

TCP in pratica

l'esperimento

ricevitore veloce e rete lenta

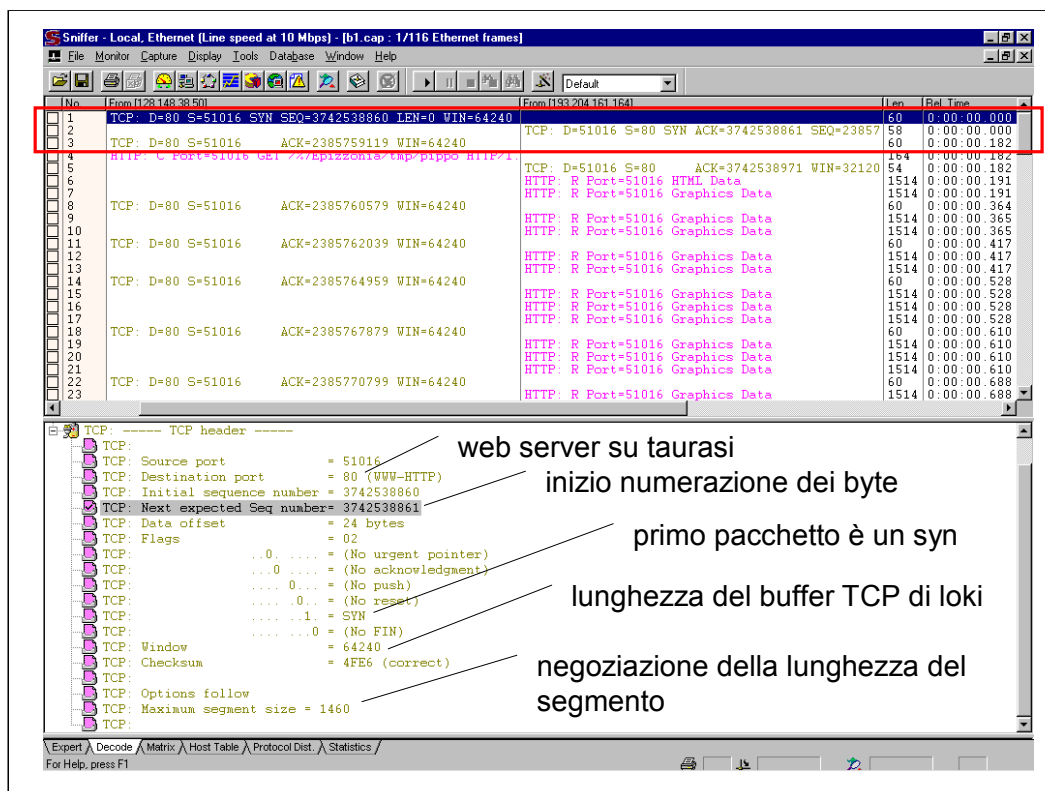
due calcolatori:

- taurasi 193.204.161.164
 - macchina dell'Università di Roma Tre (italia)
- loki 128.148.38.50
 - macchina presso la Brown University (USA)



l'esperimento

- su taurasi un web server attende una connessione (port 80)
- loki chiede al web server una file di circa 70 Kbyte, tramite HTTP (il protocollo HTTP viaggia dentro TCP)
- siamo interessati ad osservare i pacchetti TCP e come si comportano le due macchine durante il download e a capire **slow start**
- i pacchetti sono stati "sniffati" da taurasi



Osserviamo il primo pacchetto da loki a taurasi. Fa parte di un three-way-handshake. Loki avvia la connessione mandando un SYN (vedi il campo Flags del pacchetto 1). Il campo **destination port** è 80 (port standard per i web server), tale port è su taurasi. La **source port** è ininfluente e viene scelta arbitrariamente tra quelle libere di loki.

