Reti e Sistemi Operativi

DaveRhapsody

3 Ottobre 2019

Indice

Capitolo 1

La rete

La rete è un insieme di nodi, che sono tendenzialmente pc, server, terminali, e dispositivi radio che consentono ai dispositivi senza fili di collegarsi (per intenderci, gli access point wi-fi), ma attenzione!

Qui si parla di wi-fi, non della rete a cella telefonica, non si parla di 4G etc.

1.1 Il modello TCP/IP

Al centro è presente un Router, dispositivo in grado di INDIRIZZARE quelli che sono i pacchetti IP (Internet Protocol). Cos'è un pacchetto?

Pacchetto : E' un "Record/Struct" composto da un campo dati detto payload), ed un campo header, che serve al protocollo di riferimento come etichetta che dice (ad un protocollo) "Questo pacchetto lo puoi leggere". Cosa contiene l'Header?

$$header = \begin{cases} indirizzi \\ livelli\ di\ priorit \\ contatori \end{cases}$$

A che servono i contatori?
Immagina di avere un problema nella rete in cui appare un routing loop (pensate alla lista circolare)

I pacchetti è auspicabile che siano di dimensioni piccine, immaginate un film di dimensioni tendenti al Gigabyte, un pacchetto da un Giga è pressochè (ad oggi) insostenibile, pertanto, si riduce in pezzettini. Soprattutto c'è una probabilità di errore più bassa! (Pensate di mangiarvi una Fiorentina in un solo boccone o di tagliarla prima un pochino, stesso concetto.)

Un altro motivo per cui vengono usati i pacchetti è legato al fatto che inviare un qualsiasi dato occupa un canale. Intendiamoci, se da A a B (terminali) abbiamo un solo cavo, non c'è problema, no?

Ok, ma se A e B sono degli switch aventi 10 pc connessi assieme? Se uno deve caricare un giga di roba la gente deve aspettare che questo finisca MA utilizzando un sistema

a pacchetti, otteniamo che ci sia una sorta di "turno" per ogni pacchetto. Di conseguenza non si intasa la luce.

Il sistema sopracitato si chiama multiplazione o multiplexing statistico, perchè quando io metto tanti flussi di info che viaggiano sullo stesso collegamento, potrebbero accadere congestioni di rete, oppure comunque uno deve aspettare. Ecco i pacchetti dei pacchetti che attendono si chiama buffer. Più hai host di rete, più ti conviene aumentare la dimensione del buffer.

Lo switch Sono dei dispositivi che consentono di costruire la rete con più livelli, proprio perchè ad essi vengono collegati altri dispositivi. Uno switch utilizza connessioni Ethernet, ed una serie di nodi collegati ad uno switch forma una LAN. ATTENZIONE, quando si menziona l'access point, la rete che si forma si chiama WLAN, ovvero (Wireless LAN (Local area network)).

Terminale Il terminale (o host) sarà tendenzialmente un qualsiasi dispositivo che hosta applicazioni che lavorano in rete.

ATTENZIONE la rete è identificata dal Router.
Quando noi diciamo Internet, di fatto, stiamo parlando di un insieme di reti, appunto inter - net, più reti, ergo, più router. C'è un agreement totale che consente ad un terminale in una zona, per poter arrivare ad un'altra.

Vi ricordate il cammino dei nodi? Ecco, stesso concetto.
Questi nodi che si attraversano per passare da una rete A ad una rete B si chiamano autonomous systems, ed il cammino tra mittente e destinatario di un messaggio è identificato dagli autonomous per cui il pacchetto passo.

Più precisamente questi Autonomous sono dei Gateway, gestiti con un protocollo (BGP, Border Gateway Protocol) in grado di regolare le comunicazioni tra questi AS.
Ogni sistema autonomo, a sua volta all'interno avrà un certo insieme di altri router e reti, e quindi sarà presente un altro protocollo (IGP, Internal gateway protocol).

