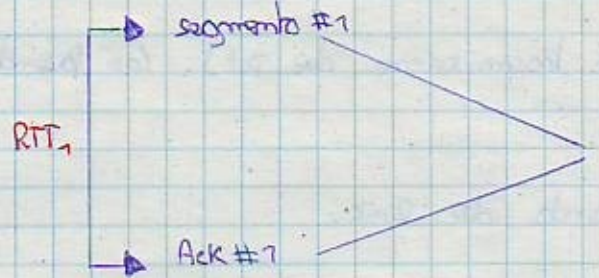
(8h)

Una rete poi può improvvisamente entrare in congestione e rallentare per cui un timeout certo non mi permette più di trasmettere perché B stesso continua a scattare. I timeout costanti poi sono da evitare e quindi si passa ai timeout ADATTATIVI. Il TCP misura costantemente l'RTT effettivo della rete.



RTT_1
RTT_2 si ha cioè un campionamento
⋮ di RTT.

Il TCP considera la rete quindi come una BLACK-BOX ed inoltra segmenti misurando ogni volta, dopo quanto arriva e acknowledge. Il TCP trasmittente mantiene una variabile denominata smoothed round trip time (SRTT) che altro non è che una stimatore statistico del valore medio corrente di RTT(t). Ad ogni acknowledge ricevuto il TCP calcola il nuovo valore di SRTT che è:

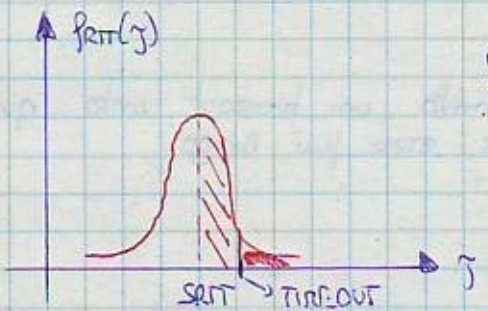
$$SRTT = (1-\alpha)\,OLD\_SRTT + \alpha\, LATEST\_RTT$$

con $\alpha = 1/8$.

Questa è una stima a media mobile. Quindi:

sulla SRTT si cerca di evitare variazioni troppo brusche del tempo.

Perciò SRTT è un filtraggio passa basso di RTT(t) e ne stima il valore medio. Ma tutto ciò non ci basta per il calcolo di timeout. In t^*, $RTT(t^*)$ non è costante, ma è una variabile casuale costante con media $\simeq$ SRTT e distribuita con una certa densità di probabilità:



{ Se TIMEOUT = SRTT non va bene
{ Se TIMEOUT > SRTT va bene.

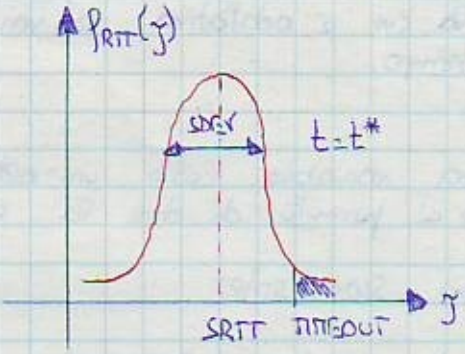
Devo quindi avere uno stimatore della variazione, si stima perciò la deviazione standard. Sia:

SDEV = Smoothed Deviation

Per ogni acknowledge ricevuto, si ha:

$NEW\_SDEV = \frac{3}{4} OLD\_SDEV + \frac{1}{4} DEV$ con $DEV = |LATEST - RTT - OLD\_SRTT|$

DEV va a contribuire alla varianza e quindi alla deviazione standard. Quindi:

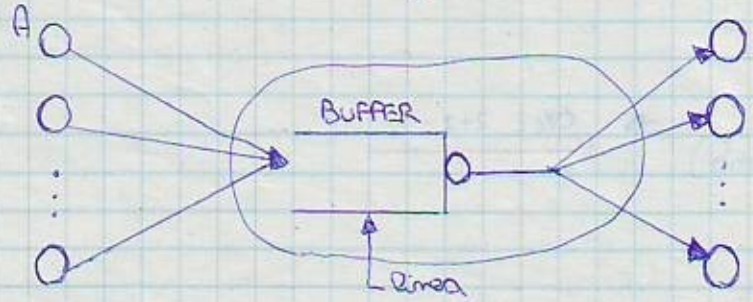


$TIMEOUT = SRTT + K \cdot SDEV$

Siccome più o meno si ha una gaussiana si ha che:

$P_{ACK\_IN\_RITARDO} \cong 10^{-K}$ con $K = 3, 4, 5 \ldots$

Questo algoritmo va sotto il nome di ALGORITMO di VAN JACOBSON che gestisce in modo dinamico ed adattivo i timeout. Questi vengono assegnati in maniera quasi ottima. I timeout vengono fissati in modo da ottenere una probabilità prefissata di superamento dei timeout anche se tutto va andato bene. Quindi Jacobson prende la misurazione e la stima del Round Trip Time della rete da cui deriva il calcolo dei timeout. Quando si perde un segmento o un acknowledge non si può applicare l'algoritmo di Jacobson. In caso di perdita del segmento o dell'acknowledge scatta un timeout e allora si applica l'algoritmo di Karn. Tale algoritmo ci dice che ad ogni trasmissione di un segmento si raddoppia il relativo timeout. Questo vale fino ad una soglia massima di ritrasmissioni, dopo di che si abbatte la connessione. Analizziamo ora il controllo della congestione nel TCP. Consideriamo:



Il controllo di flusso abbiamo visto che serve ad impedire che per esempio A sovraccarichi A'. Si noti che anche la rete può essere sovraccaricata.

Può però anche succedere che la linea sia in sovraccarico, cioè che la capacità della linea di ingresso sia maggiore della capacità della linea di uscita. Il TCP cerca di impedire o controllare gli stati di congestione della rete. Ma come fa a fare ciò? Noi sappiamo che per quanto riguarda il controllo di flusso si ha:

$\forall t \Rightarrow$ numero byte che TCP può inviare $\leq RW$ e quindi RW domina il controllo di flusso.

Per quanto riguarda il controllo della congestione esiste una finestra chiamata congestion window (CW) nel solo TCP trasmittente. Il numero di byte che il TCP può inviare in un dato istante di tempo è minore o uguale al minimo tra $RW(t)$ e $CW(t)$.

(Min(RWR(t), CWR(t))). Quindi la CWR controlla la congestione della rete. La CWR è adattativa e si adegua alla velocità della rete. Sostanzialmente la CWR è una variabile numerica e non un buffer. Ma come viene gestita la CWR? La sua gestione è piuttosto complicata. Innanzitutto il TCP funziona in due possibili regimi per quanto riguarda la CWR:

1) SLOW START
2) CONGESTION AVOIDANCE

NB: la CWR è adattativa e varia nel tempo.

Abbiamo visto che la CWR è sostanzialmente una variabile. Esiste un'altra variabile denominata SSTHRESH (Slow_Start_Threshold) che ci permette di dire la seguente cosa:

- se CWR < SSTHRESH ⇒ siamo in regime SLOW_START
- se CWR > SSTHRESH ⇒ siamo in regime CONGESTION_AVOIDANCE

Supponiamo ora che per t=0 inizi la connessione e supponiamo che CWR = 1 segmento. Si ricordi che la dimensione della CWR si misura in byte, anche se noi per comodità la misuriamo in segmenti. All'inizio si ha che CWR = MSS byte. Quindi il TCP trasmette 1 segmento perché MIN(RWR, CWR) = MIN(RWR, 1) = 1. Perciò per t=0 il TCP invia un segmento di lunghezza massima. Si noti che il TCP potrebbe inviare più segmenti, ma per il controllo di congestione ne invia solo uno. Quindi il TCP inizia in SLOW_START. Poi arriva un acknowledge del segmento e per ogni byte riscontrato la CWR aumenta di 1 byte la propria dimensione. Quindi:

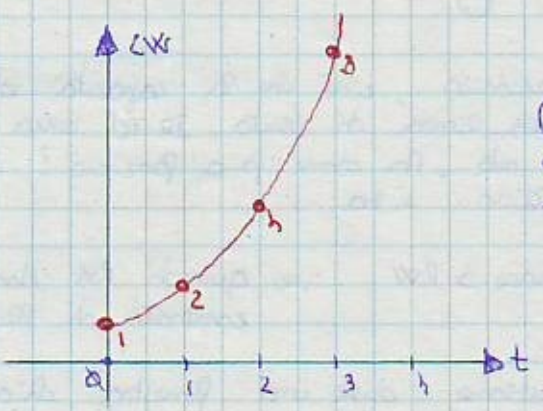
LATO TRASMITTENTE

CWR = 1
TRASMISSIONE = 1 segmento
RICEZIONE = 1 ACKNOWLEDGE (1 segmento) ⇒ CWR = 1+1 = 2 e successivamente il TCP può trasmettere MIN(2, RWR) = 2 se RWR è per ipotesi grande.

Al passo successivo:

CWR = 2
TRASMISSIONE = 2 segmenti
RICEZIONE = ACKNOWLEDGE (2 segmenti) ⇒ CWR = 2+2 = 4. ...

Quindi graficamente si ha:



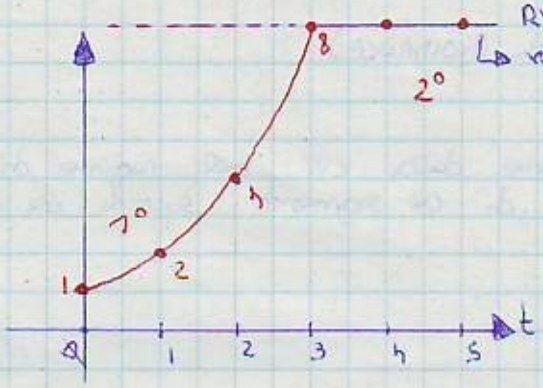
Quindi CWR aumenta esponenzialmente e aumenta esponenzialmente anche la sua velocità di trasmissione.

REGIME DI SLOW_START

Vediamo i limiti di tale funzionamento:

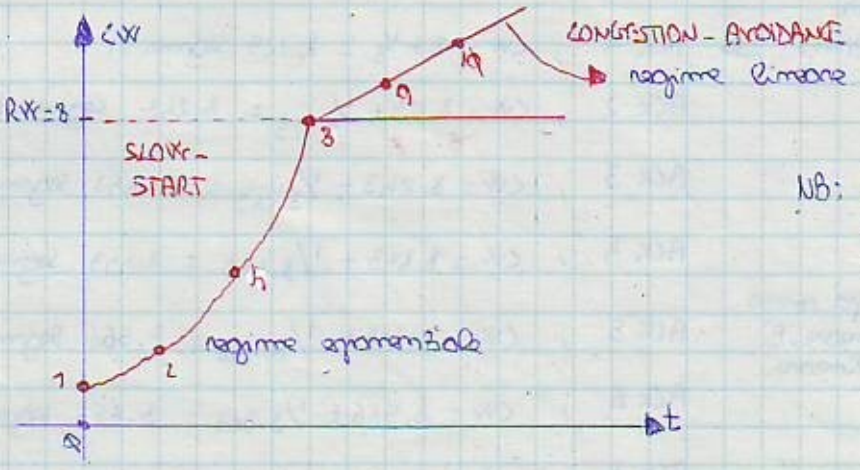
- Immaginiamo la congestion window diventa più grande della receive window.