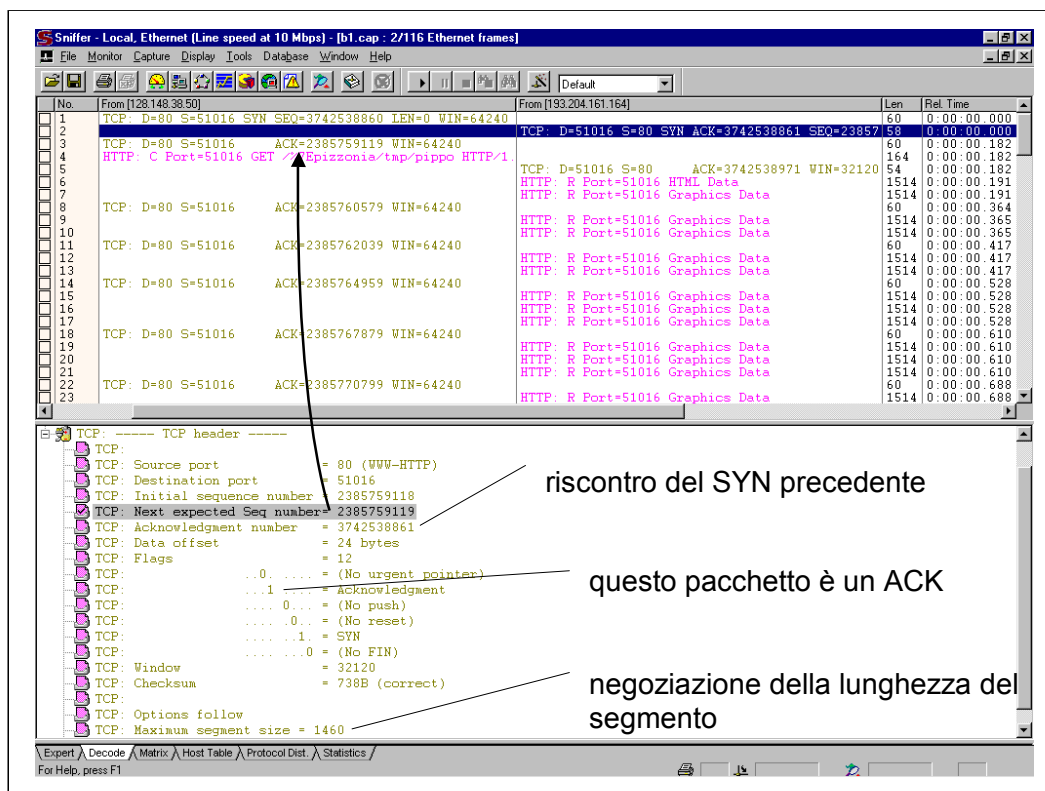
Nelle implementazioni di TCP la numerazione è sui byte e non sui pacchetti. Il campo **Initial sequence number** inizializza la sequenza di numerazione per il flusso di byte da loki a taurasi. Anche gli ACK (da taurasi a loki) saranno espressi in base a questa numerazione.

Il campo **Next expected Seq number** indica il numero del prossimo byte che verrà comunicato (questo campo non è presente nei segmenti TCP, viene calcolato dallo sniffer in base alla lunghezza del pacchetto). Nota come il SYN conta come un byte di dati.

Il campo **data offset** esprime la lunghezza dell'header TCP che stiamo esaminando.

Il campo **window** esprime quanto spazio rimane nel buffer di loki. In realtà il buffer in questo momento il buffer è vuoto (nessun dato ricevuto da loki), quindi tale numero indica l'ampiezza del buffer.

Il primo pacchetto ha una opzione: **Maximum segment size** che indica che loki desidera segmenti lunghi al più 1460 bytes.



