Potenziamento strutturale
Avviso D.D.274
del 15/02/2013

Ministero dell'Istruzione dell'Università e della Ricerca

PON Ricerca e Competitività 2007-2013 — Piano di Azione Coesione

Governo Italiano - Presidenza del Consiglio dei Ministri
Ministro per la Coesione Territoriale

Ettore Tamiro (GARR)

Performance tuning

Cosa vedremo

- Elementi di funzionamento dei protocolli di trasporto
- La cassetta degli attrezzi: di quali strumenti disponiamo per isolare e riparare un guasto?

TCP

- Elementi di funzionamento dei protocolli di trasporto

Cosa vedremo del TCP

- TCP: Transmission Control Protocol
- Modello di funzionamento
- Threeway handshake
- Multiplexing
- Buffer
- Finestra di trasmissione
- Network congestion
- Finestra di congestione

Cos'e' il TCP

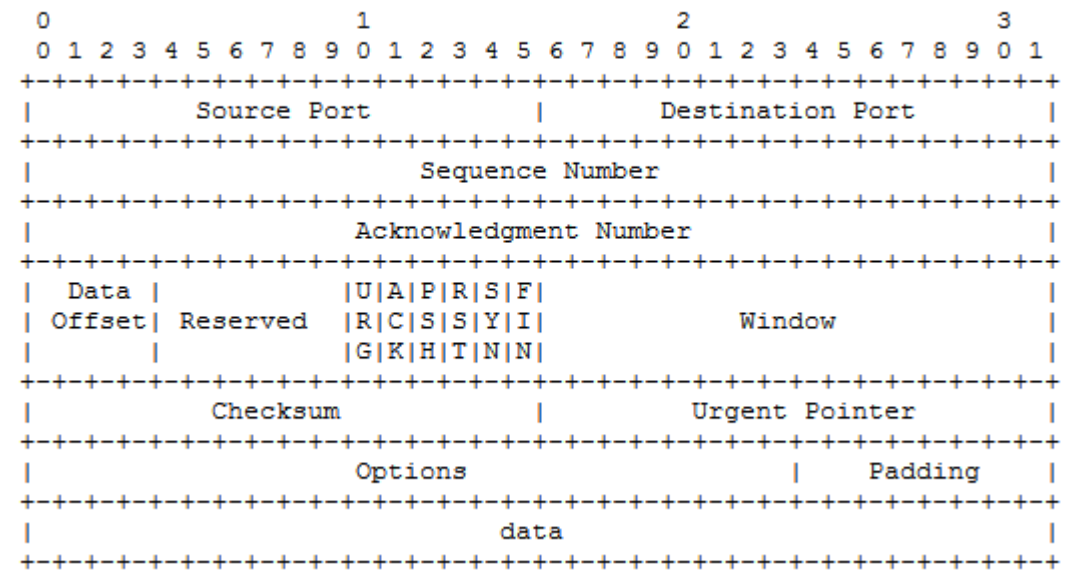
- TCP: Transmission Control Protocol
- Definito nella RFC 793
  - https://tools.ietf.org/rfc/rfc793.txt
- Offre un livello di trasporto affidabile tra due host: «client» e «server»
- E' orientato alla connessione
- Implementa meccanismi di controllo di flusso

TCP: modello di funzionamento

- Due host comunicano attraverso lo scambio di strutture dati chiamate «segmenti»
- I segmenti sono composti da
  - un header con informazioni di segnalazione
  - i dati da trasmettere: payload

TCP: header

TCP Header Format



Iniziamo a studiare l'Header TCP

- Ogni segmento e' identificato da due coppie di numeri
  - «Sequence number», «Acknowledgment number»
  - «Source port», «Destination Port»

Connessione affidabile ed ordinata

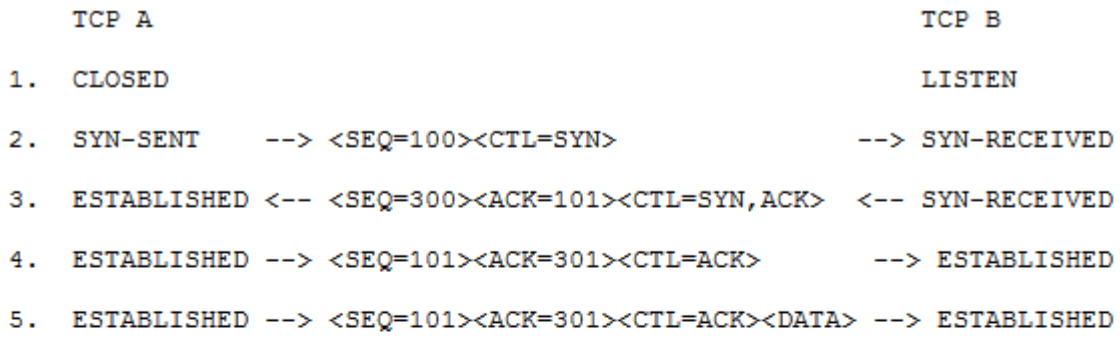
- Il «sequence number» comunica l'informazione su dove posizionare il «payload», permettendo di realizzare un flusso ordinato di dati.
- L' «Acknowledgment number» indica il prossimo byte che ci si aspetta di ricevere e, implicitamente, conferma tutti i precedenti. Se per un segmento inviato non si riceve il relativo «ACK», il segmento viene ritrasmesso garantendo l'affidabilita' della connessione

Che differenza c'e' tra gli host?

- Gli host hanno ruoli diversi
- Il «server» resta sempre in ascolto per l'arrivo di nuove connessioni dai client
- Il «client» inizia una connessione verso il server
- Per iniziare la connessione, il client esegue la <<threeway handshake»

TCP: iniziamo la connessione

- Threeway handshake



```
   TCP A                                                TCP B

1. CLOSED                                               LISTEN

2. SYN-SENT    --> <SEQ=100><CTL=SYN>               --> SYN-RECEIVED

3. ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED

4. ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>       --> ESTABLISHED

5. ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> --> ESTABLISHED
```

Basic 3-Way Handshake for Connection Synchronization

Figure 7.

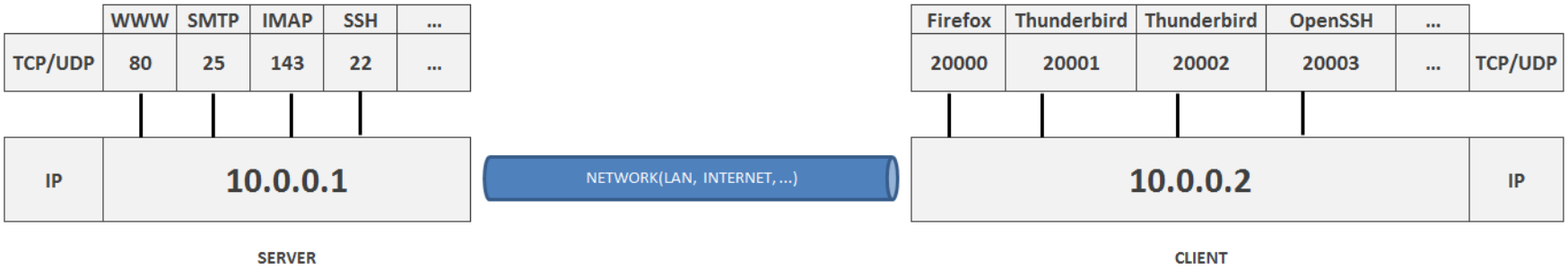
In line 2 of figure 7, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!).

TCP: multiplazione (1/2)

- «Source Port» e «Destination Port» permettono di implementare il meccanismo di multiplazione: condividere lo stesso indirizzo IP tra piu' processi
- Per analogia con la posta ordinaria, possiamo pensare all'indirizzo IP come la «via» ed al «Port Number» come il «civico»

TCP: multiplazione (2/2)



TCP: BUFFER (1/3)

- BUFFER: e' un'area di memoria, di dimensione finita, in cui mantenere temporaneamente i dati
- Il mittente mantiene i dati che non sono ancora stati confermati dal ricevente con un «ACK», in modo da poterli ritrasmettere se venissero persi
- Il ricevente mantiene i dati in arrivo dalla rete, aspettando che vengano prelevati dalle applicazioni a cui sono destinati
- Nel parallelo con la posta ordinaria, i buffer possono essere pensati come caselle di posta

TCP: BUFFER (2/3)

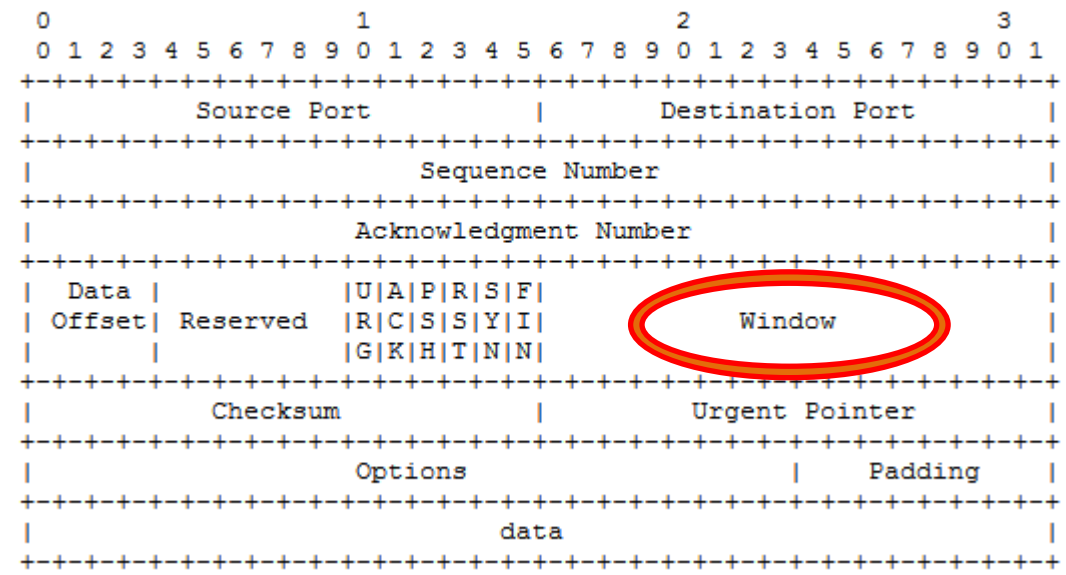
- I «buffer» sono usati per regolare diversi aspetti del comportamento delle comunicazioni in rete
- Ad esempio, nel caso di un protocollo di trasporto:
  - per trasferire file di grandi dimensioni si vorrebbero avere buffer molto capienti, il mittente potrebbe cosi' avere sempre spazio per nuovi segmenti di dati da inviare; il ricevente potrebbe accettare tutti i dati in arrivo dalla rete inviando solo pochi segmenti di «ACK»
  - per applicazioni interattive come la chat, si potrebbero usare buffer piccoli, ottimizzando la gestione della memoria

TCP: BUFFER (3/3)

- Nel caso reale, il dimensionamento dei buffer e' un aspetto molto importante del «tuning» dei protocolli di trasporto
- Il valore di questo parametro si riflette sul comportamento di varie funzioni che ne fanno uso. In particolare:
  - il buffer del mittente per il «congestion control»
  - Il buffer del ricevente per il «flow control»

TCP: window (1/5)

TCP Header Format



TCP: window (2/5)

- La «finestra» rappresenta il numero massimo di byte che il ricevente e' disposto ad accettare, ed e' condizionata dalla dimensione del buffer del ricevente; RFC 793:

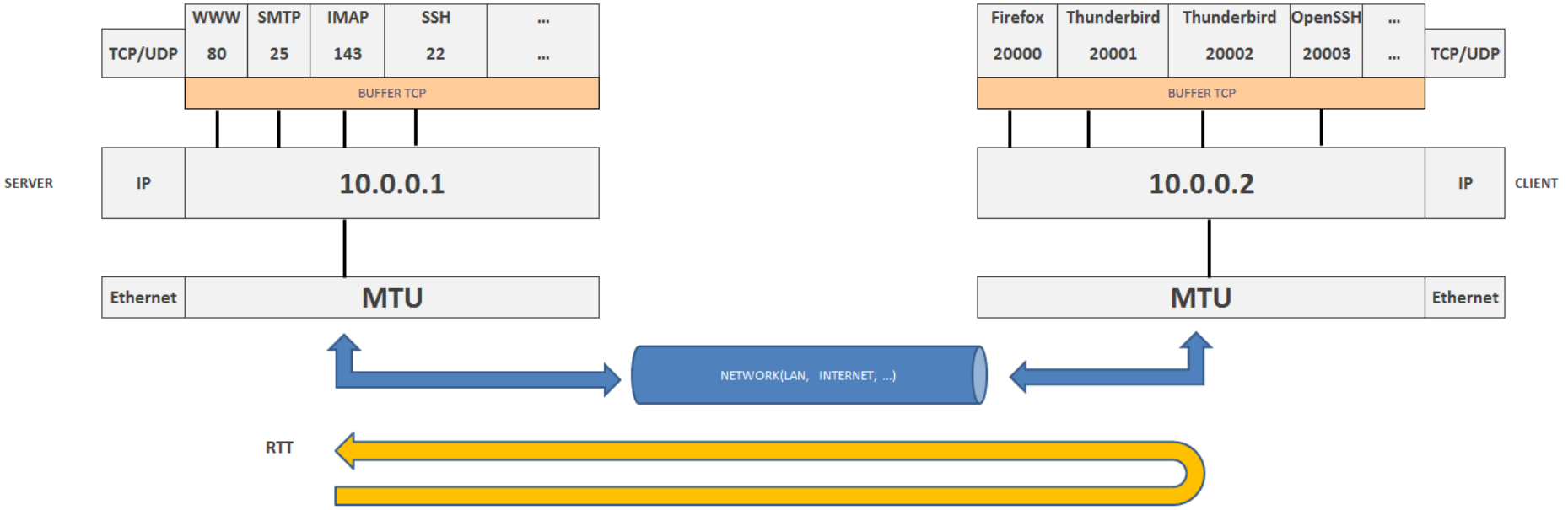
> Managing the Window
>
> The window sent in each segment indicates the range of sequence numbers the sender of the window (the data receiver) is currently prepared to accept. There is an assumption that this is related to the currently available data buffer space available for this connection.

- Una finestra puo' essere pari alle dimensioni di uno o piu' pacchetti grandi fino alla MTU
- MTU: quantita' massima di byte che possono essere contenuti in un pacchetto trasmesso da un'interfaccia. In genere coincide con il valore di MTU Ethernet: 1500 byte payload (piu' 26 byte di header)
- Il «round trip time», RTT, e' il tempo che un pacchetto impiega per eseguire un giro completo del percorso: andata e ritorno.