Perciò:



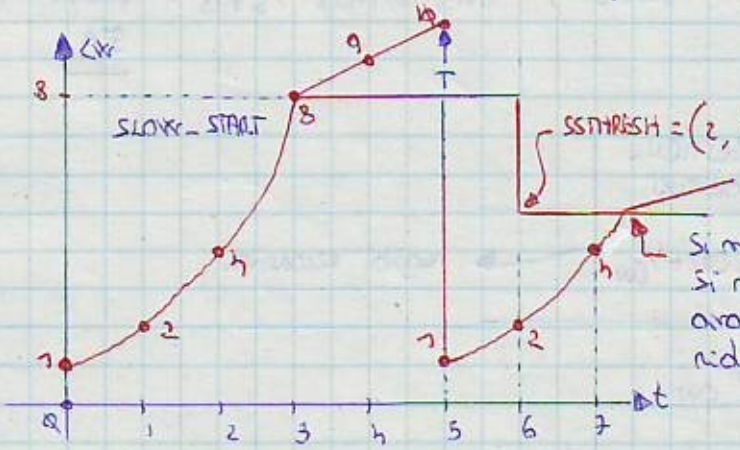
È successo che nella seconda fase è intervenuto il controllo di flusso.

Ma cosa succede quando siamo in congestion - avoidance?



NB: La CW aumenta di un segmento se si riceve un ACKNOWLEDGE (CW segmenti) in fase di CONGESTION - AVOIDANCE.

Vediamo ora cosa succede quando si perdono i segmenti.



- T: timeout

Si noti che la rete è piuttosto congestionata. Si noti inoltre che non siamo che si va avanti, incontrando troppi altri timeout, si riduce la SLOW - START.

La variabile FLIGHTSIZE mi indica il numero di segmenti che erano stati trasmessi ma che non sono stati riscontrati quando è scattato il timeout. Si noti che il TCP ogni volta aumenta la velocità di trasmissione e quindi prima o poi congestiona la rete. Quando scatta il timeout, si assenti, il TCP ricomincia a ritrasmettere da capo. Il numero di byte trasmessi è rappresentato dalla area della curva. Nel caso precisino quando il TCP trasmette il secondo pacchetto, la rete è già in congestione. Prima che il TCP ritorni la sua velocità passa parecchio tempo. Vediamo ora come viene gestita la CW.

(88)

Se ricevo un acknowledge (1 segmento), la CW aumenta di $1/CW$. Questa rappresenta un'approssimazione della vera regola. Quindi in termini di byte si ha:

$$CW = CW + 1/CW \qquad \text{(IN CONGESTION-AVOIDANCE)}$$

Sottolineiamo che stiamo analizzando la gestione della CW nel regime di congestion-avoidance. Se ricevo un acknowledge di CW segmenti succede che essendo in regime di CONGESTION-AVOIDANCE si ha:

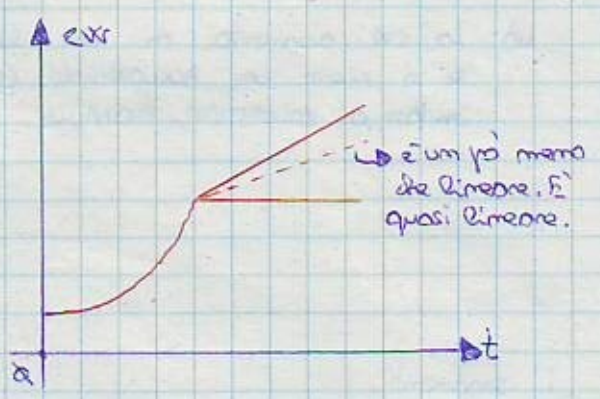
$$CW \cong CW + 1$$

Per esempio:

CW = 8
TRASMISSIONE = 8 segmenti
RICEZIONE = ACK(8 segmenti) ⇒

ACK 1, $CW = 8 + 1/8 = 8.125$ segmenti
ACK 2, $CW = 8.125 + 1/8.125 = 8.243$ segmenti
ACK 3, $CW = 8.243 + 1/8.243 = 8.369$ segmenti
ACK 4, $CW = 8.369 + 1/8.369 = 8.488$ segmenti
ACK 5, $CW = 8.488 + 1/8.488 = 8.566$ segmenti
ACK 6, $CW = 8.566 + 1/8.566 = 8.683$ segmenti
ACK 7, $CW = 8.683 + 1/8.683 = 8.798$ segmenti
ACK 8, $CW = 8.798 + 1/8.798 = 8.8911$ segmenti $\cong 9$.