Sicurezza di un AS Ogni sistema autonomo è gestito da un "master" in grado di poter decidere cosa possa passare da quella rete, specifica le famose policy di Routing. La domanda che sorge è.. Chi gestisce gli AS in toto? I gestori della rete, quelli telefonici.

1.2 Cosa influenza le prestazioni di una rete?

Analizzeremo due quantità

Latenza/ritardo La latenza (o ritardo) non è altro che il tempo che intercorre tra l'invio e la ricezione di un pacchetto. Esistono diversi tempi:

- Tempo di processing: E' una quantità infinitesimale che indica il tempo impiegato per elaboare un pacchetto

- Tempo di coda: Per misurarlo devo capire oggettivamente quanto viene usata quella coda (esistono interi studi sulle code, ma a noi ci importa una bellissima ;) (Almeno, per questo corso si intende))

  Per un sistema stabile è auspicabile che il rapporto tra i dati contenibili nel buffer ed il rate di dati trasmessi al secondo sia ≤ 1, detto meglio:

$$\frac{nL}{R} \leq 1$$

Throughput Il through put è il quantitativo di dati che riescono ad essere trasferiti in un determinato quantitativo di tempo (Avete presente Mbps, Gbps, Kbps, quella roba lì)

1.3 Stratificazione

Il concetto di stratificazione consiste nel includere le funzioni di una rete non in un solo protocollo ma in una serie di protocollo che abbiano uno specifico compito, e sia indipendente dagli altri

La filosofia KISS Keep it simple stupid, è la filosofia che regola anche il mondo Unix e Gnu/Linux, ovvero, ogni componente fa il suo mestiere, fa la sua funzione, evitare di mettere in mano ad un componente 10 compiti diversi, per intenderci.

Inoltre dividendo in più componenti io ho possibilità di testare il singolo componente, il singolo protocollo, il singolo problema SENZA, S E N Z A andare a danneggiare o danneggiare gli altri protocolli/componenti!

Stack ISO/OSI Questo è lo stack più preciso ingrandito dell'TCP/IP, visto dal livello più basso al livello più alto

- physical
- data link
- network
- transport
- session
- presentation
- application

1.4 livello Physical

Nel livello fisico viene convertito quello che è il segnale elettrico in una sequenza di 0 ed 1

1.5 Livello Data Link

Si implementa MAC (Medium Access Control), converte quelli che erano degli 0 ed 1 e li fa diventare dei pacchetti (yes, c'è il primo header, cioè quello del physical).

L'indirizzo MAC identifica univocamente un dispositivo in rete, QUALSIASI esso sia. E non solo, mondialmente, N O N esistono due dispositivi DIVERSI aventi lo STESSO Mac address. A che serve oltre questo? Serve al protocollo ip per capire CHI è il mittente o destinatario, e sì, è incluso nell'header

1.6 Rete

E' il livello di internet, l'IP protocol, con indirizzo ipv4 v6 etc.

1.7 Trasporto

E' il livello in grado di trasportare tramite internet quelli che sono i pacchetti applicazione, trasporta i datagram ip, trasporta i pacchetti appunto tramite internet, sono presenti due protocolli (TCP e UDP) che si occupano di fare questo

1.8 Applicazioni

Il nome parla da sè, non lo studieremo in questo corso.

Qui noi si studierà dal data link al transport MA in ordine invertito, ovvero partiamo da "Che cosa vuole l'applicazione?" e poi andiamo a scalare.

Precisazione: Dato un pacchetto in un qualsiasi livello (escluso lv application), quando viene aggiunto l'header, il nuovo pacchetto potrà esser letto SOLO dal protocollo del livello superiore!

Su questo stesso piano, se ho un pacchetto con già l'header aggiunto, questo potrà essere spacchettato SOLO da un protocollo del livello inferiore.

Ogni livello offre servizio al livello immediatamente superiore od inferiore in base al verso del flusso dati.