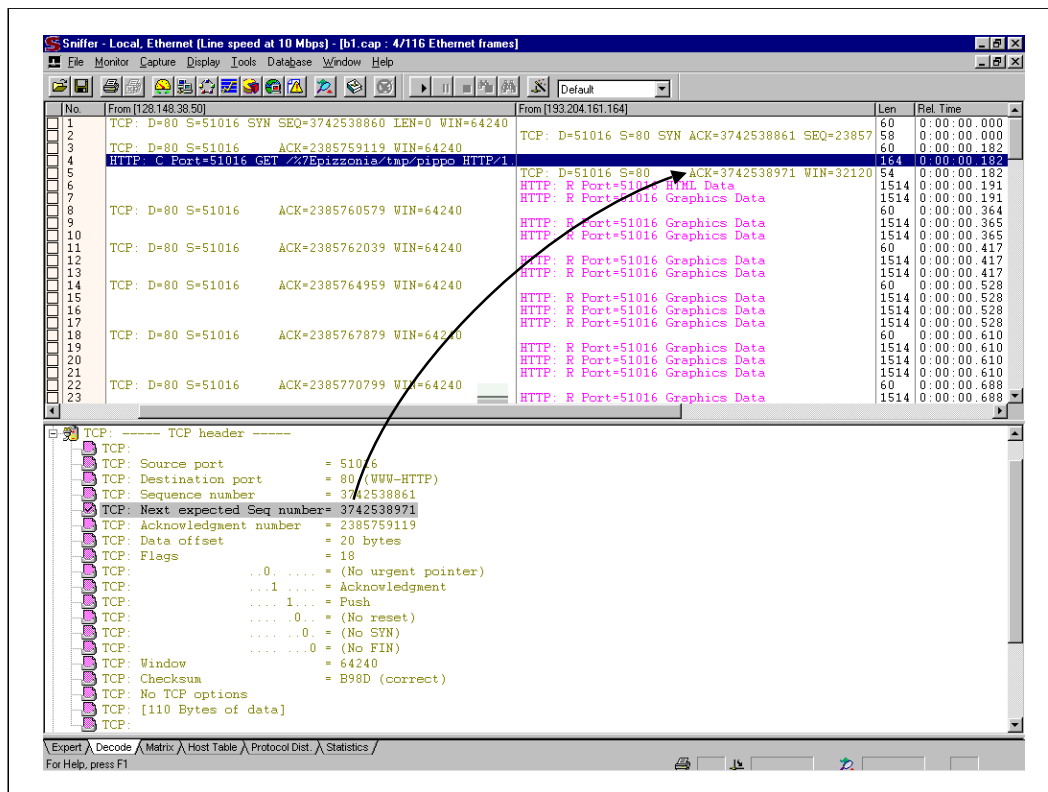
Il secondo pacchetto SYN+ACK comunica a loki l'inizio della sequenza per i byte trasmessi da taurasi.

In questo pacchetto viene riscontrato il SYN precedente, infatti il bit di ACK è a 1, quindi il campo ACK è valido, infatti lo sniffer fa vedere un campo **acknowledgment number** il cui numero è uguale a quello del campo **next expected seq number** nel pacchetto 1. Da questo loki capisce che il pacchetto 1 è stato ricevuto correttamente.

Notare come taurasi annuncia una finestra di controllo di flusso pari a 32120, mentre loki aveva annunciato una finestra di 64240 (loki è una sun ultra 2 mentre taurasi è una macchina linux).

Per comodità lo sniffer fa vedere nella lista dei pacchetti il contenuto del campo **acknowledgment number**. Facilmente possiamo vedere che questo pacchetto è riscontrato dal successivo pacchetto numero 3.

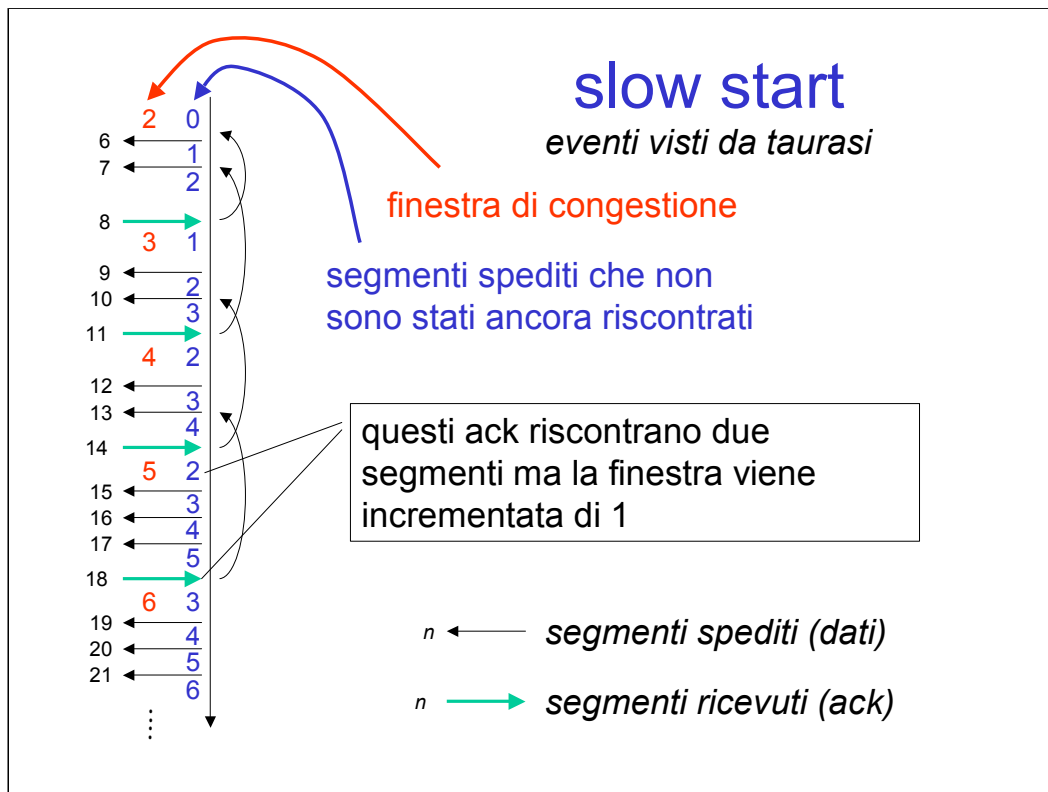
La procedura di apertura della connessione finisce con il pacchetto 3.