Quindi in realtà:



Quindi riassumendo:

GESTIONE CW

	SLOW-START	CONGESTION-AVOIDANCE
ACK(1)	$CW = CW + 1$	$CW = CW + 1/CW$ → regola ricorsiva
ACK(CW segmenti)	$CW = 2 \cdot CW$	$CW \cong CW + 1$
ACK(s segmenti) con: $1 \le s \le CW$	$CW = CW + s$	$CW \cong CW + s/CW$
	$CW < SSTHRESH$	$CW \ge SSTHRESH$

89

Consideriamo ora: IPOTESI: RETE A VELOCITA' INFINITA

A O ——— c = ∞ ———> B O

DATI:

RTT = 1 ms costante

TEMPI DI TRASMISSIONE = 0

MSS = 1 Kbyte

TIMEOUT BASE: 2 ms e raddoppia ad ogni ritrasmissione

SSTHRESH (t=0) = 8 Kbyte

RW (t=0) = 5 Kbyte ma poi B ogni tanto cambia la sua dimensione.

Ipotesi:
- RW (7 ms) = 12 Kbyte
- RW (15 ms) = 10 Kbyte
- RW (20 ms) = 5 Kbyte
- RW (29 ms) = 10 Kbyte
- RW (35 ms) = 12 Kbyte
- RW (45 ms) = 2 Kbyte
- RW (47 ms) = 10 Kbyte

CW (t=0) = 1 Kbyte e la rete ha ogni tanto dei malfunzionamenti. Essa va giù tra [10 ms, 11 ms), [30 ms, 31 ms), [39 ms, 40 ms)