La magica teoria (o ricetta) dell'incapsulamento di Dave Per spiegare meglio cosa accade, immaginatevi una fetta di prosciutto, bene, questa fetta di prosciutto rappresenta il flusso elettricoche vien gestito dal livello fisico.
A questo punto, immaginatevi due fette di pane che si chiudono sulla fetta di prosciutto. Benissimo, il Data Link ha **INCAPSULATO** il prosciutto ed ha creato un panino (Pacchetto)!

Il data link passerà questo panino al Network , e lì ci saranno altre due fette di pane che includeranno il nostro panino. Ed in questo modo abbiamo ottenuto un panino dentro ad un altro panino! Ossia, un pacchetto incapsulato in un pacchetto che semplicemente aggiunge un header (o intestazione) ;) Ma torniamo seri adesso.
Premete qui per vedere un video corso che spiega con precisione l'incapsulamento.

Capitolo 2

Livello Trasporto

Il livello di trasporto è il livello che riceve messaggi dal livello application ed ha come compito quello di mettere in comunicazione end-to-end due nodi. Nel caso del livello di trasporto non si parla di pacchetti ma di Segmenti (Dall'esempio sopra riportato sì, è stato incapsulato il pacchetto del lv network).

Quando viene analizzato il segmento si effettua la DEMULTIPLAZIONE (Demultiplexing), ovvero si analizza l'header del segmento per vedere a quale applicazione sarà destinataria di un determinato messaggio.

Cosa contiene un segmento di trasporto?

- Numero di porta: presente in ogni segmento, rappresenta una applicazione e dal punto di vista dell'applicazione è come se fosse un punto di accesso, detto anche SOCKET.

  NON CONFONDIAMOCI Il livello transport vede una porta, mentre il livello applicazione vede un socket. All'atto pratico c'è una syscall che letteralmente può attivare qualsiasi socket.

- Protocollo Transport TCP o UDP.

Che differenza c'è tra TCP e UDP? UDP è quello che usate negli streaming di Rojadirecta , in cui pur di vedere la vostra squadra (che ovviamente è la Fiorentina, vero? <3) non vi preoccupate di perdere qualche dato (tipo immagine un po' sgranata ogni tanto e simili).

Infatti UDP se ne frega se un pacchetto è arrivato, è usato per comunicazioni anche per esempio le voice chat, roba di questo genere, poichè **NON** effettua alcun controllo sulla correttezza dei messaggi, diciamo che lui ti invia roba, se non ti arriva non gli importa, va avanti, perchè qui è importante che si sia il più aggiornati possibile.
Al contrario TCP effettua questi controlli, perchè con TCP si garantisce la correttezza (sia dei dati stessi che dell'ordine in cui arrivano), pertanto potrebbe essere un problema.

Il segmento può avere sia TCP che UDP, e nel caso di UDP si ha un IP + Porta di

destinazione, o meglio

$$PacchettoUDP = \begin{cases} SourcePort: \ Porta\ sorgente \\ DestinationPort: \ Porta\ di\ destinazione \\ Lunghezza: Indica\ la\ lunghezza\ totale\ del\ segmento\ (+payload \\ dei\ livelli\ inferiori) \\ Checksum: \ Verifica\ la\ correttezza\ di\ trasmissione \end{cases}$$

Supponiamo di avere un web server, (quello che consente di vedere una pagina web) , la porta d'accesso è la stessa ma hai più utenti con la stessa pagina

Nel caso di TCP invece un segmento è composto non solo dall'IP + la porta di destinazione, ma è composto in questo modo:

$$\begin{cases} IP_A \\ IP_B \\ PORT_A \\ PORT_B \end{cases}$$

A e B in questo caso sono tendenzialmente sorgente e destinataria, ed il calcolo del checksum precedente è effettuato direttamente dal livello Transport stesso.

Attenzione, nell'introduzione abbiamo anche detto che già nel datalink abbiamo dei controlli, pertanto ogni livello può svolgere la correzione dei pacchetti, pertanto non si attiva quasi mai, perchè Transport dà per scontato che Network e Datalink gli abbian passato qualcosa di corretto.