TCP: window (3/5)

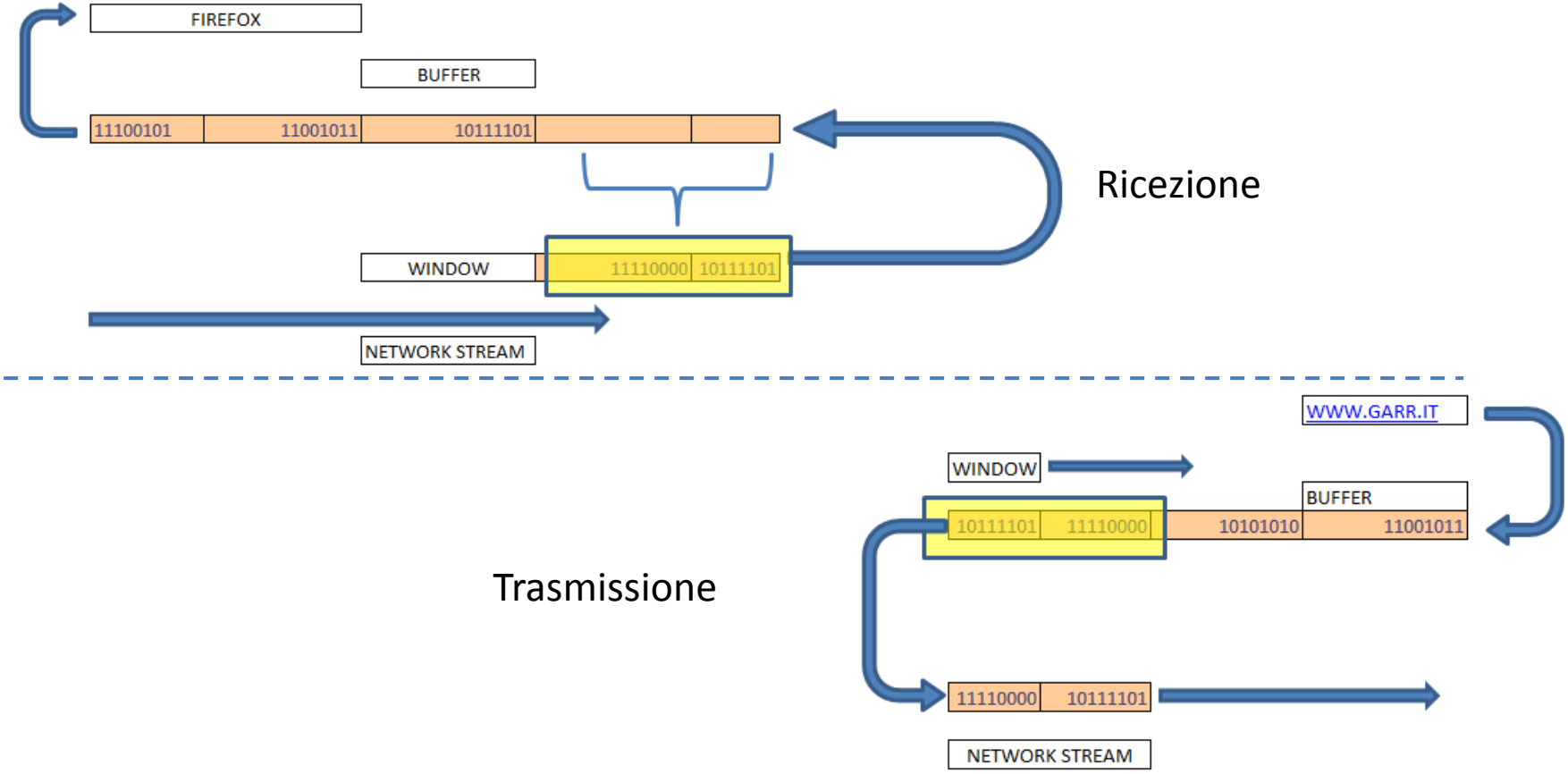
- La quantita' di dati trasmessi e non ancora confermati non puo' eccedere al dimensione della finestra
- Il mittente conserva in finestra i dati non confermati in modo da poterli ritrasmettere se persi
- Una volta confermato un segmento viene rimosso ed altri dati passano dal buffer alla finestra per essere trasmessi
- La dimensione massima della finestra e' sempre la minore tra la dimensione del buffer del mittente e la finestra annunciata ricevente

TCP: window (4/5)



TCP: window (5/5)

- L'implementazione di buffer e finestre e' peculiare di ogni sistema opeartivo e del suo stack TCP/IP, ma possiamo immaginarne un'astrazione...



TCP: cosa fa la window

- La finestra regola due funzioni diverse
  - Quella del ricevente agisce sul «flow control»: comunica al mittente di non inviare piu' dati di quelli che il mittente e' in grado di gestire
  - Quella del mittente regola il «congestion control»: evita che il collegamento venga riempito con piu' dati di quelli che e' capace di trasportare

TCP: quanta banda abbiamo?

- Il percorso tra il client ed il server puo' essere composto da molti collegamenti
- Ogni collegamento ha una propria capacita' di banda disponibile: questo valore varia nel tempo
- La capacita' di banda del collegamento tra il client ed il server e' pari alla banda massima disponibile sul piu' stretto collo di bottiglia presente nel percorso.

TCP: rapporto tra window e banda

- Avere un valore di finestra troppo grande non da' vantaggi perche' non si puo' superare la banda massima istantaneamente disponibile, e si sprecano risorse di memoria sull'host
- Una finestra troppo piccola limita la massima velocita' raggiungibile

TCP: il BDP

- Per calcolare la dimensione della finestra, tipicamente, si usa questa formula:

$$W = \text{Banda} \times \text{RTT}$$

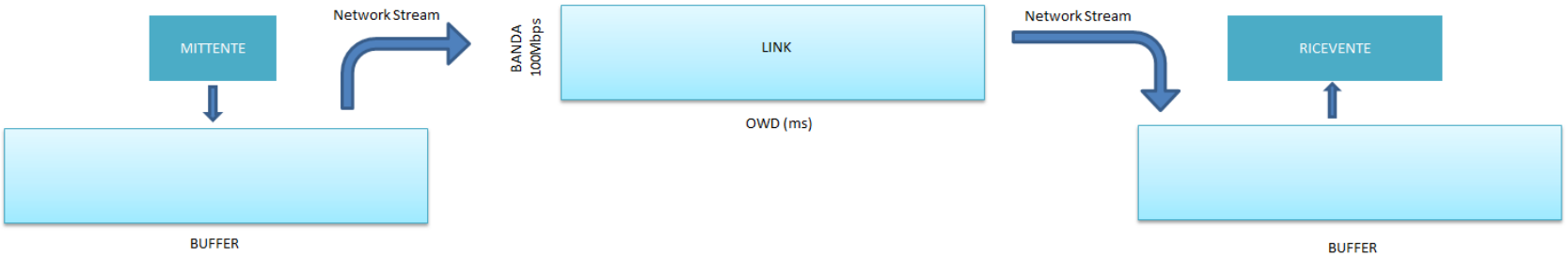
- E' il «Bandwidth delay product»:BDP. Rappresenta la capacita' del link tra il mittente ed il ricevente.

TCP: il BDP

- Proviamo a vedere come funziona il BDP in una condizione di banda piena
- Immaginiamo un link con 100Mbps tra due host

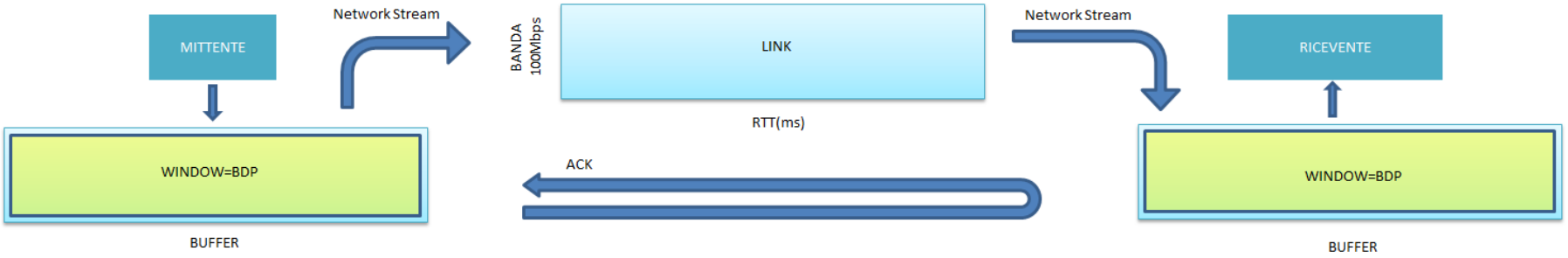
TCP: il BDP

- Questo link puo' essere immaginato come un tubo largo come la banda e lungo come il tempo impiegato per completare il percorso dal mittente al ricevente: one way delay (OWD)
- Il nostro compito e' dimensionare il buffer TCP in modo da gestire una window capace di contenere i dati in transito nel tubo e di creare una condizione ideale per cui ad ogni dato che preleva il ricevente ve ne sia un in arrivo dal mittente



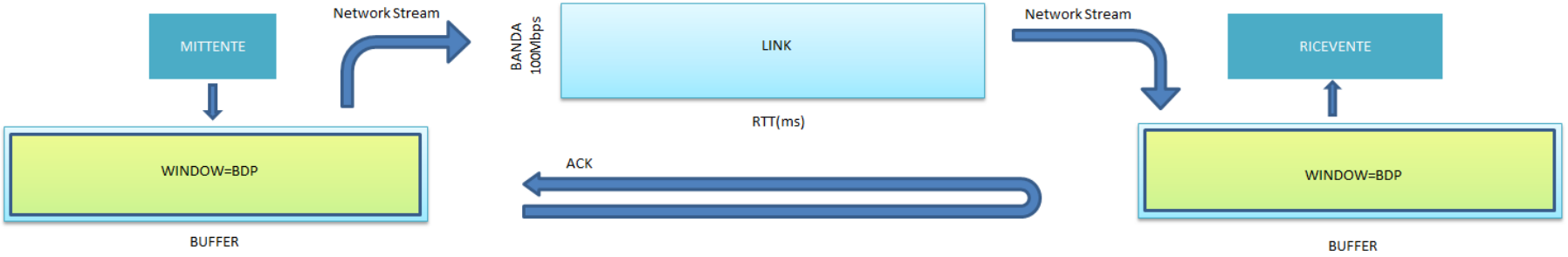
TCP: il BDP

- Non basta sapere la dimensione del tubo
- Per usarlo in maniera ottimale, il mittente deve sapere ogni quanto inviare un pacchetto
- La frequenza dei pacchetti in uscita dal tubo e' molto simile alla frequenza di arrivo degli ACK al mittente:
  - Infatti, per ogni pacchetto preso il ricevente invia indietro un ACK
  - La spaziatura tra gli ACK fornisce al mittente un «clock» affidabile per l'invio
- Useremo come «clock» l'RTT



TCP: il BDP

- A questo punto abbiamo le informazioni necessarie: conosciamo la dimensione del link ed il tempo impiegato ad attraversarlo
- Allochiamo un buffer capace di gestire una finestra pari al BDP
- Questa condizione e' solo ipotetica
- Il caso reale e' piu' complesso, ma la regola rimane

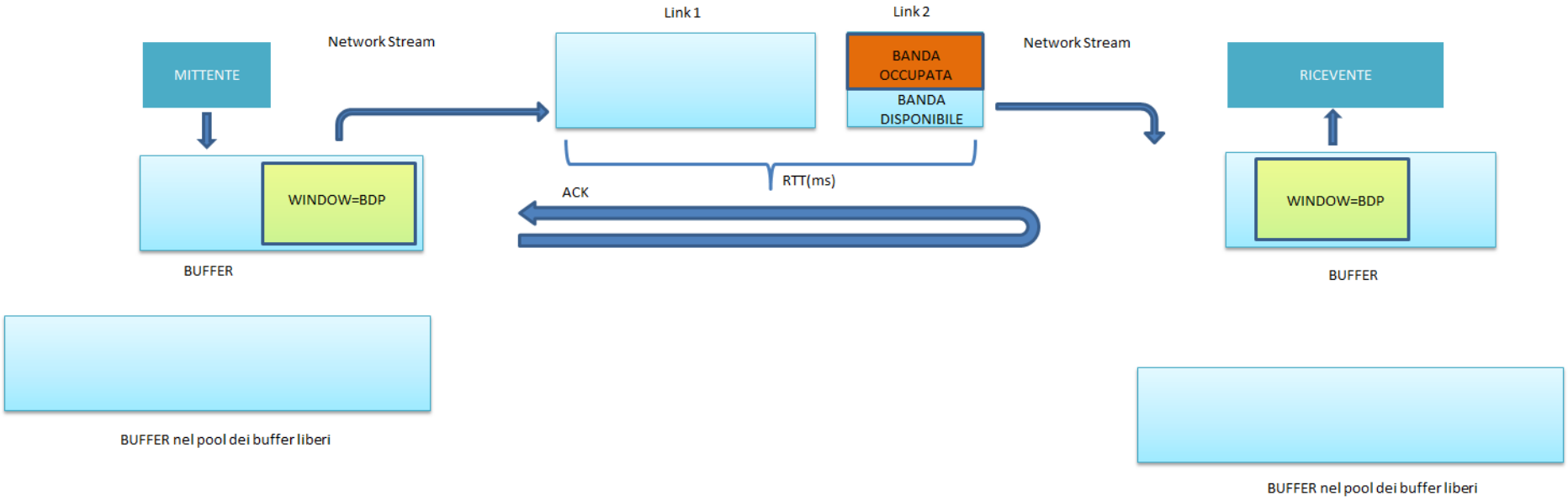


TCP: il BDP

- Il percorso tra due host puo' essere composto da molti link e possiamo incontrare colli di bottiglia
- Nell'esempio che segue, il link 1 ha 100Mbps di banda disponibile, ma il link 2 ha solo 40Mbps
- Entrambi gli host dispongono di interfacce 100Mbps, quindi vorrebbero poter gestire questa capacita' di banda

TCP: il BDP

- Questo percorso ha un collo di bottiglia quindi la sua banda e' 40Mbps
- A parita' di RTT, il BDP a 100Mbps e' maggiore che a 40Mbps
- Possiamo pensare che il TCP mantenga un pool di buffer di dimensione variabile e che li allochi secondo le necessita'
- Come vederemo piu' avanti, il TCP mantiene anche la «congestion window»



TCP: alcuni esempi di BDP

- Il primo «ACK» arrivera' nel tempo di 1 RTT quindi per mantenere il link sempre impegnato la finestra deve essere >= BDP

Mbps	bit	RTT (ms)	WINDOW (MB)
10	10000000	10	0.013
100	100000000	10	0.125
1000	1000000000	10	1.250

- Le attuali implementazioni di TCP possono dimensionare dinamicamente i buffer e possiamo configurare questo comportamento

TCP: come fare il tuning di BDP?

- Potremmo impostare la dimensione minima a 4096 byte (che generalmente corrisponde ad una pagina di memoria)
- La dimensione media al BDP di una destinazione vicina, la dimensione massima al BDP di una desitnazione remota.

TCP: opzione window scale

- Il campo «window» e' di 16 bit, quindi puo' arrivare al massimo a 65535 byte
- Per aumentare questa dimensione e' stata introdotta l'opzione «window scale»
- Si tratta di un fattore moltiplicativo della «window» espresso come potenza di 2
- Sfrutta il campo «options» dell'header TCP
- RFC 1323:
  - https://www.ietf.org/rfc/rfc1323.txt

TCP: ci serve un BDP pieno?