Il pacchetto 4 contiene una richiesta di un file. Il pacchetto 5 riscontra il del pacchetto 4.

L'effettivo flusso di dati da taurasi a loki avviene a partire dal pacchetto 6.

Da ora in poi tutti i pacchetti da loki a taurasi conterranno dei dati (1514 bytes) e tutti i pacchetti da taurasi a loki saranno degli ack (60 bytes).



La numerazione nelle sessioni TCP è basata sui byte, tuttavia, è possibile riportare tutti i ragionamenti ad una numerazione sui pacchetti, in questo caso l'operazione è particolarmente semplice poiché i segmenti hanno tutti la stessa dimensione pari a MSS. Nella slide per semplicità utilizziamo una numerazione in cui l'unità di conto è il pacchetto.

I dati sono stati "intuiti" in base all'osservazione del traffico. La soglia per slow start non è nota.

TCP può spedire pacchetti finché la finestra di congestione non è satura. Quando il pacchetto 6 parte da taurasi la finestra di congestione è pari a 2 e nessun pacchetto mandato è ancora senza riscontro, quindi taurasi manda due pacchetti ed attende un ack.

Il pacchetto 8 è l'ack per 6. L'arrivo di un ack fa incrementare la finestra di congestione di una unità. Nota come la finestra di congestione viene incrementata di 1 per ogni ack. Questo mi dà una crescita esponenziale della finestra.