Il codice di implementazione del controllo è tipo di migliaia di righe di codice per correggere o prevenire errori, ed è giustissimo così. Contiamo che ormai TCP e UDP non si possono rompere, sono più che sicuri.

2.1 TCP

Il concetto è che si vuole trasferire dati in modo affidabile, o meglio, si vuole fare in modo che i dati arrivino a destianazione e siano corretti in generale.

Il trasferimento dati affidabile dipende da una serie di fattori, e si verifica quando ho una rete con le seguenti caratteristiche:

1. Nessuna perdita

2. Nessun errore

3. I dati son corretti e presenti tutti quanti

Nel caso avvengano errori si può ricorrere a diverse metodiche di soluzione, come ad esempio la tecnica di ritrasmissione. Ossia, non ti è arrivato il mio pacchetto? Ok, te lo rimando. Nell'atto pratico è una cosa del tipo

$$\begin{cases} if(ricevoAcknoledgment)\ then\ mandoSucessivo(); \\ else\ if(ricevoNotAcknoledgment)\ rimandoPacchetto(); \end{cases}$$

Quello che chiamo not acknoledgment (che da ora chiameremo solo ACK) viene inviato SE il pacchetto è non corretto, per esempio per il checksum.

E se il pacchetto poi è lo stesso? Ne ricevo due? No. I pacchetti sono numerati hanno un ordine, pertanto

$$\begin{cases} if(ricevoACK)m+1; \\ elsem \end{cases}$$

con m che è il "pezzetto" successivo o precedente, è importante che si capisca questo.

Ora cerchiamo di integrare, è necessario che il sistema mi mandi indietro sì se ha ricevuto ma ANCHE la posizione a cui è arrivato! (Sistema PAR, positive acknoledgmente with retransmition).

Quindi mi mandi un messaggio e io ti rispondo dicendo ok messaggio, tu mi mandi il due, io rispondo ok due, tu mi mandi tre io rispondo ok due, mi rimandi tre.

Perdite Sì, ok, ma se uno non mi riesce a rispondere? Si imposta un semplice timer la cui scadenza implica che ti rimando il pacchetto. Un po' come quando dici una cosa con la musica alta a uno e lui ti guarda tipo fisso... Passa tempo, e poi ripeti la cosa.

E se io continuo a rispondere? Lui che fa continua come un pazzo a ripetere la stessa cosa come Ciuchino che chiedeva a Shrek ogni 2 millisecondi "Siamo arrivati?"? NO! C'è un contatore che si aziona ad ogni mancata risposta, una volta che arriva ad un tot, chiudo la comunicazione.

$$\begin{cases} if(timerGoesOff)\ \ trasmetti(messaggio_i)\ AND\ setTimer(0); \\ else\ if(ACK_i)\ trasmetti(messaggio_{i+1}\ AND\ setTimer(0)); \\ else\ trasmetti(messaggio_i)\ AND\ setTimer(0); \end{cases}$$

ATTENZIONE Supponendo un timer troppo corto, accadrebbe che il destinatario non fa in tempo a rispondere, e quindi il mittente rifà la domanda prima di ricevere la risposta, hai già due domande con una risposta.

All'atto pratico questo problema esiste davvero, è un problema reale, pertanto si deve riuscire a capire il modo in cui il timer vada adattato alle condizioni di rete, che a seconda del traffico può essere più lenta o anche più veloce.

2.1.1 I tempi di una connessione

Riprendiamo il concetto di latenza (tempo che intercorre tra invio e ricezione), bene, quello è concettualmente il tempo di trasmissione, da qui possiamo calcolare che

$$utilizzoRete = \frac{\frac{L}{R}}{latenza + \frac{L}{R}}$$

in cui $\frac{L}{R}$ Ha R che è la velocità di trasferimento dei pacchetti, mentre invece L sarebbe la lunghezza , ossia: $\frac{L}{R} = \frac{Lunghezza}{Velocita' \ trasmissione}$ E la latenza... E' la latenza.