- Sebbene il BDP sia il valore ideale per dimensionare i buffer, non avremo sempre una sola connessione che usa tutta la banda
- Spesso vogliamo che piu' connessioni concorrenti usino una porzione di banda.
- Potremmo impostare il BDP ad ½ o ¼ del valore previsto

TCP: metodo per calcolare BDP

- Per trovare un valore di RTT da usare nel calcolo della «window» possiamo usare il comando «ping» verso alcune destinazioni note

TCP: il BDP di un posto vicino

- Per il limite inferiore facciamo ping verso www.google.it, che e' una destinazione direttamente connessa alla rete GARR

```
PING www.google.it (173.194.113.56) 56(84) bytes of data.
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=1 ttl=55 time=21.4 ms
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=2 ttl=55 time=21.3 ms
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=3 ttl=55 time=21.3 ms
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=4 ttl=55 time=21.3 ms
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=5 ttl=55 time=21.3 ms
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=6 ttl=55 time=21.3 ms
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=7 ttl=55 time=21.3 ms
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=8 ttl=55 time=21.3 ms
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=9 ttl=55 time=21.2 ms
64 bytes from fra02s20-in-f24.1e100.net (173.194.113.56): icmp_req=10 ttl=55 time=21.0 ms
^C
--- www.google.it ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9008ms
rtt min/avg/max/mdev = 21.044/21.328/21.419/0.189 ms
```

Mbps	bit	RTT (ms)	WINDOW (MB)
10	10000000	21	0.026
100	100000000	21	0.263
1000	1000000000	21	2.625

TCP: il BDP di un posto lontano

- Per il limite superiore www.internet2.edu, che si trova oltre oceano ed e' raggiungibile da GARR attraverso la rete Europea dell'Educazione e della Ricerca: GEANT.

```
PING webprod2.internet2.edu (207.75.164.248) 56(84) bytes of data.
64 bytes from internet2.edu (207.75.164.248): icmp_req=1 ttl=52 time=156 ms
64 bytes from internet2.edu (207.75.164.248): icmp_req=2 ttl=52 time=156 ms
64 bytes from internet2.edu (207.75.164.248): icmp_req=3 ttl=52 time=156 ms
64 bytes from internet2.edu (207.75.164.248): icmp_req=4 ttl=52 time=156 ms
64 bytes from internet2.edu (207.75.164.248): icmp_req=5 ttl=52 time=156 ms
64 bytes from internet2.edu (207.75.164.248): icmp_req=6 ttl=52 time=156 ms
64 bytes from internet2.edu (207.75.164.248): icmp_req=7 ttl=52 time=156 ms
64 bytes from internet2.edu (207.75.164.248): icmp_req=8 ttl=52 time=156 ms
64 bytes from internet2.edu (207.75.164.248): icmp_req=9 ttl=52 time=156 ms
64 bytes from internet2.edu (207.75.164.248): icmp_req=10 ttl=52 time=156 ms
^C
--- webprod2.internet2.edu ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8999ms
rtt min/avg/max/mdev = 156.092/156.297/156.828/0.402 ms
```

Mbps	bit	RTT (ms)	WINDOW (MB)
10	10000000	156	0.195
100	100000000	156	1.950
1000	1000000000	156	19.500

TCP: stimiamo il BDP

- Considerando finestre di ¼ del BDP e che visto che RTT non e' fisso arrotondiamo, potremmo usare

Mbps	bit	RTT (ms)	WINDOW (MB)
10	10000000	21	0.026
100	100000000	21	0.263
1000	1000000000	21	2.625

Mbps	bit	RTT (ms)	WINDOW (MB)
10	10000000	156	0.195
100	100000000	156	1.950
1000	1000000000	156	19.500

	MIN (BYTE)	MED (BYTE)	MAX (BYTE)
1Gbps	4096	768000	5120000

TCP: tuning dei S.O.

- Il tuning dei buffer nei vari sistemi operativi puo' essere regolato cosi:
- Windows Vista/7/8 solo il buffer ricevente
- Windows server mittente e ricevente
- Linux 2.4 solo mittente
- Da Linux 2.6 mittente e ricevente
- Da Mac OSX 10.5 mittente e ricevente

TCP: tuning di Linux

- Con Linux possiamo regolare i buffer con
  - Buffer ricevente
    - echo “<min> <default> <max>” > /proc/sys/net/ipv4/tcp_rmem
  - Buffer mittente
    - echo “<min> <default> <max>” > /proc/sys/net/ipv4/tcp_wmem
  - Window Scaling
    - echo 1 > /proc/sys/net/ipv4/tcp_window_scaling
  - Le modifiche posso essere rese permanenti in
    - /etc/sysctl.conf

TCP: tuning di Windows

- Con MS-Windows, l'autotuning usa buffer fino a 16MB
- Puo' essere configurato dalla «net-shell»
  - netsh int tcp set global autotuninglevel=<opzione>
- Le opzioni disponibili sono:
  - disabled: RWIN di 64KB
  - highlyrestricted
  - restricted
  - normal: default
  - Experimental
- A questo link troverete la loro documentazione
  - http://www.speedguide.net/articles/windows-7-vista-2008-tweaks-2574

TCP: tuning di OSX

- Per OSX editare /etc/sysctl.conf
- Da OSX 10.9
  - net.inet.tcp.win_scale_factor=8
    net.inet.tcp.autorcvbufmax: 16777216
    net.inet.tcp.autosndbufmax: 16777216
- Prima di 10.9
  - net.inet.tcp.win_scale_factor=8
    kern.ipc.maxsockbuf=16777216
- A questo link trovate la documentazione per OSX
  - https://rolande.wordpress.com/2010/12/30/performance-tuning-the-network-stack-on-mac-osx-10-6/

Gestione della congestione

TCP: gestione della congestione

- Il TCP dispone di procedure per gestire la congestione
- L'impiego di banda viene automaticamente regolato in base alla qualita' del collegamento

TCP: slow start

- La soluzione proposta da Van Jacobson, nel 1988, prende il nome di «slow start»
- RFC 5681
  - https://tools.ietf.org/html/rfc5681
- Il documento di Van Jacobson si trova qui:
  - http://faculty.cord.edu/zhang/cs345/assignments/researchPapers/congavoid.pdf
- Le variabili usate sono:
  - Congestion window, «cwnd»
  - Slow start threshold, «ssthresh»

TCP: componenti dello slow start

- Questa tecnica e' la combinazione di 7 algoritmi
    1) round-trip-time variance estimation
    2) exponential retransmit timer backoff
    3) slow-start
    4) more aggressive receiver ack policy
    5) dynamic window sizing on congestion
    6) Karn's clamped retransmit backoff
    7) fast retransmit

TCP: algoritmi trattati nel doc

- Il documento di V. Jacobson descrive gli algoritmi da 1 a 5 e rimanda a documenti appositi per gli algoritm 6 e 7
- Sebbene vengano usati molteplici algoritmi, i cui fondamenti tecnici non sono banali, l'implementazione dello slow start e' particolarmente compatta

TCP: elementi dello slow start

Proviamo ad approfondire alcuni elementi dello «slow start»

TCP: flusso conservativo

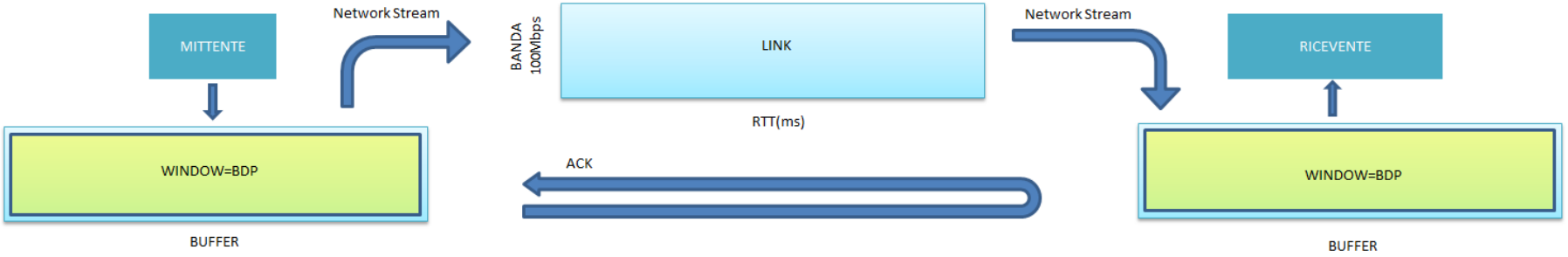
- Uno flusso TCP deve obbedire al principio di «conservazione dei pacchetti»
- La congestione e' causata dagli HOP che violano questo principio
- Un flusso che rispetti questo principio e' detto «conservativo»:
  - un pacchetto nuovo non viene aggiunto al flusso se prima non e' stato prelevato un pacchetto vecchio

TCP: cerchiamo un clock

- Abbiamo bisogno di un «clock» che ci dica quando «impegnare la linea»
- Abbiamo gia' visto un clock di questo tipo

TCP: ACK CLOCK (1/2)

- Abbiamo detto che gli «ACK» informano il mittente che un pacchetto e' stato prelevato dal flusso
- Abbiamo chiamato «RTT» il tempo tra la trasmissione del paccheto e l'arrivo dell' «ACK»



TCP: ACK CLOCK (2/2)

- P_b : spaziatura tra pacchetti dovuta al collo di bottiglia
- P_r : spaziatura tra pacchetti al ricevente
- A_r : spaziatura tra «ACK» inviati dal ricevente
- A_b : spaziatura tra «ACK» nel collo di bottiglia
- A_s : spaziatura tra «ACK» al mittente

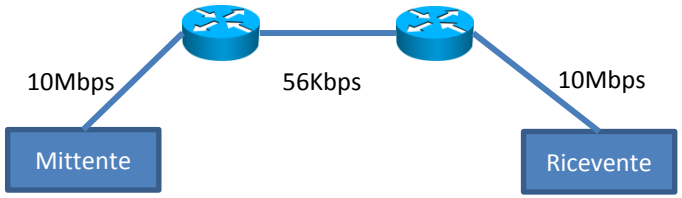
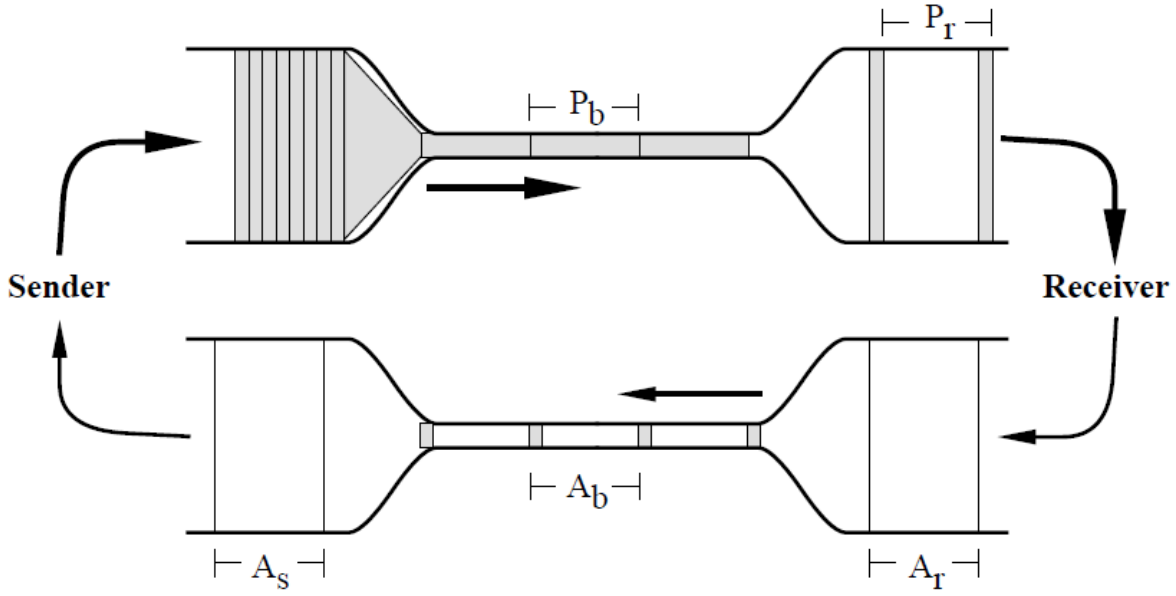


Figure 1: Window Flow Control 'Self-clocking'



TCP: quanto trasmettere?

- Il clock ci dice «quando» impegnare la linea...
- ...dobbiamo stabilire «quanto» impegnarla

TCP: la congestion window (1/4)

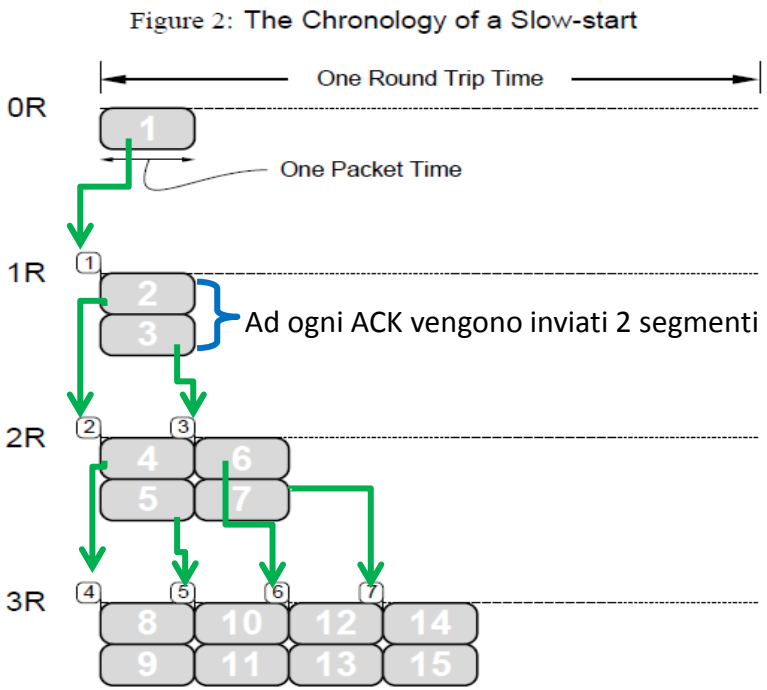
- Il TCP mantiene la soglia di apertura della finestra nella variabile «cwnd»
- La «cwnd» e' la massima quantita' di byte che la rete e' capace di trasportare in un dato momento

TCP: la congestion window (2/4)

- IL TCP del mittente manterra' la propria finestra di trasmissione sempre al valore piu' basso tra quella annunciata dal mittente e la «cwnd»
- Ad inizio connessione, dopo aver eseguito la «threeway handshake», viene avviato l'algoritmo di «slow start»

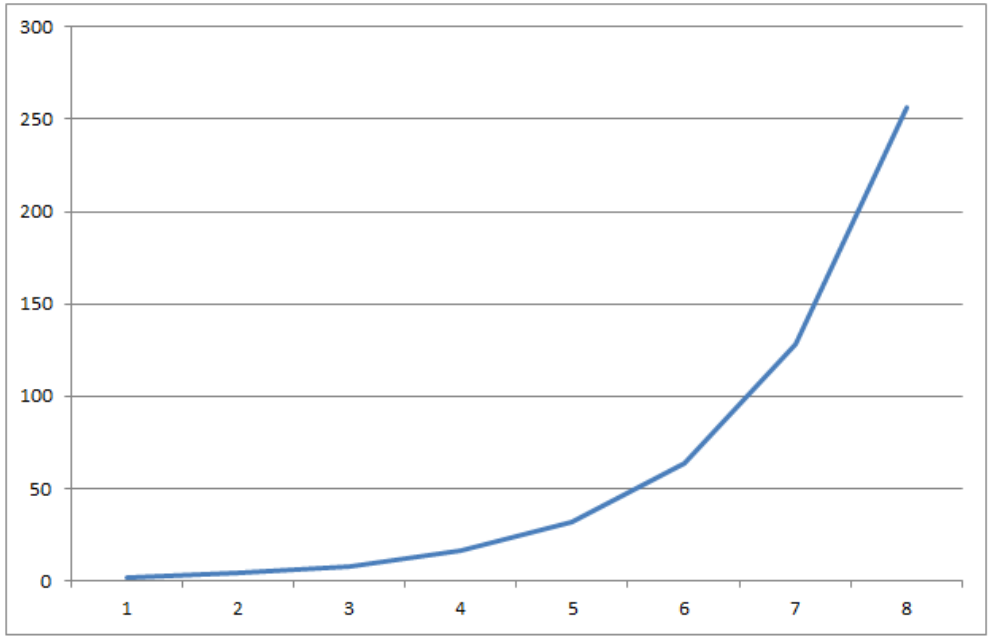
TCP: la congestion window (3/4)

- La «cwnd» e' posta inizialemte ad 1 segmento, trascorso un RTT diventa 2, poi 4, etc...
- Per ogni «ACK» ricevuto
  - cwnd=cwnd+1