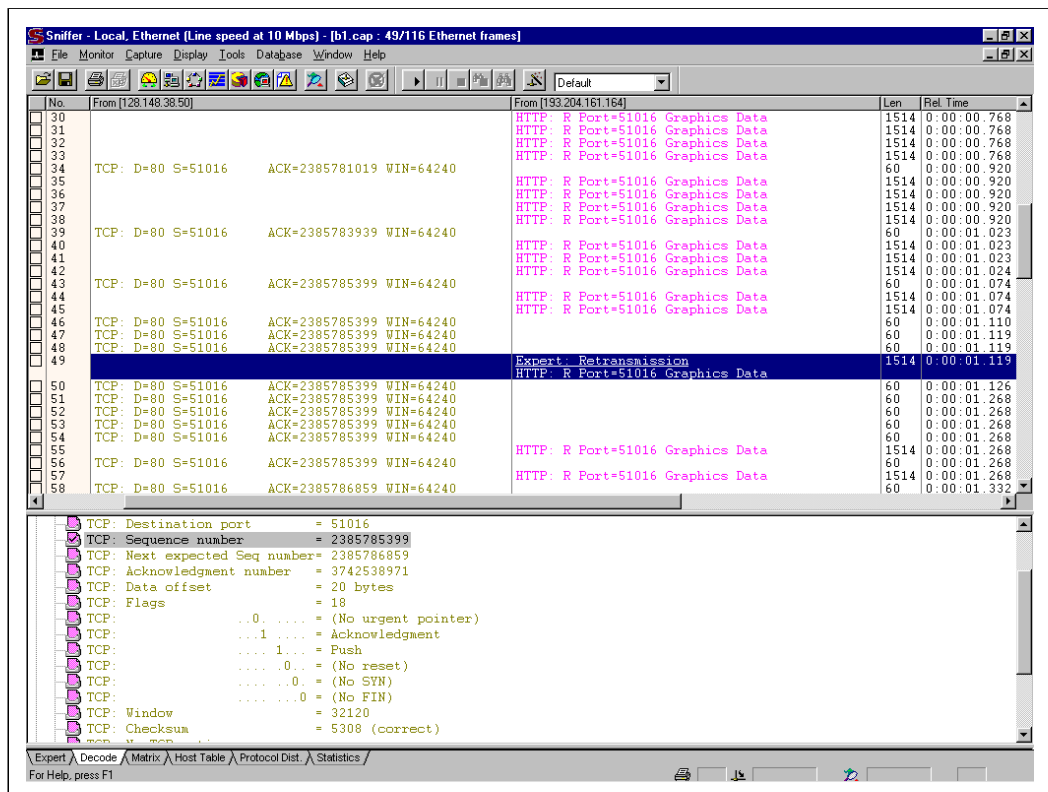
| Sniffer - Local Ethernet (Line speed at 10 Mbps) - [b1.cap : 30/116 Ethernet frames] | | | | | | | |
|--|--|--|----------------------------------|--|------|-------------|--|
| No. | From [128.148.38.50] | | From [193.204.161.164] | | Len | Rel. Time | |
| 25 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.688 | |
| 26 | TCP: D=80 S=51016 ACK=2385772259 WIN=64240 | | HTTP: R Port=51016 Graphics Data | | 60 | 0:00:00.735 | |
| 27 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.735 | |
| 28 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.735 | |
| 29 | TCP: D=80 S=51016 ACK=2385776639 WIN=64240 | | | | 60 | 0:00:00.768 | |
| 30 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.768 | |
| 31 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.768 | |
| 32 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.768 | |
| 33 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.768 | |
| 34 | TCP: D=80 S=51016 ACK=2385781019 WIN=64240 | | | | 60 | 0:00:00.920 | |
| 35 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.920 | |
| 36 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.920 | |
| 37 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.920 | |
| 38 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:00.920 | |
| 39 | TCP: D=80 S=51016 ACK=2385783939 WIN=64240 | | | | 60 | 0:00:01.023 | |
| 40 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:01.023 | |
| 41 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:01.023 | |
| 42 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:01.024 | |
| 43 | TCP: D=80 S=51016 ACK=2385785399 WIN=64240 | | | | 60 | 0:00:01.074 | |
| 44 | | | HTTP: R Port=51016 Graphics Data | | 1514 | 0:00:01.074 | |
| 45 | TCP: D=80 S=51016 ACK=2385785399 WIN=64240 | | | | 1514 | 0:00:01.074 | |
| 46 | TCP: D=80 S=51016 ACK=2385785399 WIN=64240 | | | | 60 | 0:00:01.110 | |
| 47 | TCP: D=80 S=51016 ACK=2385785399 WIN=64240 | | | | 60 | 0:00:01.119 | |
| 48 | TCP: D=80 S=51016 ACK=2385785399 WIN=64240 | | | | 60 | 0:00:01.119 | |
| 49 | | | Expert: Retransmission | | 1514 | 0:00:01.119 | |
| | | | HTTP: R Port=51016 Graphics Data | | | | |
| 50 | TCP: D=80 S=51016 ACK=2385785399 WIN=64240 | | | | 60 | 0:00:01.126 | |
| 51 | TCP: D=80 S=51016 ACK=2385785399 WIN=64240 | | | | 60 | 0:00:01.268 | |
| 52 | TCP: D=80 S=51016 ACK=2385785399 WIN=64240 | | | | 60 | 0:00:01.268 | |
| 53 | TCP: D=80 S=51016 ACK=2385785399 WIN=64240 | | | | 60 | 0:00:01.268 | |

| | |
|-------------------------------|-------------------------------|
| TCP: | |
| TCP: Source port | = 80 (WWW-HTTP) |
| TCP: Destination port | = 51016 |
| TCP: Sequence number | = 2385783939 |
| TCP: Next expected Seq number | = 2385785399 |
| TCP: Acknowledgment number | = 3742538971 |
| TCP: Data offset | = 20 bytes |
| TCP: Flags | = 18 |
| TCP: | ...0... = (No urgent pointer) |
| TCP: | ...1... = Acknowledgment |
| TCP: | ...1... = Push |
| TCP: | ...0... = (No reset) |
| TCP: | ...0... = (No SYN) |
| TCP: | ...0... = (No FIN) |

Il pacchetto 49 è un esempio di ritrasmissione. Cerchiamo di capire cosa è successo. Il pacchetto 30 è regolarmente riscontrato con l'ack 43.