2.1.2 Protocolli a Finestra Scorrevole (Sliding Windows)

Io trasferisco un insieme di w pacchetti, i quali impiegheranno $w * \frac{L}{R} \geq \frac{L}{R} + RTT$ tempo ad esser trasposti, pertanto ne esce che

$$w \geq \frac{RTT \cdot R}{L} + 1$$

Questo ovviamente in mancanza di errori e perdite.

Osservazione: Stiamo ragionando su sistemi concettualmente indistruttibili, quindi diciamo che non si è quasi menzionata la possibilità che un host possa disconnettersi.

2.1.3 Go Back N

Poniamo caso che io trasmetto 5 messaggi: $m_{1,...,5}$ e mi arriva l'ok di ricezione dall'1 al 3, malgrado li abbia trasmessi tutti e 5, pertanto ciò che accade è che rinvierò dal messaggio 3 in poi.

2.1.4 Selective Repeat

Traccio la vita di ogni pacchetto individualmente, ossia ritrasmetto solo il pacchetto mancante, il problemino è legato al fatto di avere tanti timer, ma oggi c'è da considere che abbiamo processori nell'ordine dei Ghz, no problem.

Quando all'epoca il massimo processore esistente era di massimo 10 Mhz, insomma chiaro che ora essa sia la migliore delle soluzione. Ma cosa richiede più precisamente l'SR?

- Ack individuali per ogni pacchetto
- Un timer per ogni singolo pacchetto
- Buffer sul ricevitore (E' un array di booleani all'atto pratico)

Ci sono dei casi in cui ci sono interazioni tra numero di pacchetto e dimensione finestra, ma sono problematiche di cui non approfondiamo perchè non ci interessa siccome non può praticamente mai capitare

2.2 Formato del pacchetto

- 20 byte
  - 16 bit Source + 16 bit destination port
  - 32 bit Sequence Number
    Se i numeri di sequenza contassero il numero di segmenti, pensate averne 2^{32} E INFATTI, vengono contati in Byte, quindi il numero di sequenza fa riferimento al byte
  - 32 bit ACK Number
  - 8 bit di cui 4 indicano l'header length e gli altri 4 son inutilizzati
    16 bit di recive window, ovvero quanti bit posso accettare
  - 16 bit di checksum + urgent data (in seguito vedremo le funzioni dei vari flag)
- Payload (χ Byte, perchè la dimensione è variabile, dipende dal carico di dati che trasporta)

Diventa ragionevole contare i Byte e non i segmenti perchè nel periodo in cui è nato TCP il costo per la banda di rete era elevato, quindi il singolo byte faceva la differenza in una trasmissione.

2.2.1 Le Flag

- CWR: Congestion window reduced
- ECE: End to End congestion
- U: Urgent Data
- P: Push
  Trasmetti senza far funzionare i tuoi algoritmi che assemblano un tot di dati.
- ACK: Ti dice se è un ACK
- RST: Reset della connessione per via di un problema nella sequenza
  oppure il ricevitore ti chiude la porta e non apre la connessione
- SYN: Sincronizzazione, semplicemente apre la connessione
- FIN: Chiude la connessione

2.2.2 Gestione dei riscontri

Devo inviare dei pacchetti, e quindi devo riscontrarli, come si gestisce il valore dei puntatori?

A invia un messaggio con Seq = 10, ACK = 57, [ciao] stiamo intendendo un messaggio avente questi 3 parametri principali, li possiamo mettere pure a sistema così, per intenderci meglio:

$$messaggio = \begin{cases} Seq = 10 \\ ACK = 57 \\ [ciao] \end{cases}$$

Cosa mi aspetto come risposta? Beh io sto comunicando che siamo al 57_{esimo} ACK e al 10 byte di sequenza, pertanto tu mi aspetto che mi si dica UEH sono arrivato a 57 e mi risponde:

$$messaggio = \begin{cases} Seq = 57 \\ ACK = 14 \\ [bella] \end{cases}$$

Il concetto è che la sequenza della risposta coincide con l'ack della "domanda", sembra sbagliato, no? Bene, se ti comunico che sono a λ mi aspetto che tu mi risponda che sei arrivato alla sequenza λ. Pertanto in risposta accadrà che la sequenza dovrà coincidere con l'ack precedente.