TCP: la congestion window (4/4)

- Si tratta di crescita esponenziale.
- Il tempo di «apertura della finestra» e'
  - $R * \log_2(W)$



TCP: la congestion avoidance (1/2)

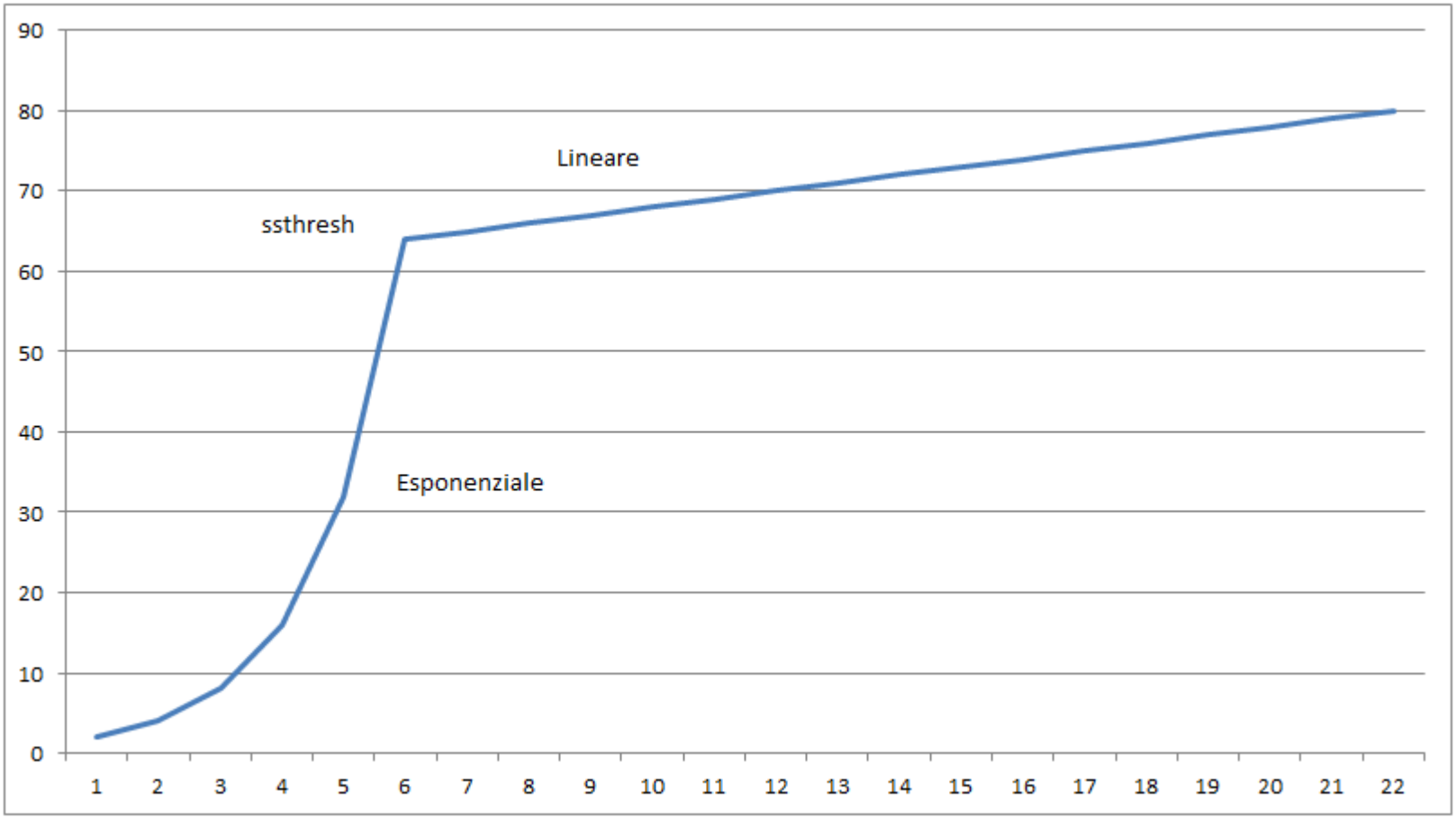
- Usare solo crescita esponenziale non esclude che si verifichi congestione
- «ssthresh» rappresenta la soglia dell'ultima congestione rilevata e segna il passaggio tra la fase di «slow start» e quella di «congestion avoidance»
- Si tratta di procedimenti diversi ma usati congiuntamente

TCP: la congestion avoidance (2/2)

- Quando TCP rileva congestione:
  - ssthresh=cwnd/2
  - cwnd=1 segmento
  - Se (cwnd<ssthresh) crescita esponenziale
    - Cwnd=cwnd+1
  - Altrimenti la crescita diventa lineare
    - cwnd=cwnd+(1/cwnd)

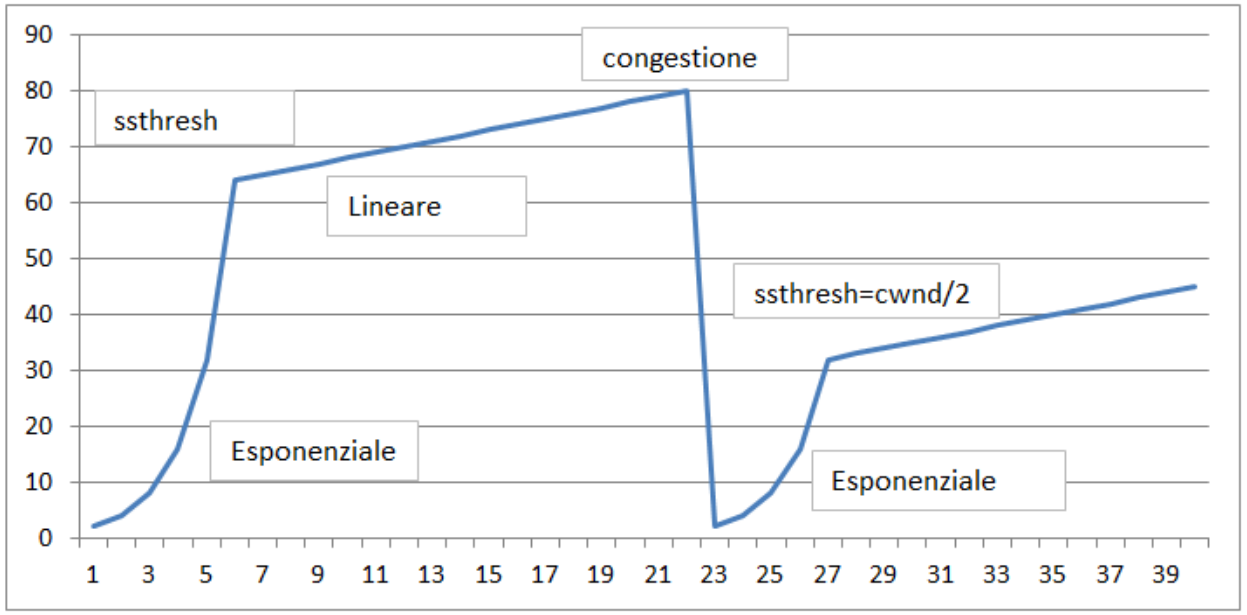
```
if (cwnd < ssthresh)
        /* if we're still doing slow-start
         * open window exponentially */
        cwnd += 1;
else
        /* otherwise do Congestion
         * Avoidance increment-by-1 */
        cwnd += 1/cwnd;
```

TCP: fase esponenziale e lineare



TCP: TAHOE

- Questa implementazione prende il nome di «TCP TAHOE», dal nome della realease di BSD UNIX nella quale fu introdotta
- Questo e' il comportamento di una sequenza «slow start» e «congestion avoidance»



TCP: alcuni dettagli (1/10)

Uno sguardo all'esperimento di V. Jacobson

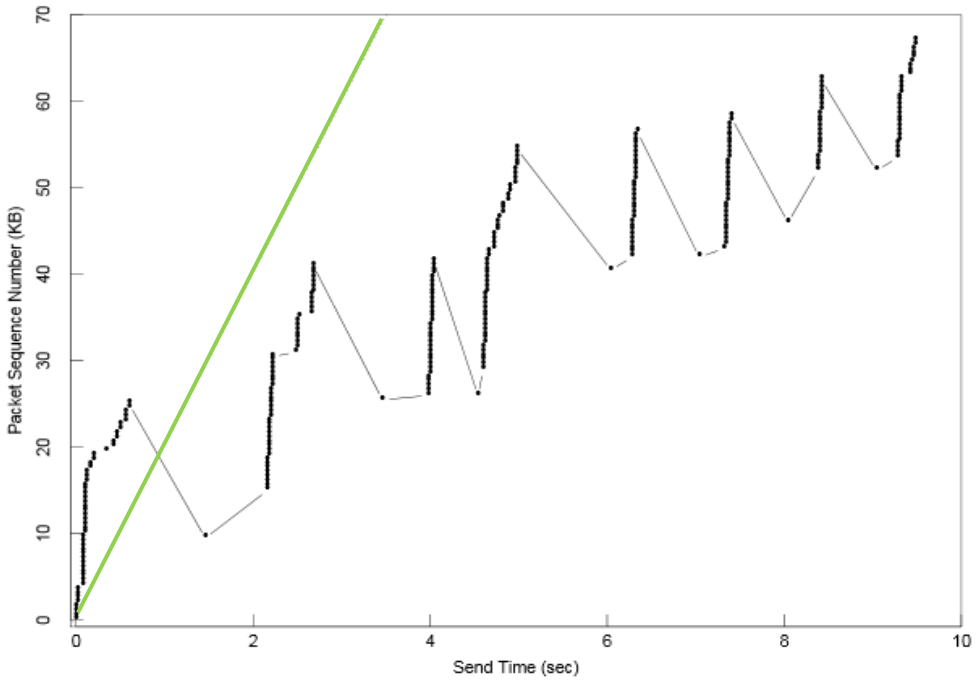
TCP: alcuni dettagli (2/10)

- Nel 1986 si verificarono frequenti e diffusi fenomeni di congestione della rete
- Tutti gli utenti di 4.3BSD segnalarono un evidente degrado delle performance di rete
- Ad esempio il collegamento tra gli edifici del «Lawrence Berkeley Laboratory» e «UC Berkeley», collegati con un link a microonde e distanti 365 metri, passo' da 32Kbps a 40bps

TCP: alcuni dettagli (3/10)

- In questo grafico pacchetti/tempo, vediamo che l'impiego di banda del TCP era molto aggressivo e portava a frequenti ritrasmissioni
- La retta verde rappresenta il comportamento ideale di una connessione e la sua pendenza la banda disponibile

Figure 3: Startup behavior of TCP without Slow-start



TCP: alcuni dettagli (4/10)

- V. Jacobson ed altri, convinti che le cause fossero in un malfunzionamento del software, iniziarono a raccogliere le segnalazioni
- Implementarono una collezione di strumenti software come debugger a livello kernel, traceroute, etc, capaci di fornire reportistica dettagliata e li distribuirono agli utenti di 4.3BSD
- Analizzarono il comportamento del TCP in relazione agli eventi

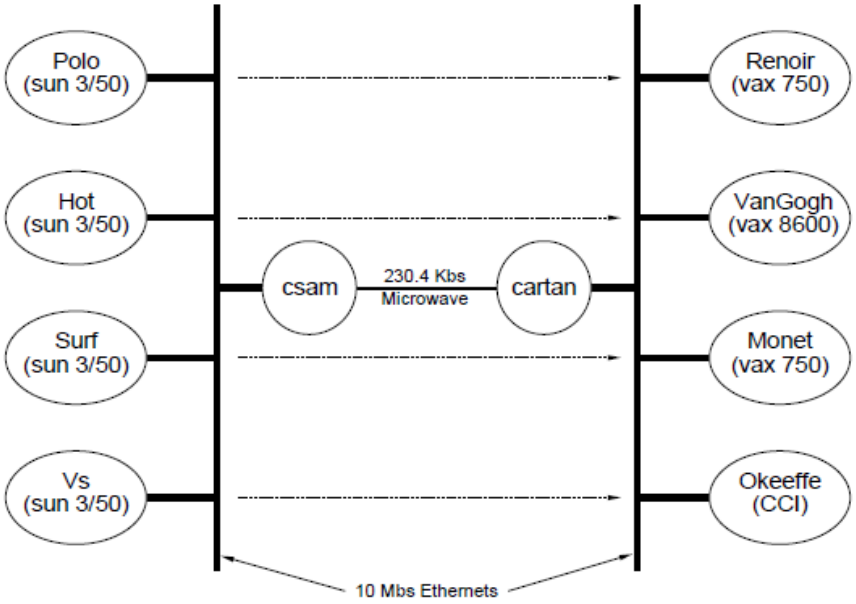
TCP: alcuni dettagli (5/10)

- Elaborarono una soluzione, appunto lo «slow start» e la «congestion avoidance», e ne dimostrarono l'efficacia
- La soluzione fu distribuita agli utenti ed ancora oggi e' alla base di tutte le tecniche di impiego efficiente della banda disponibile

TCP: alcuni dettagli (6/10)

- L'ambiente di test era composto da due reti a 10Mbps connesse attraverso un link a microonde da 230.4Kbps

Figure 7: Multiple conversation test setup

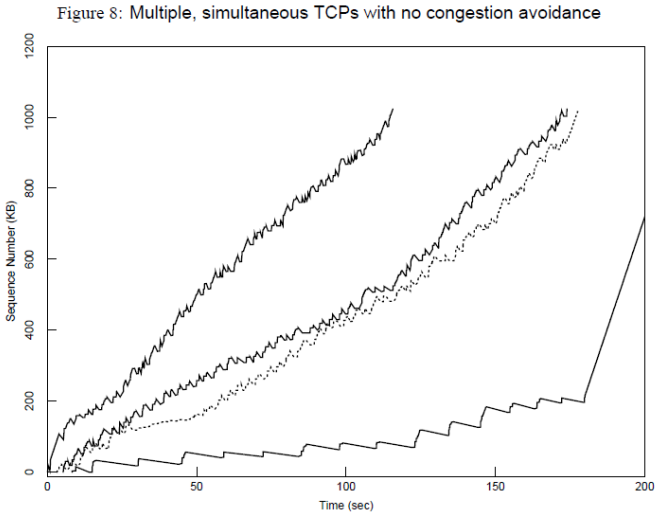


TCP: alcuni dettagli (7/10)

- Il link a microonde poteva gestire una coda di 50 pacchetti di 512Byte, quindi 25KB
- Su questa rete vennero eseguiti 4 trasferimenti di file da 1MB spaziati di 3 secondi
- I file venivano trasferiti come 2048 pacchetti di 512Byte
- Ad ogni connessione sarebbe spettato ¼ dei 25KB, cioe' circa 6KB
- Ogni connessione aveva una finestra di 16KB cioe' 32 pacchetti di 512Byte

TCP: alcuni dettagli (8/10)

- Due connessioni erano suffcienti a superare la capacita' della coda del link a microonde
- Quattro connessioni creavano un overflow di 78 pacchetti: circa il 160% di capacita' della coda
- Senza «congestion avoidance», su 11.000 pacchetti 4.000 furno ritrasmissioni: circa il 160%



TCP: alcuni dettagli (9/10)