TCP ogni volta che riceve un pacchetto fuori sequenza non può inviare un ack selettivo ma rimanda l'ultimo ack inviato.

Gli ack da 46 in poi sono tutte copie di 43. Evidentemente il pacchetto 31 è andato perso (probabilmente a causa di un router congestionato) e quindi la ricezione dei pacchetti da 32 in poi causa l'invio di una sequenza di ack replicati. Dopo un certo numero di ack (nel nostro caso 4 ma dipende dall'implementazione) taurasi reinvia il pacchetto (pacchetto 49).



Il fatto che il pacchetto 49 sia una ritrasmissione è evidenziato dallo sniffer ma si può evincere dal fatto che il **sequence number** di tale pacchetto è pari a quello dell'ack ripetuto.

L'ack per il pacchetto 49 arriva regolarmente dopo un po'. Confronta l'ack 58 con il **next expected sequence number** del pacchetto 49.

esperimento

ricevitore lento e rete veloce

- due calcolatori sulla stessa rete locale:
 - 10.0.0.2 è un web server
 - 10.0.0.3 è un client
- il client richiede circa 60 Kbyte al server
- questa volta vogliamo analizzare il comportamento della **finestra di controllo di flusso**

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|----------|----------|-------------|----------|--------|---|
| 1 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 60 → 80 [RST] Seq=16734436 Win=0 Len=0 |
| 2 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8192 Len=0 |
| 3 | 0.000000 | 10.0.0.2 | 10.0.0.3 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 4 | 0.000000 | 10.0.0.2 | 10.0.0.3 | HTTP | 244 | 200 OK |
| 5 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 6 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 7 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 8 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 9 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 10 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 11 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 12 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 13 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 14 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 15 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 16 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 17 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 18 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 19 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=2920 Len=0 |
| 20 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=2920 Len=0 |
| 21 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=2920 Len=0 |
| 22 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=0 Len=0 |
| 23 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 24 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 25 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 26 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 27 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 28 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 29 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 30 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=8760 Len=0 |
| 31 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 32 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 33 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 34 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 35 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 36 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 37 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 38 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 39 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 40 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 41 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 42 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 43 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 44 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 45 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 46 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 47 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |
| 48 | 0.000000 | 10.0.0.3 | 10.0.0.2 | TCP | 60 | 80 → 80 [ACK] Seq=16734437 Win=1460 Len=0 |

Anche qui il flusso di dati inizia al pacchetto 6.

La macchina ricevente è un sistema Windows ed annuncia una finestra di 8760, che corrisponde a 6 pacchetti poiché durante il three-way-handshake il **maximum segment size** è risultato essere di 1460.

L'ack 7 provoca l'apertura della finestra di *congestione* a 2 pacchetti e annuncia una finestra di controllo di flusso sempre di 8760. Questo significa che l'applicazione ha fatto in tempo a leggere i buffer prima che venisse mandato l'ack. Seguono due pacchetti. Dopo l'ack 10 seguono 3 pacchetti e 4 dopo l'ack numero 14. Attenzione: il pacchetto 19 riscontra tutti i pacchetti fino al 18 ma annuncia una finestra di soli 2920 bytes. Questo perché l'applicazione non ha fatto in tempo a consumare i dati. il server può mandare solo due segmenti (20 e 21), pur avendo una finestra di congestione maggiore. L'ack 22 riscontra gli ultimi due segmenti ma la finestra di controllo di flusso è zero. Il server attende che l'applicazione sul client consumi dei dati. L'ack 23 ha lo scopo di annunciare una finestra più larga (8760 bytes). Nota come tale ack è ridondante per quanto riguarda il riscontro di bytes infatti riscontra gli stessi bytes del pacchetto 22. I pacchetti 24-29 saturano ancora la finestra (vedi ack 30). L'ack 31 annuncia che la finestra si è aperta un po', soli 1460 bytes, ed il server prontamente manda un solo pacchetto.

Morale gli ack possono essere generati anche dall'applicazione che consuma bytes per notificare che la finestra di controllo di flusso si è cambiata.