Se all'inizio si era al segmeto 10, verrà risposto che si è al segmento 57 MA quando verrà inviato di nuovo si riscambia, diciamolo in modo più sensato:

$$EstimatedRTT = (I - \alpha) \cdot EstimatedRTT + \alpha \cdot SampleRTT$$

Con α = 0,125 nella maggior parte dei casi (E' il valore tipico). Ed rtt che sarebbe il rounded trasmission time, cioè in parole povere il tempo che ci si impiega a trasmettere.

Tutto va in funzione di come cambia l'RTT nel tempo, conviene avere un RTT che nel tempo rimane lo stesso, non conviene avere un RTT troppo ballerino

$$DevRTT = (1 - \beta) \cdot DevRTT + \beta \cdot |SampleRTT - EstimatedRTT$$

Dove dev sta per deviazione (O margine di sicurezza) e tipicamente β=0,25
Infine definiamo quello che è il $Timeout = EstimatedRTT + 4 \cdot DevRTT$

2.2.3 Passo 2: Ritrasmissione

Quando non va tutto bene potrei dovere ritrasmettere dei segmenti, perchè? Generalmente per via di o timeout oppure ack duplicati.

Il nostro trasmettitore fondamentalmente trasmette segmenti, nel caso il ricevitore non riceve allora questo ritrasmette, e nelle puntate precedenti si è visto cosa può accadere (Timer e ACK duplicati), ed infine resetta il timer.

ATTENZIONE NON esiste un preciso modo per cui possano verificarsi dei problemi di ricezione, possono accadere miliardi di differenti inconvenienti, per dire anche il fatto che ti arriva un segmento a metà, l'altro solo inizio e fine

2.2.4 La perdita dei pacchetti

Se il trasmettitore invia un pacchetto che però non arriva, il ricevitore manderà l'ack precedenti dicendo che manca un pezzo, possono esserci diverse situazioni. (Si ok, perchè manda un altro ACK? Semplicemente perchè potrebbe essersi perso pure quello, non funziona una s**a in questa rete.).

2.2.5 Pacchetti che arrivano non in sequenza

Abbiamo visto il selective Repeat, ma c'è anche un altro modo un po' diverso che ha un unico timer (soluzione che impiega più ram che processore) perchè sul trasmittente avremo un array di ack che dice dove e cosa è stato ricevuto. Esempio:

Io invio da 1 fino a 5, mi rispondi 4, io ti mando poi 6, 7, 8, allora in base a quale elemento dell'array non risulta true io ti mando quella zona. Ad esempio ho true gli ack da 1 a 8 MA manca proprio il 5. Allora semplicemente mando il 5. OVVIAMENTE ricordiamoci che il timer è utilizzato anche in questo caso.

Potrebbe anche accadere di avere dei riscontri cumulativi o duplicati. Il cumulativo è quando tu mi invii 1, 2 e 3 e io rispondo solo 3, tu dai per scontato che io abbia ricevuto i precedenti.

2.2.6 La receive Window

In un meccanismo a finestra ho una.. Finestra in cui posso trasmettere. Cosa determina la suddetta? In pratica controllo la quantità di dati che posso trasmettere. Se in capo mi entrano 3 parole al minuto durante le lezioni, e i prof ne dicono 500 leggendo le slide, ne perdo un patrimonio in una lezione, stessa cosa per le reti.