- Con «congestion avoidance» su 8.281 pacchetti 89 furno ritrasmissioni: 1%

Figure 8: Multiple, simultaneous TCPs with no congestion avoidance

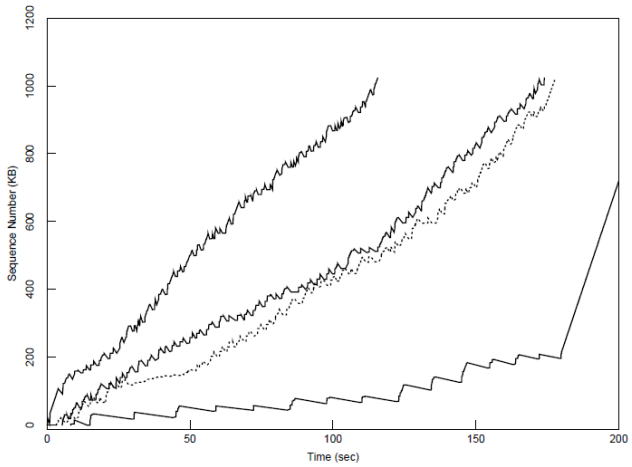
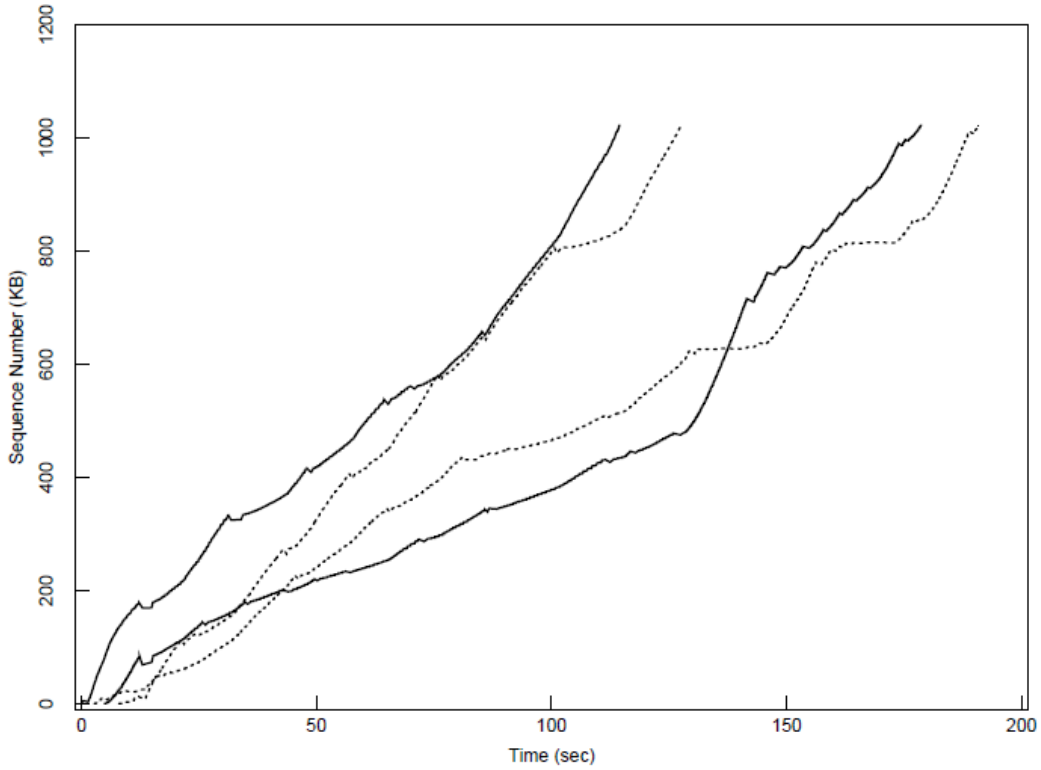


Figure 9: Multiple, simultaneous TCPs with congestion avoidance



TCP: alcuni dettagli (10/10)

- Impiego di banda con e senza «slow start»

Figure 3: Startup behavior of TCP without Slow-start

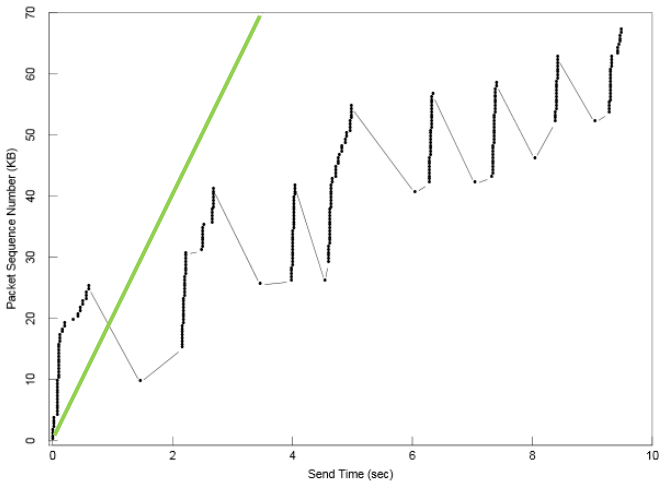
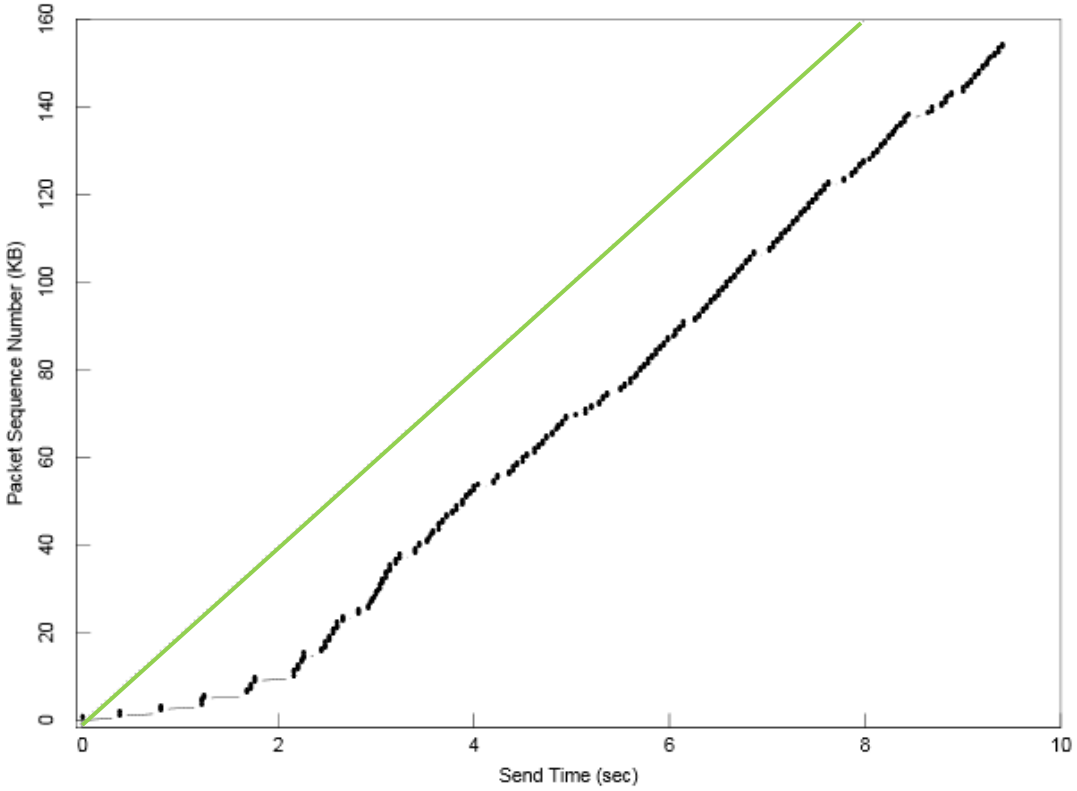


Figure 4: Startup behavior of TCP with Slow-start



TCP: tecnica di troubleshooting

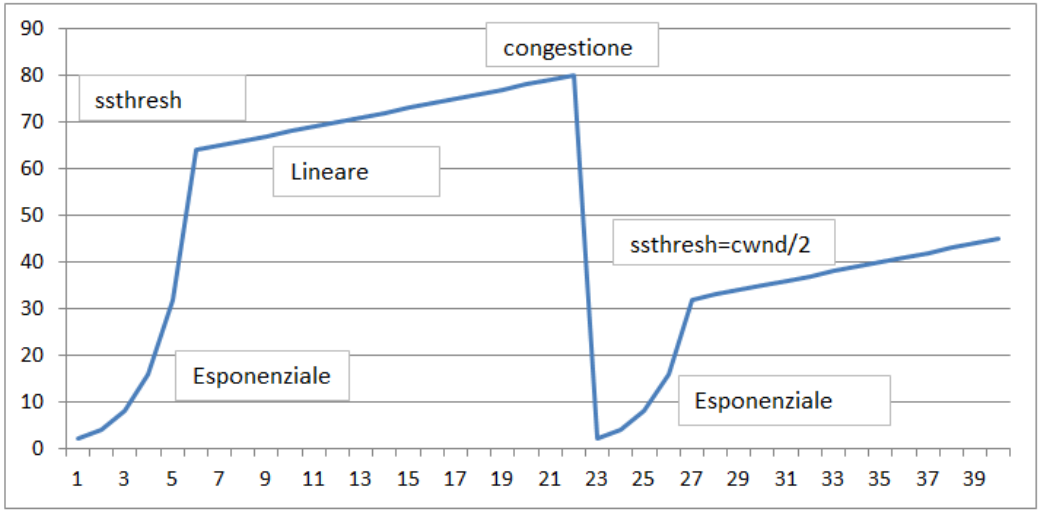
- La procedura usata per investigare i problemi di congestione nel 1986 e' ancora valida ed e' in uso a tutt'oggi:
  - Produrre uno schema dettagliato della rete
  - Documentarne le caratteristiche
  - Segmentare ed introdurre sonde
  - Produrre reportistica dettagliata
  - Intervenire sui parametri di configurazione

TCP: slow start va sempre bene?

- Per connessioni con cwnd piccole questo comportamento va bene
- Per cwnd grandi e' lento

TCP: come si comporta TAHOE?

- In caso di congestione forte si deve eseguire slow start, tuttavia:
  - TAHOE ha una decrescita forte e un recupero lento
  - Non distingue trai tipi di conestione



TCP: quante congestioni esistono?

- Ci sono almeno due tipi di congestione
  - Forte congestione: scadenza del timer di ritrasmissione
  - Lieve congestione: arrivo di 3 «ACK» duplicati. I segmenti nuovi sono giunti al ricevente che li conferma inviando un «sequence number» piu' basso e pari al primo byte perso

TCP: quando usare slow start

- Se la congestione e' forte, si esegue slow start
  - si registra la «ssthresh» e si resetta la «cwnd»
  - Viene eseguito lo «slow start» fino a cwnd<ssthresh, poi la crescita diventa lineare di cwnd=cwnd+(1/cwnd)

TCP: quando possiamo usare altro

- Per congestione leggera si puo' fare di meglio che TAHOE
- Ci sono implementazioni di TCP che riconoscono i vari tipi di congestione
- E per congestione leggera eseguono gli algoritmi di «fast retransmit» e «fast recovery»

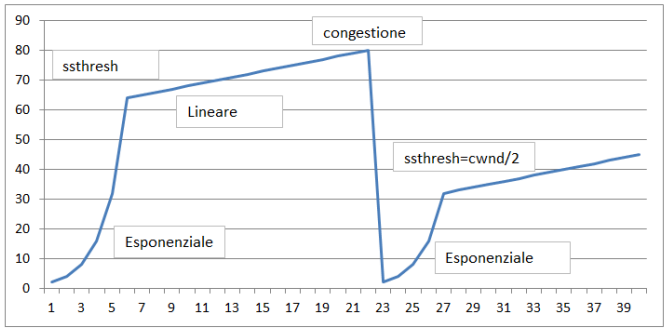
TCP: algoritmo RENO (1/3)

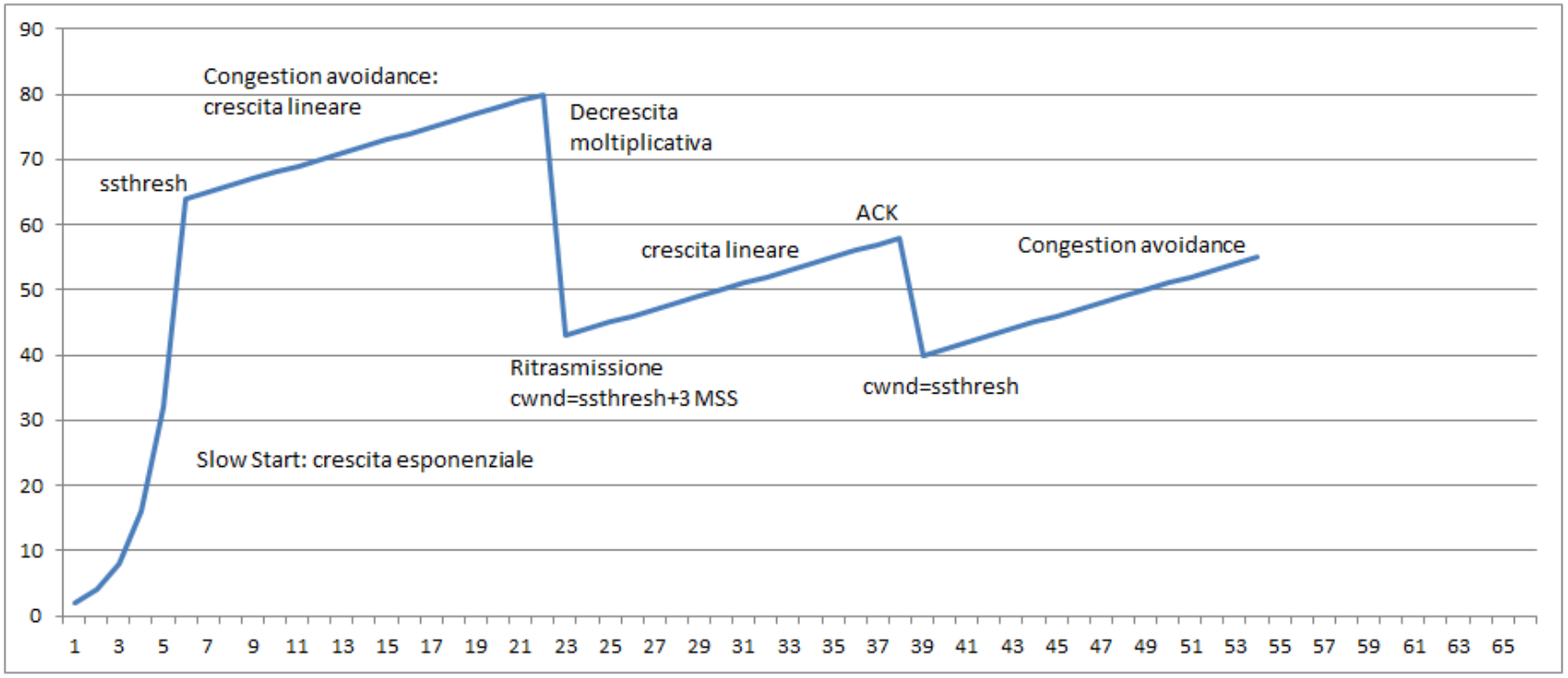
- TCP RENO e’ una implementazione di TCP che
  - Per congestione forte esegue TAHOE
  - Per congestione leggera esegue algoritmi «fast»

TCP: algoritmo RENO (2/3)

- Appena rileva 3 ACK duplicati, RENO ritrasmette il segmento richiesto senza aspettare che scada il suo timer
- Imposta:
  - ssthresh=cwnd/2
  - cwnd=ssthresh+3 MSS
- Avvia la crescita lineare: cwnd=cwnd + 1 MSS
- Quando riceve «ACK» per il segmento perso pone
  - cwnd=ssthresh
- Avvia «congestion avoidance»

TCP: algoritmo RENO (3/3)





TCP: c'e' altro che possiamo fare?

