittografia a chiave pubblica. Alice manda l'informazione crittografata con la sua chiave privata. Bob nota la chiave pubblica di Alice, decifra il testo e li confronta.
Potrebbe comunque essere usato l'**attacco man in the middle**. Trudy riceve il challenge, lo manda ad Alice, Alice lo cifra e lo manda a Trudy che risponde a Bob con il messaggio mandato da Alice. L'attacco funziona quando Trudy riesce ad essere quanto più trasparente nella comunicazione. Effettivamente la comunicazione tra Alice e Bob funziona comunque bene. In realtà non mi devo fidare della chiave pubblica dichiarata da Alice ma devo andare a prendere la chiave da una CA.