Siccome in una rete è meglio evitare questo problema si implementano la ReceiveWindow e la CongestionWindow (Finestra di ricezione e congestione), e il valore che dovremo assumere sarà $W = min\{CongWindow, RcWindow\}$, ricordando sempre la formula che la dimensione dei pacchetti da trasmettere è $\frac{W}{RTT}$

2.3 Inizio della connessione: HandShake a 3 vie

Handshake vuol dire letteralmente stretta di mano, tipo saluto, prendiamo per presupposto di avere due host che chiamiamo A e B

Questi non si conoscono, pertanto A manda a B un messaggio con

$$Ascrive: \begin{cases} SYN \\ Seq = \kappa \\ [] \end{cases}$$

$$Brisponde : \begin{cases} SYN \\ Seq = \psi \\ ACK = \kappa + 1[] \end{cases}$$

Che tradotto in termini pratici è tipo:
A: OH INIZIO A κ B: OH HO RICEVUTO $\kappa + 1$, IO SONO A ψ

$$Areplica : \begin{cases} SYN \\ ACK = \psi + 1[] \end{cases}$$

Che è traducibile in "OK HO RICEVUTO $\psi + 1$"

2.4 Fine della connessione

$$Ascrive : \begin{cases} FIN \\ Seq = \kappa \\ [] \end{cases}$$

$$Brisponde : \begin{cases} FIN \\ ACK = \kappa + 1[] \end{cases}$$

Se però B non ha ancora finito, allora gli manda solo un ACK, mentre se ha finito anche lui, semplicemente si chiude la connessione

2.5 Congestione di rete

Avviene nel momento in cui viene inviata una quantità di pacchetti SUPERIORE a quella che normalmente può essere ricevuta, che è come quando ad Analisi venivano scritte troppe cose alla lavagna e non si riusciva a starci dietro, prima o poi avvengono perdite di dati.

Per evitare le congestioni ci sono una serie di meccanismi. Perciò iniziamo a definire una nuova quantità: MSS = Maximum segment Size, ovvero dimensione massima di un segmento.

2.5.1 AIDM: Additive Increase Mult. Decrease

Invio un messaggio, tu rispondi ok, poi ne mando due, rispondi due, poi tre e rispondi tre. Progressivamente cresce la finestra, arriverò ad un valore λ per cui non riesci a rispondere e c'è una perdita, ma può esser di due tipi:

1. Loss: Perdita
2. Timeout: Ci impiego troppo tempo a passare tutto

Il peggio tra i due è il timeout, perchè manca proprio una risposta, mentre se ho delle perdite almeno c'è comuniazione! Perciò cosa succede, appena c'è una perdita viene dimezzato il numero di dati che invio. Se la prima perdita si verifica a δ allora ripartirò da $\frac{\delta}{2}$.

Notiamo subito che questo metodo è estremamente lento, perciò c'è un secondo metodo

che fa la stessa cosa ma invece di incrementarsi di uno si incrementa esponenzialmente (1, 2, 4, 8 ,16, 2^n)

Questo meccanismo esponenziale si chiama Slow Start e funziona allo stesso modo poi per quando si dimezza tutto.
Quello che abbiamo visto è come fare per trovare la quantità di dati massima che è possibile trasmettere SENZA PERDITE.

Come si gestisce il timeout? In questo caso va abbassata di tanto la finestra, perchè in questo caso non viene nemmeno ricevuto il pacchetto, scade il timeout ancor prima di aver rimandato indietro il pacchetto perciò come si agisce?

Il funzionamento è identicop a quello per la perdita di pacchetti, si ragiona incrementando esponenzialmente il numero di pacchetti, poi si arriva al timeout, si dimezza, e si riparte da lì MA questa volta incrementando di 1 alla volta.

Questo incremento di un'unità per volta viene chiamata Congestion Avoidance, e (come da nome) va a risolvere le congestioni.

Ok, ma perchè non fare direttamente 1 alla volta? Perchè il tempo è una risorsa essenziale, e ragionare di pacchetto in pacchetto richiederebbe troppe risorse, semplicemente in questo modo si risparmia tempo, tutto lì.

Capitolo 3

Sistemi operativi

Un sistema operativo è un software, un programma, che agisce da intermediario tra l'utente ed il suo computer.

Molto semplicemente anche solo per scrivere questi appunti, e voi per visualizzarli stiamo usando un sistema operativo. Ma andiamo per livelli

1. Utente: Esseri umani, OPPURE anche altri computer, l'utente può essere qualcuno o qualcosa che necessità l'intervento del computer fondamentalmente.