- Collegamenti con molta banda e RTT lunghi vengono detti LFN: «Long Fat Networks»
- Su questi collegamenti, anche usando le tecniche descritte, il tempo necessario per riempire la banda a causa di perdite sarebbe grande: il BDP di un collegamento 10Gb e'
  - 10ms=12.5MB
  - 100ms=125MB

TCP: oltre TAHOE e RENO… (1/3)

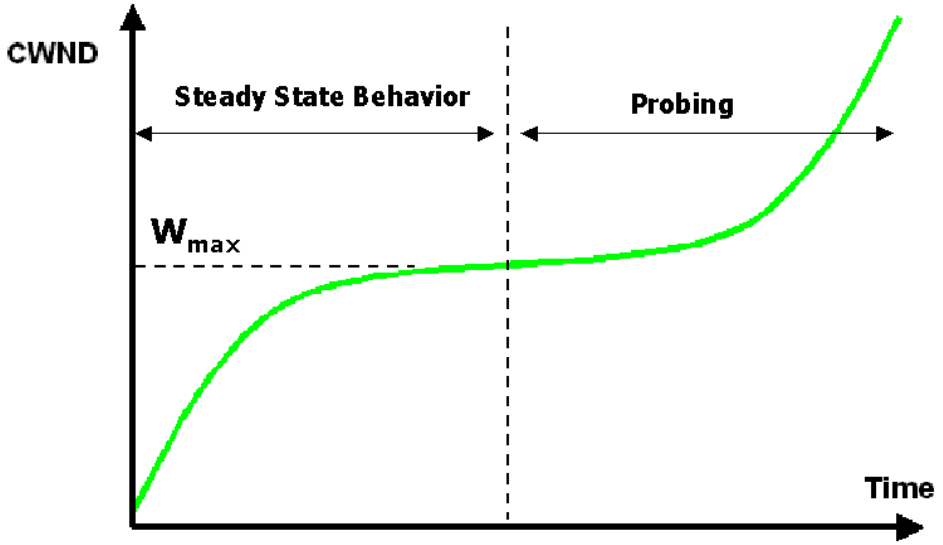
- I sistemi operativi permettono di cambiare a runtime l'implementazione delle procedure di «fast retransmit» e «fast recovery» ed piu' in generale del «congestion control»
- Esistono diversi algoritmi che possono produrre considerevoli variazioni alle curve che abbiamo visto

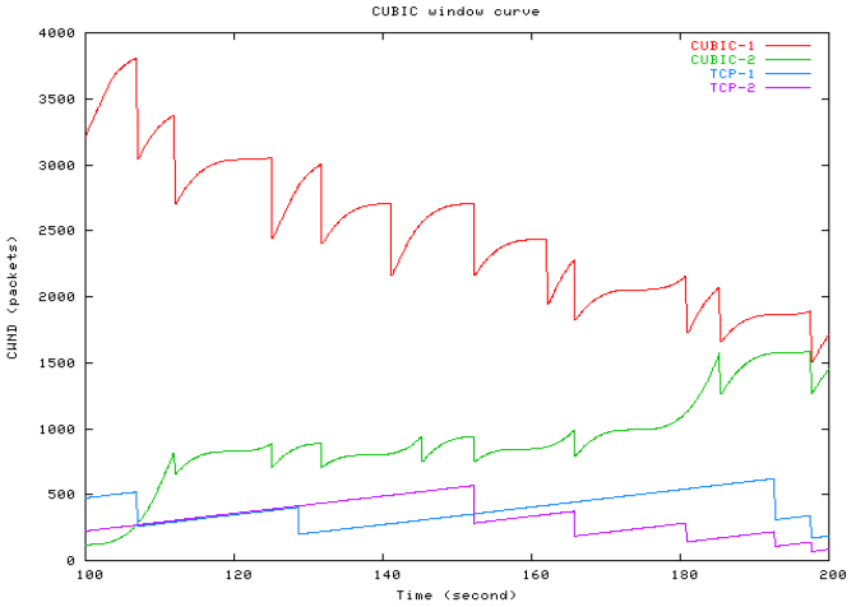
TCP: oltre TAHOE e RENO… (2/3)

- Si tratta di algoritmi derivati da RENO
- Usano opzioni TCP come «selective acks»
  - RFC 2018
    - https://tools.ietf.org/html/rfc2018
- L'algoritmo «CUBIC», ad esempio, non si basa sugli ACK
  - http://research.csc.ncsu.edu/netsrv/?q=content/bic-and-cubic
  - http://www4.ncsu.edu/~rhee/export/bitcp/cubic-paper.pdf

TCP: oltre TAHOE e RENO… (3/3)

- CUBIC prova a far crescere rapidamente la «cwnd» ma, in prossimita' dell'ultima congestione rilevata, diminuisce il ritmo di incremento: in questa fase la curva e' concava
- Se il precedente punto di congestione viene superato con successo,la curva torna a crescere rapidamente diventando convessa





TCP: alcuni algoritmi

- Alcuni degli algoritmi di congestion control presenti in Linux oltre RENO
  - Cubic (default di Linux dal kernel 2.6.19)
  - Scalable
  - vegas
  - Illinois
  - Westwood
  - …

TCP: configurare Linux

- Con Linux
  - Lista dei moduli caricati
    - sysctl net.ipv4.tcp_available_congestion_control
  - Per caricare un modulo
    - modprobe tcp_algoritmo
      - Esempio: modprobe tcp_scalable
  - Per attivare il modulo
    - sysctl -w net.ipv4.tcp_congestion_control=algoritmo
      - Esempio: sysctl -w net.ipv4.tcp_congestion_control=scalable

La cassetta degli attrezzi

- La cassetta degli attrezzi

Alcuni strumenti

- Cosa deve esserci nella cassetta degli attrezzi:
  - Ping
  - Traceroute
  - Iperf
  - NDT
  - TCPDUMP
  - TCPtrace
  - Wireshark
  - Looking Glass

PING (1/2)

- Ping
- Si basa sullo scambio di messaggi ICMP:
  - «echo request»
  - «echo reply»
- Ci dice se un host e' raggiungibile ed in che RTT
- Perdita di pacchetti
- Filtri di banda

```
PING www.garr.it (193.206.158.2) 56(84) bytes of data.
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=1 ttl=64 time=0.606 ms
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=2 ttl=64 time=0.261 ms
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=3 ttl=64 time=0.254 ms
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=4 ttl=64 time=0.268 ms
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=5 ttl=64 time=0.234 ms
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=6 ttl=64 time=0.224 ms
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=7 ttl=64 time=0.236 ms
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=8 ttl=64 time=0.228 ms
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=9 ttl=64 time=0.260 ms
64 bytes from lx1.dir.garr.it (193.206.158.2): icmp_req=10 ttl=64 time=0.257 ms
^C
--- www.garr.it ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9010ms
rtt min/avg/max/mdev = 0.224/0.282/0.606/0.110 ms
```

PING (2/2)

- Non tutti gli host rispondono al «ping» e spesso i firewall lo filtrano.
- Un server che e' raggiungibile al «www» potrebbe, ad esempio, non esserlo al «ping»

Traceroute (1/4)

- Il «traceroute» mostra il percorso dall'host che lo esegue fino alla destinazione
- Funziona con ICMP, UDP e TCP
- Il percorso inverso potrebbe essere differente e non lo si puo' vedere

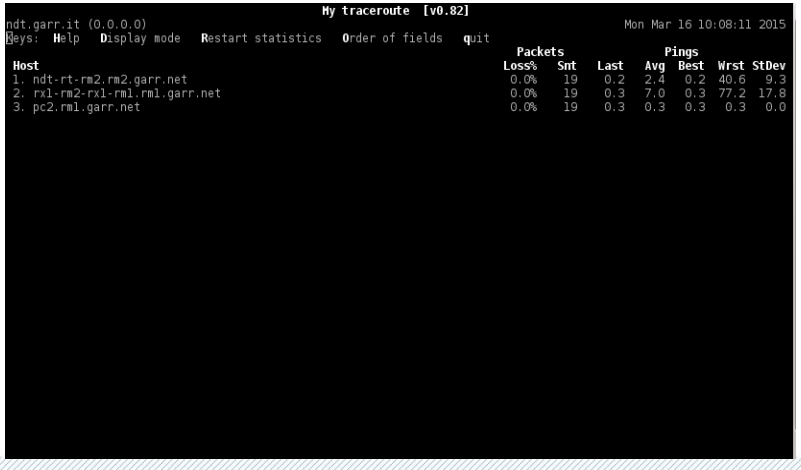
Traceroute (2/4)

- Funziona mandando pacchetti verso la destinazione impostando il TTL inizialmente ad 1 ed incrementandolo di 1 ad ogni iterazione
- Ogni host intermedio vedra' arrivare un pacchetto con TTL=1 e lo scartera' mandando indietro un messaggio di errore. L'host tracciato rispondera' correttamente.

```
traceroute to www.google.it (173.194.113.79), 30 hops max, 60 byte packets
 1  ru-dirgarrlan-ru-dir.dir.garr.it (193.206.158.249)  0.640 ms  0.644 ms  0.680 ms
 2  ru-dir-l1-rx2-rm2.rm2.garr.net (193.206.138.209)  3.520 ms  3.524 ms  3.567 ms
 3  rx2-rm2-rx2-mi2.mi2.garr.net (90.147.80.66)  15.143 ms  13.299 ms  15.135 ms
 4  rx2-mi2-r-mi2.mi2.garr.net (90.147.80.77)  12.296 ms  12.302 ms  12.292 ms
 5  r-mi2-google.mi2.garr.net (193.206.129.130)  12.443 ms  12.445 ms  12.465 ms
 6  216.239.47.128 (216.239.47.128)  12.411 ms 209.85.249.54 (209.85.249.54)  12.214 ms  26.439 ms
 7  72.14.232.78 (72.14.232.78)  21.850 ms 72.14.232.76 (72.14.232.76)  22.042 ms 72.14.232.78 (72.14.232.78)  22.038 ms
 8  72.14.234.232 (72.14.234.232)  22.501 ms 209.85.251.179 (209.85.251.179)  21.463 ms 209.85.251.249 (209.85.251.249)  22.109 ms
 9  209.85.242.209 (209.85.242.209)  23.261 ms  22.552 ms  22.915 ms
10  fra02s21-in-f15.1e100.net (173.194.113.79)  21.781 ms  21.414 ms  21.002 ms
```

Traceroute (3/4)

- Usando i messaggi di errore possiamo ottenere delle informazioni utili
- Percorso
- Tempi hop-to-hop
- Routing loop: coppia di router che si «rimbalzano» il pacchetto senza farlo arrivare a destinazione
- Blackhole: router oltre i quali l'instradamento si perde
- MTR e' una comoda implementazione di traceroute



Traceroute (4/4)

- Anche traceroute puo' essere filtrato nella sua versione ICMP, possiamo provare con TCP o UDP
- Viene comunque fermato dai firewall NAT
- Non rileva switch ed apparati di livello 2
- Per avere un output leggibile e' utile, ma non indispensabile, che sia stata prodotta la zona DNS inversa e che i nomi siano esplicativi

IPERF

- IPERF
- Si tratta di uno strumento «client/server» per la misurazione di banda tra due host
- Misura la banda sia TCP che UDP
- Permette di impostare la dimesione della finestra TCP

IPERF: cosa fa

- Esegue trasferimenti di dati prodotti in memoria, quindi non risente del tempo di accesso al disco
- Puo' usare sessioni multiple, quindi trasferimenti simultanei, o una sola sessione
- Ci dice, in base alle condizioni di rete ed alla gestione dei buffer sull'host, qual e' la massima banda TCP

IPERF: come andrebbe usato

- IPERF e' lo strumento usato piu' di frequente per la gestione dei problemi di perfomance
- Per avere una visione realistica del funzionamento di un link, va' usato in assenza di altro traffico
- Bisogna eseguire le prove sempre nei due versi:
  - A turno ogni host fara' il server (ricevente) e poi il client (mittente)

IPERF: UDP

- Quando usato in modalita’ UDP, IPERF prova costantemente a riempire il link fino alla banda indicata sulla riga di comando
- La riuscita di questo test ci dira’ la banda presente ed il packet loss

IPERF: esempio

- IPERF
- Un client ed un server in esecuzione
- Entrambi chiedono update ogni 5s (-i 5)
- Il client esegue un test di 20s (–t 20)
- Il sever alloca una window da 9.5MB ed il client da 684KB
- Si riesce a riempire il link da 1Gbps

```
noc@ndt:~$ iperf -s -i 5
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 9.54 MByte (default)
------------------------------------------------------------
[  4] local 193.206.131.178 port 5001 connected with 193.204.221.72 port 50046
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 5.0 sec   538 MBytes   902 Mbits/sec
[  4]  5.0-10.0 sec   564 MBytes   946 Mbits/sec
[  4] 10.0-15.0 sec   551 MBytes   924 Mbits/sec
[  4] 15.0-20.0 sec   564 MBytes   946 Mbits/sec
[  4]  0.0-20.0 sec  2.17 GBytes   930 Mbits/sec
```

```
noc@pc2:~$ iperf -c ndt.garr.it -t 20 -i 5
------------------------------------------------------------
Client connecting to ndt.garr.it, TCP port 5001
TCP window size:  684 KByte (default)
------------------------------------------------------------
[  3] local 193.204.221.72 port 50046 connected with 193.206.131.178 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec   538 MBytes   903 Mbits/sec
[  3]  5.0-10.0 sec   564 MBytes   946 Mbits/sec
[  3] 10.0-15.0 sec   551 MBytes   924 Mbits/sec
[  3] 15.0-20.0 sec   564 MBytes   947 Mbits/sec
[  3]  0.0-20.0 sec  2.17 GBytes   930 Mbits/sec
noc@pc2:~$
```

IPERF: esempio

- Il test deve essere eseguito nel verso opposto ed avere risultati simmetrici

```
noc@ndt:~$ iperf -c ndt3.garr.it -t 20 -i 5
------------------------------------------------------------
Client connecting to ndt3.garr.it, TCP port 5001
TCP window size: 9.54 MByte (default)
------------------------------------------------------------
[  3] local 193.206.131.178 port 30024 connected with 193.204.221.72 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec   573 MBytes   962 Mbits/sec
[  3]  5.0-10.0 sec   563 MBytes   945 Mbits/sec
[  3] 10.0-15.0 sec   564 MBytes   946 Mbits/sec
[  3] 15.0-20.0 sec   566 MBytes   950 Mbits/sec
[  3]  0.0-20.0 sec  2.21 GBytes   949 Mbits/sec
noc@ndt:~$
```

```
noc@pc2:~$ iperf -s -i 5
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size:  684 KByte (default)
------------------------------------------------------------
[  4] local 193.204.221.72 port 5001 connected with 193.206.131.178 port 30024
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 5.0 sec   564 MBytes   946 Mbits/sec
[  4]  5.0-10.0 sec   564 MBytes   947 Mbits/sec
[  4] 10.0-15.0 sec   564 MBytes   947 Mbits/sec
[  4] 15.0-20.0 sec   564 MBytes   947 Mbits/sec
[  4]  0.0-20.1 sec  2.21 GBytes   947 Mbits/sec
```

IPERF: cambiamo la window

- Proviamo a vedere cosa succede se cambiamo la window
- Il client chiede 10KB (-w 10K)
- La banda che il test riesce a riempire e' la meta': 545Mbps
- I buffer del mittente sono troppo piccoli

```
noc@ndt:~$ iperf -s -i 5
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 9.54 MByte (default)
------------------------------------------------------------
[  4] local 193.206.131.178 port 5001 connected with 193.204.221.72 port 50656
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 5.0 sec   324 MBytes   544 Mbits/sec
[  4]  5.0-10.0 sec   325 MBytes   545 Mbits/sec
[  4] 10.0-15.0 sec   324 MBytes   544 Mbits/sec
[  4] 15.0-20.0 sec   325 MBytes   545 Mbits/sec
[  4]  0.0-20.0 sec  1.27 GBytes   544 Mbits/sec
```

```
noc@pc2:~$ iperf -c ndt.garr.it -w 10K -t 20 -i 5
------------------------------------------------------------
Client connecting to ndt.garr.it, TCP port 5001
TCP window size: 20.0 KByte (WARNING: requested 10.0 KByte)
------------------------------------------------------------
[  3] local 193.204.221.72 port 50656 connected with 193.206.131.178 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec   324 MBytes   544 Mbits/sec
[  3]  5.0-10.0 sec   325 MBytes   545 Mbits/sec
[  3] 10.0-15.0 sec   324 MBytes   544 Mbits/sec
[  3] 15.0-20.0 sec   325 MBytes   545 Mbits/sec
[  3]  0.0-20.0 sec  1.27 GBytes   544 Mbits/sec
noc@pc2:~$
```