⇓

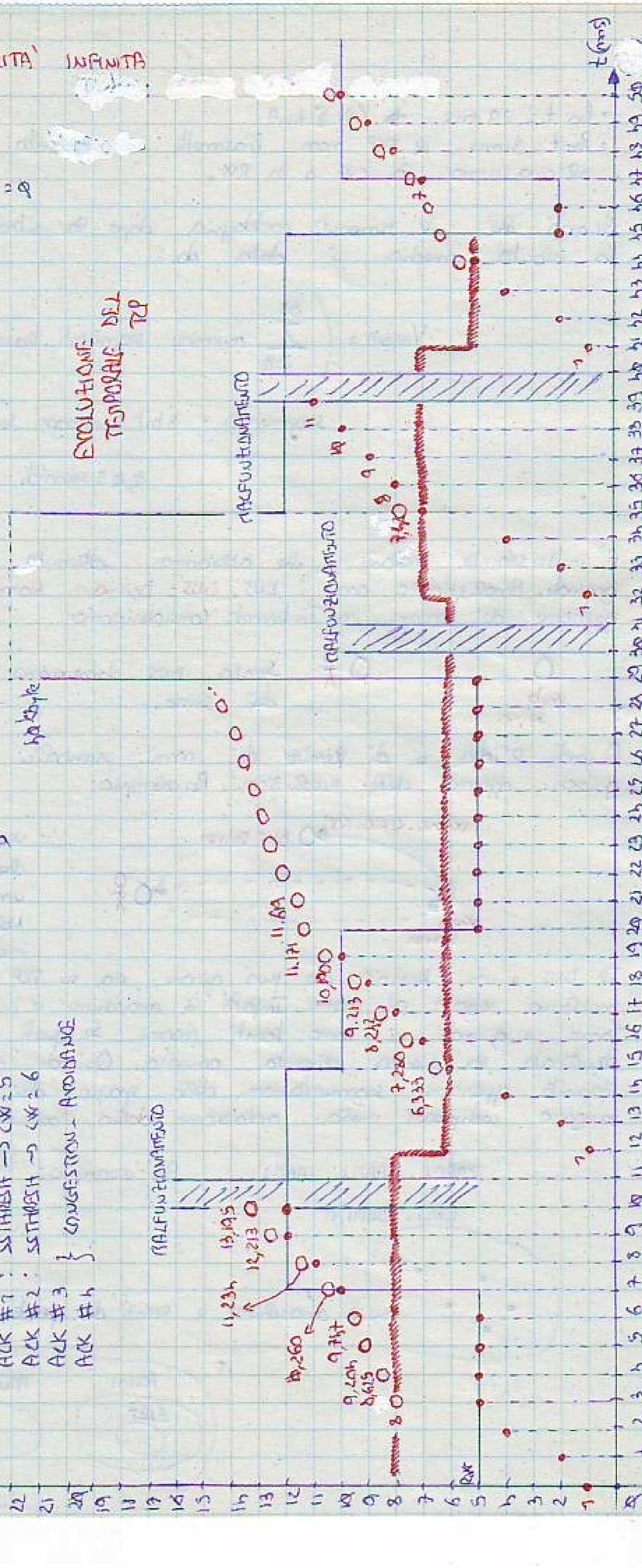
perdite di segmenti

Vogliamo sapere qual'è l'evoluzione del TCP da 0 a 50 ms e la velocità media in tale intervallo.

Legenda: { o = numero segmenti trasmessi
{ O = CW

▓▓▓ = SSTHRESH

——— = RW



Annotazioni:

- Per t = 12 mi accorgo che mi è arrivato un ACK. SSTHRESH viene quindi abbassato a: $MAX(2, \frac{FLIGHTSIZE}{2}) = MAX(2,6) = 6$
- Per t = 15 ricevo ACK(4) e transito da SLOW-START a CONGESTION-AVOIDANCE. Quindi ACK vengono trattati diversamente.

CW = 4

SSTHRESH = 6 e ricevo ACK(4)

ACK #1 : SSTHRESH → CW = 5

ACK #2 : SSTHRESH → CW = 6

ACK #3 } CONGESTION-AVOIDANCE

ACK #4 }

(9@)

- Per $t = 29$ ms $\Rightarrow$ CWR $\cong$ 14.9
- Per $t = 31$ ms il TCP non trasmette in quanto il TCP trasmittente ha già riempito al massimo la CWR o la RWR.

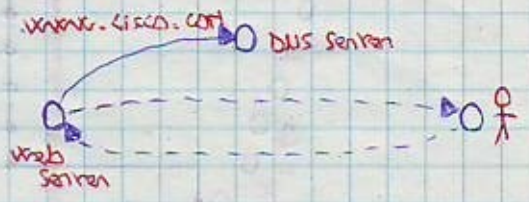
Si noti che il timeout raddoppia dopo la ritrasmissione dello stesso segmento. La velocità media è data da:

$$V_{MEDIA} = \frac{\left(\sum_{i=0}^{59} \text{numero segmenti trasmessi } (t = i \text{ ms}) \right) \cdot 1000 \text{ byte per ogni segmento} \cdot 8 \text{ bit per ogni byte}}{0{,}05 \text{ secondi}} \rightarrow (\text{bit/s})$$

È interessante notare che abbiamo ottenuto una capacità finita non tanto grande. Analizziamo ora i DNS. DNS = Domain Name System ed è un sistema per la gestione dei nomi in Internet. Consideriamo:

Senza DNS dovremmo specificare esplicitamente l'indirizzo IP del server.

Quindi l'idea è di fornire dei nomi simbolici alle macchine e ai servizi. I server DNS quindi offrono delle risoluzioni. Per esempio:



www.cisco.com è un nome simbolico che mi identifica un dominio. www è una label che identifica un servizio. Una risoluzione classica in DNS torna indietro un indirizzo IP.

Il DNS è un servizio che può girare sia su TCP che su UDP. Su Internet esiste un serio problema legato ai nomi. Infatti le macchine e i servizi nel mondo della rete sono tanti e quindi ci sono tanti nomi. Se questi nomi li organizziamo in uno spazio sfruttando la ricerca diventa onerosa. Quindi si stabilisce una gerarchia dei nomi tramite opportune segmentazioni dello spazio dei nomi stessi. Si ricordi che i nomi vengono codificati nella notazione dotted cioè nel tipo:

LABEL . LABEL . LABEL ... Per esempio: www . ELET . POLITI . IT

macchine e servizi del politecnico di Milano.