2. Applicativi/Application programs: Sono il software applicativo, nel senso che si applicano per risolvere un problema pratico, ad esempio per dire programmi di editing foto, oppure il browser, cioè fondamentalmente estendono le funzioni fattibili dalla macchina

3. Sistema operativo : Controlla e gestisce l'hardware al servizio dei nostri applicativi, fa appunto come si diceva da intermediario

4. Hardware: Ci si rifà al modello di Von Neumann, CPU Ram e I/O, sempre lì si ritorna, tutte li risorse messe a disposizione dalla ferraglia per intenderci

Perchè non vogliamo che gli applicativi usino direttamente l'hardware ma ci spariamo sopra un OS che gestisca? Per due aspetti.

1. Da un lato astrae le risorse hardware, presentando una macchina estesa più facile da programmare (Files invece di blocchi di dati)

2. Dall'altro, gestisce le risorse hardware del computer, assegnandole ai programmi in maniera equa ed efficiente e controllando che questi le usino correttamente

    - Nasce dalla necessità di eseguire più programmi per più utenti sulla stessa macchina

3.1 Storicamente

Per quanto concerne l'hardware rimangono buone tutte le varie nozioni ricevute ad Architettura degli Elaboratori, lì sono state approfondite meglio da questo punto di vista (giustamente.)

Il sistema operativo oggettivamente fa in modo che le risorse dell'hardware siano gestibili per eseguire più di un software alla volta sul pc, perchè di fatto alla volta si esegue un solo programma MA ci vien data l'impressione che siano più di uno alla volta.

L'esigenza di gestire le risorse è nata assieme ai primi computer fondamentalmente, quelli a valvole termoioniche, CIOE' OH SON DELLE LAMPADINE UN PO' PARTICOLARI. Come già avrete sentito almeno 200 volte, i primi computer, aventi un miliardesimo della potenza di calcolo di un nostro smartphone, eran grandi quanto una stanza, e venivano programmati direttamente staccando e attaccando i cavi in base a cosa si dovesse fare... Immaginatevi me a prendere gli appunti saltando da una parte all'altra per attaccare e staccare spine. Altro che jogging e palestra.

Solo che le valvole si guastavano, era più il tempo per ripararli che quello per usarli MA eran 20000 mila volte più veloce di fare i calcoli su C A R T A. (Computer's rules)

Dopo l'invenzione dei transistor i computer diventano più affidabili, inoltre, i transistor non si fulminano, son più rapidi, 10 volte più piccini delle valvole. Insomma, progressivamente i computer son diventati più piccini, veloci, e meno costosi (pensate agli smartwatch tipo MiBand per intenderci).

In seguito all'aver capito che programmare usando spine è un po' difficoltoso, tipo mezza giornata per fare a = c + b, per intenderci, allora si è iniziato a programmare con l'elaborazione **batch**, con programmi monitor per il caricamento ed esecuzione dei job (job control languages), (qua già si usavano le schede perforate), e qui nascevano i linguaggi ad alto livello.

Progressivamente poi c'è stato il passaggio ai circuiti integrati, infatti IBM con la magica linea 360 cerca di avere un'unica linea di computer adatta sia per ditte commerciali che per calcoli scientifici E NON SOLO perchè avevan creato dei computer che potessero essere compatibili tra loro ma differivano per la capacità di calcolo

Ci son due problemi

1. Lo stesso software doveva funzionare su tutti i computer della linea
2. Occorreva gestire le risorse della macchina in maniera efficiente in tutti gli scenari applicativi

3.2 Tecnica Multiprogrammativa

Molti programmi caricati in memoria contemporaneamente, che tradotto significa che quando un programma è impegnato nell'i/o il processore passa ad eseguirne un altro

3.3 Spooling

L'I/O viene spostato in un buffer in autonomia dal processore, perchè appunto il processore interagirà con il buffer (si fa prima) anzichè con la periferica. (Le code di stampa sono un ottimo esempio)