IPERF: regoliamo la window

- Proviamo a capire come regolare meglio il buffer
- Facciamo un ping tra i due host ed osserviamo l'RTT
- Calcoliamo il BDP per 1Gbps con l'RTT appena trovato
- E settiamo correttamente il buffer chiesto da IPERF

IPERF: proviamo a regolare il BDP

- Un ping rileva un RTT di 0.28ms ed il relativo BDP e’

Mbps	bit	RTT (ms)	WINDOW (MB)	WINDOW (KB)
1000	1000000000	0.28	0.035	35.00

IPERF: proviamo la nuova window

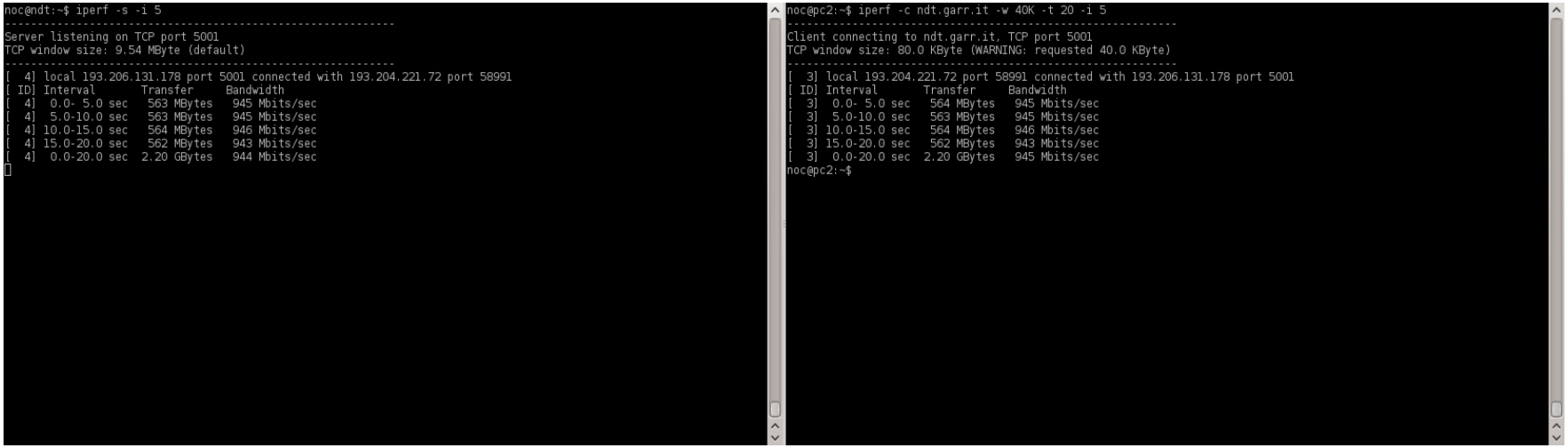
- Possiamo rifare la prova con un buffer da 35KB

```
noc@ndt:~$ iperf -s -i 5
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 9.54 MByte (default)
------------------------------------------------------------
[  4] local 193.206.131.178 port 5001 connected with 193.204.221.72 port 56837
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 5.0 sec   561 MBytes   941 Mbits/sec
[  4]  5.0-10.0 sec   561 MBytes   942 Mbits/sec
[  4] 10.0-15.0 sec   561 MBytes   941 Mbits/sec
[  4] 15.0-20.0 sec   563 MBytes   944 Mbits/sec
[  4]  0.0-20.0 sec  2.19 GBytes   941 Mbits/sec
```

```
PING ndt.garr.it (193.206.131.178) 56(84) bytes of data.
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=1 ttl=62 time=0.272 ms
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=2 ttl=62 time=0.262 ms
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=3 ttl=62 time=0.263 ms
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=4 ttl=62 time=0.260 ms
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=5 ttl=62 time=0.263 ms
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=6 ttl=62 time=0.279 ms
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=7 ttl=62 time=0.261 ms
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=8 ttl=62 time=0.263 ms
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=9 ttl=62 time=0.261 ms
64 bytes from ndt.garr.it (193.206.131.178): icmp_req=10 ttl=62 time=0.261 ms
^C
--- ndt.garr.it ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9011ms
rtt min/avg/max/mdev = 0.260/0.264/0.279/0.017 ms
noc@pc2:~$
noc@pc2:~$
noc@pc2:~$
noc@pc2:~$
noc@pc2:~$
noc@pc2:~$ iperf -c ndt.garr.it -w 35K -t 20 -i 5
------------------------------------------------------------
Client connecting to ndt.garr.it, TCP port 5001
TCP window size: 70.0 KByte (WARNING: requested 35.0 KByte)
------------------------------------------------------------
[  3] local 193.204.221.72 port 56837 connected with 193.206.131.178 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec   561 MBytes   941 Mbits/sec
[  3]  5.0-10.0 sec   561 MBytes   941 Mbits/sec
[  3] 10.0-15.0 sec   561 MBytes   941 Mbits/sec
[  3] 15.0-20.0 sec   563 MBytes   944 Mbits/sec
[  3]  0.0-20.0 sec  2.19 GBytes   942 Mbits/sec
noc@pc2:~$
```

IPERF: verifichiamo la window

- Abbiamo ancora una volta il nostro Gbps ed allocando una porzione di buffer molto minore di quella allocata automaticamente: 35KB contro 684KB
- Se ripetiamo il test con buffer 40KB la situazione non migliora molto: ovviamente bisogna tenere conto che l'RTT varia un po' nel tempo



IPERF: piu' sessioni simultanee

- Con IPERF possiamo anche eseguire piu' sessioni simultanee
- Facciamo un test con 2 sessioni (-P 2)
- Le sessioni condivideranno la banda ma la loro somma sara' compatibile con i risultati precedenti

```
noc@ndt:~$ iperf -s -i 5
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 9.54 MByte (default)
------------------------------------------------------------
[  4] local 193.206.131.178 port 5001 connected with 193.204.221.72 port 41375
[  5] local 193.206.131.178 port 5001 connected with 193.204.221.72 port 41376
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 5.0 sec   281 MBytes   471 Mbits/sec
[  5]  0.0- 5.0 sec   283 MBytes   475 Mbits/sec
[SUM]  0.0- 5.0 sec   564 MBytes   946 Mbits/sec
[  4]  5.0-10.0 sec   282 MBytes   473 Mbits/sec
[  5]  5.0-10.0 sec   282 MBytes   474 Mbits/sec
[SUM]  5.0-10.0 sec   564 MBytes   947 Mbits/sec
[  4] 10.0-15.0 sec   276 MBytes   463 Mbits/sec
[  5] 10.0-15.0 sec   288 MBytes   484 Mbits/sec
[SUM] 10.0-15.0 sec   564 MBytes   947 Mbits/sec
[  4] 15.0-20.0 sec   284 MBytes   476 Mbits/sec
[  5] 15.0-20.0 sec   280 MBytes   471 Mbits/sec
[SUM] 15.0-20.0 sec   564 MBytes   946 Mbits/sec
[  4]  0.0-20.0 sec  1.10 GBytes   471 Mbits/sec
[  5]  0.0-20.0 sec  1.11 GBytes   476 Mbits/sec
[SUM]  0.0-20.0 sec  2.20 GBytes   946 Mbits/sec
```

```
noc@pc2:~$ iperf -c ndt.garr.it -w 35K -t 20 -i 5 -P 2
------------------------------------------------------------
Client connecting to ndt.garr.it, TCP port 5001
TCP window size: 70.0 KByte (WARNING: requested 35.0 KByte)
------------------------------------------------------------
[  4] local 193.204.221.72 port 41376 connected with 193.206.131.178 port 5001
[  3] local 193.204.221.72 port 41375 connected with 193.206.131.178 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 5.0 sec   283 MBytes   475 Mbits/sec
[  3]  0.0- 5.0 sec   281 MBytes   471 Mbits/sec
[SUM]  0.0- 5.0 sec   564 MBytes   947 Mbits/sec
[  4]  5.0-10.0 sec   282 MBytes   474 Mbits/sec
[  3]  5.0-10.0 sec   282 MBytes   473 Mbits/sec
[SUM]  5.0-10.0 sec   564 MBytes   947 Mbits/sec
[  4] 10.0-15.0 sec   288 MBytes   484 Mbits/sec
[  3] 10.0-15.0 sec   276 MBytes   463 Mbits/sec
[SUM] 10.0-15.0 sec   564 MBytes   947 Mbits/sec
[  4] 15.0-20.0 sec   280 MBytes   470 Mbits/sec
[  4]  0.0-20.0 sec  1.11 GBytes   476 Mbits/sec
[  3] 15.0-20.0 sec   284 MBytes   476 Mbits/sec
[SUM] 15.0-20.0 sec   564 MBytes   946 Mbits/sec
[  3]  0.0-20.0 sec  1.10 GBytes   471 Mbits/sec
[SUM]  0.0-20.0 sec  2.20 GBytes   947 Mbits/sec
noc@pc2:~$
```

IPERF: UDP

- Possiamo eseguire un test UDP (-u)
- Dovremo indicare la banda desiderata: proviamo con 100Mbps (-b 100M)
- Sul server (ricevente)possiamo vedere le informazioni di jitter e packet loss

```
noc@ndt:~$ iperf -u -s -i 5
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  512 KByte (default)
------------------------------------------------------------
[  3] local 193.206.131.178 port 5001 connected with 193.204.221.72 port 45443
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0- 5.0 sec  59.9 MBytes   101 Mbits/sec   0.002 ms    0/42734 (0%)
[  3]  5.0-10.0 sec  59.9 MBytes   101 Mbits/sec   0.003 ms    0/42735 (0%)
[  3] 10.0-15.0 sec  59.9 MBytes   101 Mbits/sec   0.003 ms    0/42735 (0%)
[  3] 15.0-20.0 sec  59.9 MBytes   101 Mbits/sec   0.003 ms    0/42735 (0%)
[  3]  0.0-20.0 sec   240 MBytes   101 Mbits/sec   0.003 ms    0/170939 (0%)
[  3]  0.0-20.0 sec  1 datagrams received out-of-order
```

```
noc@pc2:~$ iperf -u -c ndt.garr.it -t 20 -i 5 -b 100M
------------------------------------------------------------
Client connecting to ndt.garr.it, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  684 KByte (default)
------------------------------------------------------------
[  3] local 193.204.221.72 port 45443 connected with 193.206.131.178 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec  59.9 MBytes   101 Mbits/sec
[  3]  5.0-10.0 sec  59.9 MBytes   101 Mbits/sec
[  3] 10.0-15.0 sec  59.9 MBytes   101 Mbits/sec
[  3] 15.0-20.0 sec  59.9 MBytes   101 Mbits/sec
[  3]  0.0-20.0 sec   240 MBytes   101 Mbits/sec
[  3] Sent 170940 datagrams
[  3] Server Report:
[  3]  0.0-20.0 sec   240 MBytes   101 Mbits/sec   0.003 ms    0/170939 (0%)
[  3]  0.0-20.0 sec  1 datagrams received out-of-order
noc@pc2:~$
```

NDT

- NDT fornira' informazioni circa:
- Banda disponibile tra client e server
- Configurazione dettagliata dello stack TCP/IP del client: le informazioni possono essere condivise con il NOC di GARR per valutare assieme eventuali misconfigurazione del client
- Problemi di negoziazione
- Presenza di firewall/NAT

NDT

- Alcuni problemi di NDT:
  - Java non e' sempre abilitato nel browser
  - Il progetto su cui si basa, «Web100», non e' piu' mantenuto, ma di recente e' stata rilasciata la versione «RC» del nuovo progetto Web10G

TCPDUMP

- TCPDUMP
- E’ uno strumento a riga di comando basato su libpcap
- Permette di impostare in modalita’ promiscua la scheda di rete e di vedere, o salvare su file (opzione –w), l’header di tutti i pacchetti che attraversano il link
- Si puo’ vedere anche il payload (opzione –s)

TCPDUMP

- Il suo comoporatamento puo' essere regolato con filtri molto granulari
- Possiamo filtrare per protocollo, porta, host, flag, ...
- Le opzioni di filtro possono essere combinate con operatori logici

TCPDUMO: un esempio

- Un esempio preso dal manuale di tcpdump

To print all packets arriving at or departing from *sundown*:

```
tcpdump host sundown
```

To print traffic between *helios* and either *hot* or *ace*:

```
tcpdump host helios and \( hot or ace \)
```

To print all IP packets between *ace* and any host except *helios*:

```
tcpdump ip host ace and not helios
```

To print all traffic between local hosts and hosts at Berkeley:

```
tcpdump net ucb-ether
```

To print all ftp traffic through internet gateway *snup*: (note that the expression is quoted to prevent the shell from (mis-)interpreting the parentheses):

```
tcpdump 'gateway snup and (port ftp or ftp-data)'
```

To print traffic neither sourced from nor destined for local hosts (if you gateway to one other net, this stuff should never make it onto your local net).

```
tcpdump ip and not net localnet
```

To print the start and end packets (the SYN and FIN packets) of each TCP conversation that involves a non-local host.

```
tcpdump 'tcp[tcpflags] & (tcp-syn|tcp-fin) != 0 and not src and dst net localnet'
```

To print all IPv4 HTTP packets to and from port 80, i.e. print only packets that contain data, not, for example, SYN and FIN packets and ACK-only packets. (IPv6 is left as an exercise for the reader.)

```
tcpdump 'tcp port 80 and (((ip[2:2] - ((ip[0]&0xf)<<2)) - ((tcp[12]&0xf0)>>2)) != 0)'
```

To print IP packets longer than 576 bytes sent through gateway *snup*:

```
tcpdump 'gateway snup and ip[2:2] > 576'
```

To print IP broadcast or multicast packets that were *not* sent via Ethernet broadcast or multicast:

```
tcpdump 'ether[0] & 1 = 0 and ip[16] >= 224'
```

To print all ICMP packets that are not echo requests/replies (i.e., not ping packets):

```
tcpdump 'icmp[icmptype] != icmp-echo and icmp[icmptype] != icmp-echoreply'
```

TCPTRACE

- TCPTRACE
- Permette di studiare i file raccolti con tcpdump ed altri strumenti simili
- Puo' riprodurre ed analizzare l'evoluzione della connessione tra due host e fornire dettagliate informazioni circa:
  - Durata della connessione,
  - byte e segment scambiati,
  - ritrasmissioni,
  - RTT,
  - Window,
  - Throughput
  - Etc..
- Se sul file tcpdump sono presenti piu' connessioni possiamo scegliere di analizzarne una in particolare piu' di una

TCPTRACE: un esempio

- Un esempio preso dal manuale di tcptrace

```
Beluga:/Users/mani> tcptrace -l malus.dmp.gz

1 arg remaining, starting with 'malus.dmp.gz'
Ostermann's tcptrace -- version 6.4.6 -- Tue Jul 1, 2003

32 packets seen, 32 TCP packets traced
elapsed wallclock time: 0:00:00.037948, 843 pkts/sec analyzed
trace file elapsed time: 0:00:00.404427
TCP connection info:
1 TCP connection traced:
TCP connection 1:
        host a:        elephus.cs.ohiou.edu:59518
        host b:        a17-112-152-32.apple.com:http
        complete conn: yes
        first packet:  Thu Jul 10 19:12:54.914101 2003
        last packet:   Thu Jul 10 19:12:55.318528 2003
        elapsed time:  0:00:00.404427
        total packets: 32
        filename:      malus.dmp.gz
   a->b:                              b->a:
     total packets:             16          total packets:             16
     ack pkts sent:             15          ack pkts sent:             16
     pure acks sent:            13          pure acks sent:             2
     sack pkts sent:             0          sack pkts sent:             0
     dsack pkts sent:            0          dsack pkts sent:            0
     max sack blks/ack:          0          max sack blks/ack:          0
     unique bytes sent:        450          unique bytes sent:      18182
     actual data pkts:           1          actual data pkts:          13
     actual data bytes:        450          actual data bytes:      18182
     rexmt data pkts:            0          rexmt data pkts:            0
     rexmt data bytes:           0          rexmt data bytes:           0
     zwnd probe pkts:            0          zwnd probe pkts:            0
     zwnd probe bytes:           0          zwnd probe bytes:           0
     outoforder pkts:            0          outoforder pkts:            0
     pushed data pkts:           1          pushed data pkts:           1
     SYN/FIN pkts sent:        1/1          SYN/FIN pkts sent:        1/1
     req 1323 ws/ts:           Y/Y          req 1323 ws/ts:           Y/Y
     adv wind scale:             0          adv wind scale:             0
     req sack:                   Y          req sack:                   N
     sacks sent:                 0          sacks sent:                 0
     urgent data pkts:           0 pkts     urgent data pkts:           0 pkts
     urgent data bytes:          0 bytes    urgent data bytes:          0 bytes
     mss requested:           1460 bytes    mss requested:           1460 bytes
     max segm size:            450 bytes    max segm size:           1448 bytes
     min segm size:            450 bytes    min segm size:            806 bytes
     avg segm size:            449 bytes    avg segm size:           1398 bytes
     max win adv:            40544 bytes    max win adv:            33304 bytes
     min win adv:             5840 bytes    min win adv:            33304 bytes
```

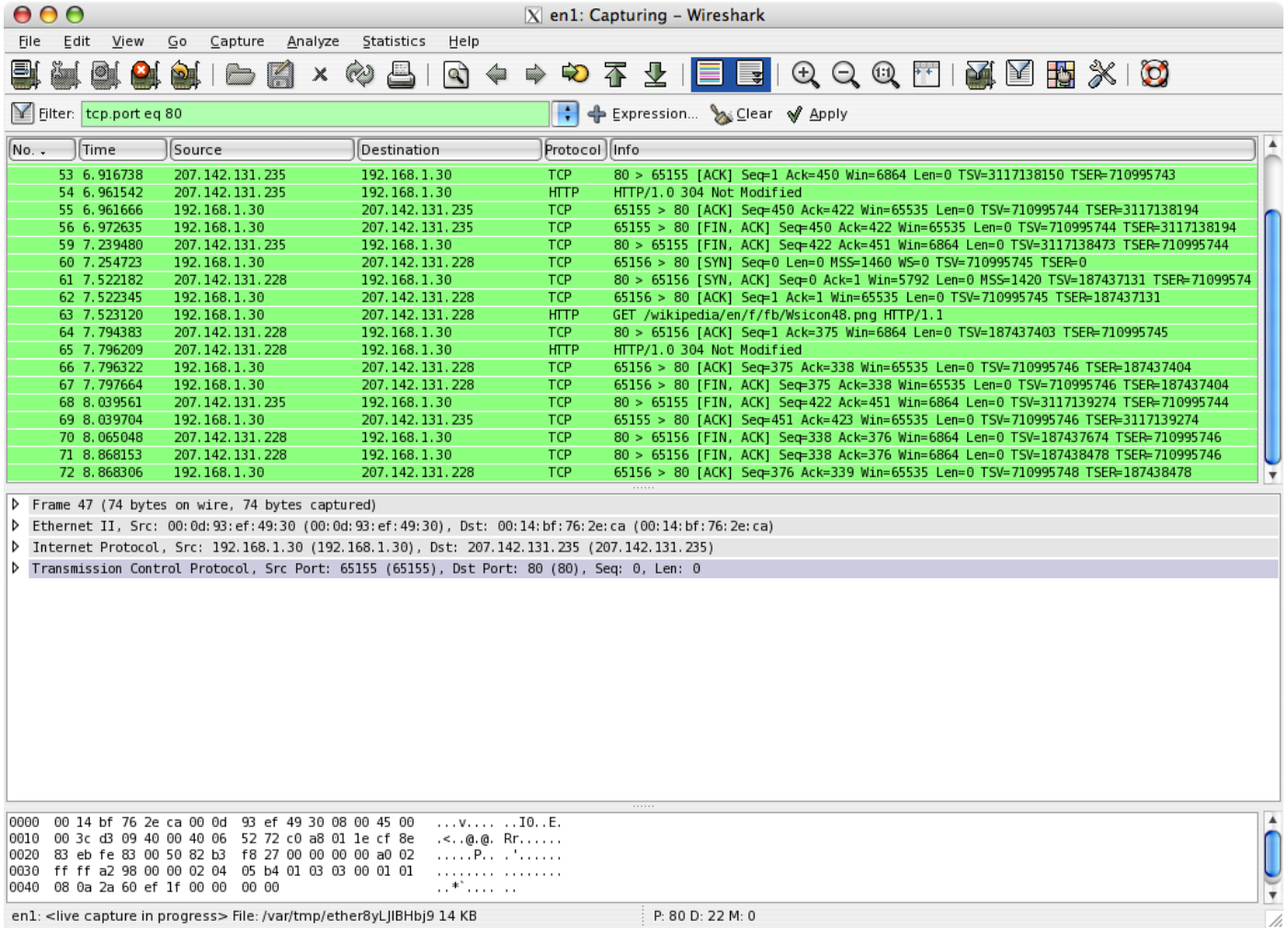
WIRESHARK

- WIRESHARK
- E’ un ulteriore strumento per la cattura e l’analisi del traffico di rete
- E’ compatibile con tcpdump e ne ingloba le caratteristiche
- E’ multipiattaforma: funziona su Linux, Windows, OSX, etc..
- Dispone di interfaccia grafica e riga di comando

WIRESHARK

- Possiamo eseguire la cattura e salvataggio del traffico con tcpdump e poi aprire il file in wireshark, oppure fare tutto direttamente da wireshark
- Il traffico puo' essere visto in tempo reale durante la cattura

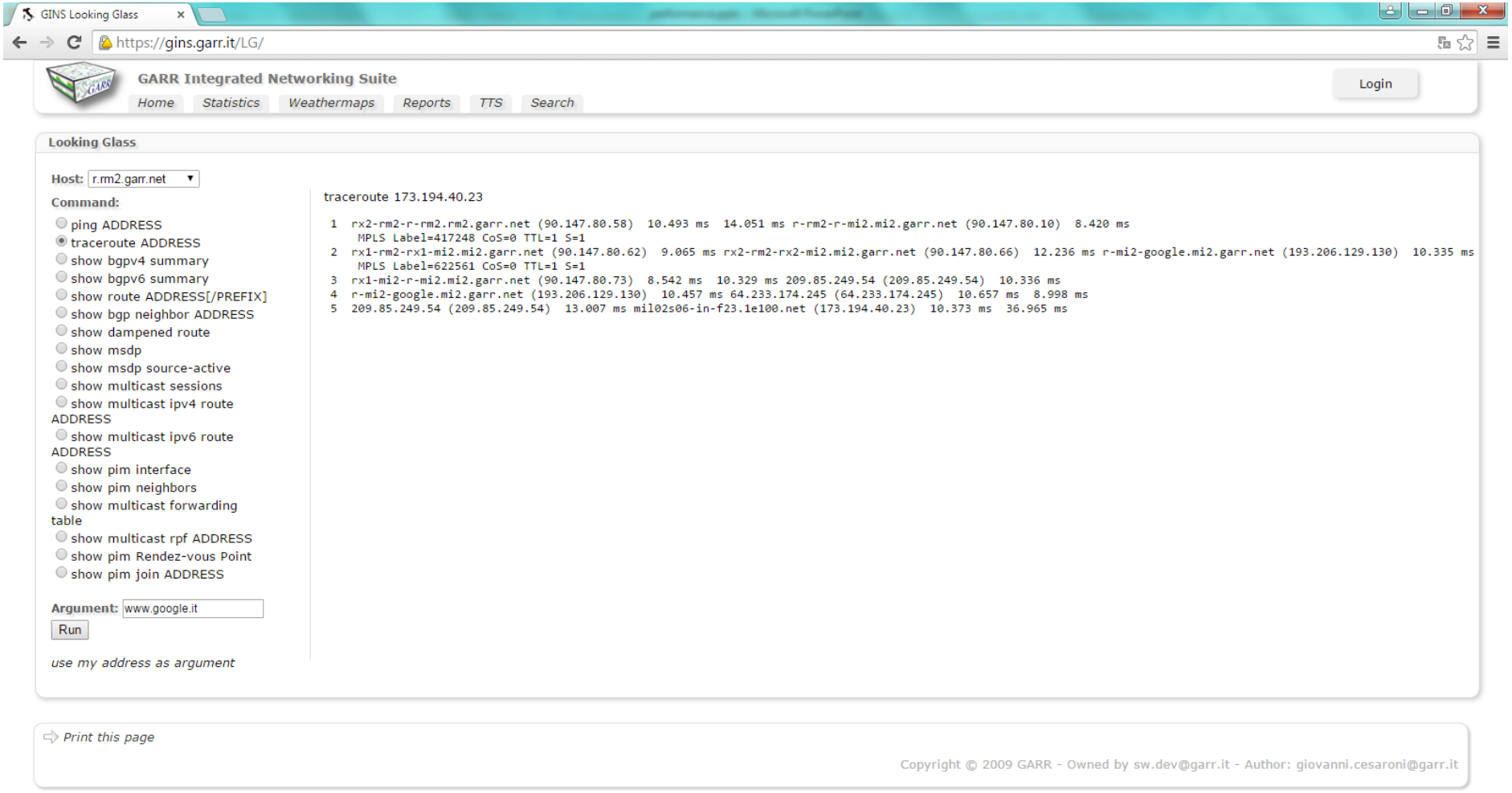
WIRESHARK


en1: Capturing – Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: tcp.port eq 80
Expression... Clear Apply
No. Time Source Destination Protocol Info
53 6.916738 207.142.131.235 192.168.1.30 TCP 80 > 65155 [ACK] Seq=1 Ack=450 Win=6864 Len=0 TSV=3117138150 TSER=710995743
54 6.961542 207.142.131.235 192.168.1.30 HTTP HTTP/1.0 304 Not Modified
55 6.961666 192.168.1.30 207.142.131.235 TCP 65155 > 80 [ACK] Seq=450 Ack=422 Win=65535 Len=0 TSV=710995744 TSER=3117138194
56 6.972635 192.168.1.30 207.142.131.235 TCP 65155 > 80 [FIN, ACK] Seq=450 Ack=422 Win=65535 Len=0 TSV=710995744 TSER=3117138194
59 7.239480 207.142.131.235 192.168.1.30 TCP 80 > 65155 [FIN, ACK] Seq=422 Ack=451 Win=6864 Len=0 TSV=3117138473 TSER=710995744
60 7.254723 192.168.1.30 207.142.131.228 TCP 65156 > 80 [SYN] Seq=0 Len=0 MSS=1460 WS=0 TSV=710995745 TSER=0
61 7.522182 207.142.131.228 192.168.1.30 TCP 80 > 65156 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1420 TSV=187437131 TSER=71099574
62 7.522345 192.168.1.30 207.142.131.228 TCP 65156 > 80 [ACK] Seq=1 Ack=1 Win=65535 Len=0 TSV=710995745 TSER=187437131
63 7.523120 192.168.1.30 207.142.131.228 HTTP GET /wikipedia/en/f/fb/Wsicon48.png HTTP/1.1
64 7.794383 207.142.131.228 192.168.1.30 TCP 80 > 65156 [ACK] Seq=1 Ack=375 Win=6864 Len=0 TSV=187437403 TSER=710995745
65 7.796209 207.142.131.228 192.168.1.30 HTTP HTTP/1.0 304 Not Modified
66 7.796322 192.168.1.30 207.142.131.228 TCP 65156 > 80 [ACK] Seq=375 Ack=338 Win=65535 Len=0 TSV=710995746 TSER=187437404
67 7.797664 192.168.1.30 207.142.131.228 TCP 65156 > 80 [FIN, ACK] Seq=375 Ack=338 Win=65535 Len=0 TSV=710995746 TSER=187437404
68 8.039561 207.142.131.235 192.168.1.30 TCP 80 > 65155 [FIN, ACK] Seq=422 Ack=451 Win=6864 Len=0 TSV=3117139274 TSER=710995744
69 8.039704 192.168.1.30 207.142.131.235 TCP 65155 > 80 [ACK] Seq=451 Ack=423 Win=65535 Len=0 TSV=710995746 TSER=3117139274
70 8.065048 207.142.131.228 192.168.1.30 TCP 80 > 65156 [FIN, ACK] Seq=338 Ack=376 Win=6864 Len=0 TSV=187437674 TSER=710995746
71 8.868153 207.142.131.228 192.168.1.30 TCP 80 > 65156 [FIN, ACK] Seq=338 Ack=376 Win=6864 Len=0 TSV=187438478 TSER=710995746
72 8.868306 192.168.1.30 207.142.131.228 TCP 65156 > 80 [ACK] Seq=376 Ack=339 Win=65535 Len=0 TSV=710995748 TSER=187438478
Frame 47 (74 bytes on wire, 74 bytes captured)
Ethernet II, Src: 00:0d:93:ef:49:30 (00:0d:93:ef:49:30), Dst: 00:14:bf:76:2e:ca (00:14:bf:76:2e:ca)
Internet Protocol, Src: 192.168.1.30 (192.168.1.30), Dst: 207.142.131.235 (207.142.131.235)
Transmission Control Protocol, Src Port: 65155 (65155), Dst Port: 80 (80), Seq: 0, Len: 0
0000 00 14 bf 76 2e ca 00 0d 93 ef 49 30 08 00 45 00 ...v.... ..I0..E.
0010 00 3c d3 09 40 00 40 06 52 72 c0 a8 01 1e cf 8e .<..@.@. Rr.....
0020 83 eb fe 83 00 50 82 b3 f8 27 00 00 00 00 a0 02 .....P.. .'......
0030 ff ff a2 98 00 00 02 04 05 b4 01 03 03 00 01 01 ....
0040 08 0a 2a 60 ef 1f 00 00 00 00 ..*`.... ..
en1: <live capture in progress> File: /var/tmp/ether8yLJIBHbj9 14 KB
P: 80 D: 22 M: 0

GARR Looking Glass

- Looking Glass
- Il GARR mette a disposizione dei suoi utenti un «looking glass» all'indirizzo:
  - http://www.noc.garr.it/index.php?option=com_content&view=article&id=130&Itemid=61
- Con questo trumento gli utenti possono autonomamente raccogliere informazioni statistiche e di routing dal punto di vista dei router GARR
- E' possibile eseguire ping, traceroute e vari comandi di «show» per molti protocolli di routing

GARR Looking Glass



Riferimenti

- Alcuni riferimenti oltre quelli gia' indicati:
- GARR NOC:
  - http://www.noc.garr.it/
- GEANT
  - http://kb.pert.geant.net/PERTKB/WebHome
- ESNet
  - http://fasterdata.es.net/host-tuning/

Domande?

- Domande?

Grazie

GRAZIE!