

Esercitazione Ethereal sul TCP

Questa esercitazione mira ad investigare i segmenti TCP spediti e ricevuti nel trasferimento di un file di 150 KB dal proprio computer ad un server remoto.

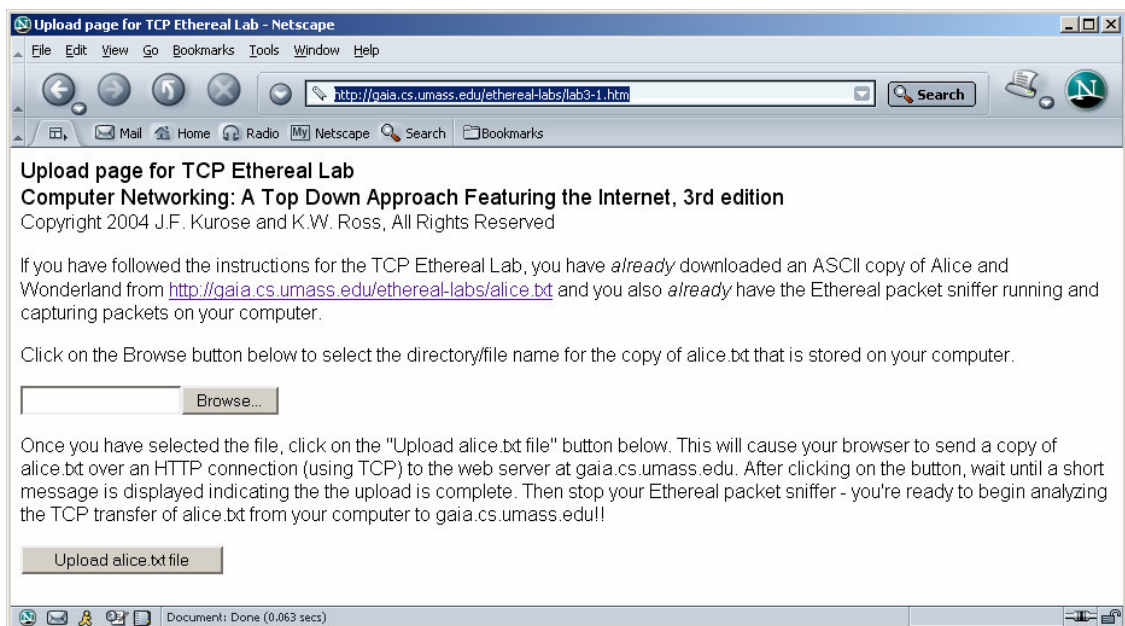
In particolare si sofferma sul numero di *acknowledgment* per fornire un trasferimento affidabile di dati; si osserverà, anche, l'algoritmo di controllo della congestione in azione.

Verrà, infine, brevemente investigata la performance (*throughput* e *round-trip time*) della connessione TCP tra il proprio computer ed il server.

1. Cattura del trasferimento TCP dal proprio computer ad un server remoto

Eeguire le seguenti:

- Immagazzinare sul proprio computer la seguente risorsa <http://gaia.cs.umass.edu/ethereal-labs/alice.txt> .
- Visitare <http://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html>, ottenendo una schermata simile a quella sottostante:



- Usare il pulsante *Browse per inserire il nome del percorso completo* del file sul proprio computer. Non si preme il pulsante *“Upload alice.txt file”*.
- Avviare la cattura del pacchetto tramite Ethereal.

- Nel proprio browser premere il pulsante “*Upload alice.txt file*” per caricare il file dal server `gaia.cs.umass.edu`. Una volta caricato il file all’interno della finestra del browser usato apparirà un breve testo di congratulazioni.
- Fermare Ethereal.

2. Un primo sguardo alla traccia catturata

Prima di analizzare il comportamento della connessione TCP in dettaglio, è opportuno ottenere una vista ad alto livello del file di traccia catturato.

A tale scopo si effettui un filtro dei pacchetti visualizzati nella finestra Ethereal inserendo “tcp” nel campo di specifica del filtro posto in alto in tale finestra.

Si dovrebbero osservare le sequenze di messaggi TCP e HTTP scambiati tra il proprio computer e `gaia.cs.umass.edu`. Si dovrebbe anche vedere l’iniziale *three-way handshake* contenente un messaggio *SYN*. Inoltre si dovrebbero vedere messaggi HTTP POST e una serie di “HTTP Continuation” spediti dal proprio computer verso `gaia.cs.umass.edu`. (A seconda delle impostazioni e della versione di Ethereal è possibile che al posto di HTTP Continuation ci sia “tcp segment of a reassembled pdu” dove PDU stà per Personal Data Unit ossia dati scambiati tra entità dello stesso livello).

Si osservi che nella realtà non esistono nè i messaggi “HTTP Continuation” nè i messaggi “tcp segment of a reassembled pdu” – questo è un modo di Ethereal per indicare che ci sono molteplici segmenti che sono usati per eseguire un singolo messaggio HTTP. Si dovrebbero anche vedere i segmenti *TCP ACK* che sono stati restituiti da `gaia.cs.umass.edu` al proprio computer.

Nel fornire risposta alle domande si dovrebbe produrre un tabulato, quando possibile, dei pacchetti che hanno permesso di rispondere alla domanda. Il tabulato è da usare per spiegare la risposta. Per ottenere il tabulato di un pacchetto selezionare *File->Print*, scegliere *Selected packet only*, scegliere *Packet summary line*, ed infine selezionare la quantità minima di pacchetti di cui si ha bisogno per fornire risposta alla domanda.

Poichè tale esercitazione è rivolta al TCP piuttosto che all’HTTP, cambiare la finestra “elenco dei pacchetti catturati” di Ethereal in modo che siano mostrati i segmenti TCP contenenti i messaggi HTTP piuttosto che i messaggi HTTP. A tal proposito selezionare *Analyze->Enabled Protocols*, togliere la spunta sulla casella HTTP box e cliccare su *OK*. Inoltre, poiché in questa esercitazione si vogliono vedere i numeri della sequenza TCP (e non i numeri della sequenza relativa che Ethereal invece visualizza) si operi come segue: andare su *Edit>Preferences>Protocols>IP* e togliere la spunta da “relative sequence numbers”. In questo modo si ottiene una sequenza di segmenti TCP spediti tra il proprio computer e `gaia.cs.umass.edu`. Si utilizzerà la traccia catturata per studiare il comportamento TCP nella parte restante della presente esercitazione.

3. Principi fondamentali di TCP

Rispondere alle seguenti domande per i segmenti TCP:

1. Quali sono l'indirizzo IP ed il numero della porta TCP usati dal proprio client (source) per trasferire il file a `gaia.cs.umass.edu`? Quali sono l'indirizzo IP ed il numero di porta usati da `gaia.cs.umass.edu` per ricevere il file?

IP address: 192.168.1.102

Numero di porta TCP: 1161

Computer di destinazione: `gaia.cs.umass.edu`

Indirizzo IP: 128.119.245.12

Numero di porta TCP: 80

2. Quale è il numero di sequenza usato del segmento *TCP SYN* che è usato per iniziare la connessione TCP tra il computer client e `gaia.cs.umass.edu`? Cosa c'è nel segmento che lo identifica come *SYN segment*?

Il numero di sequenza del segmento TCP SYN è usato per iniziare la connessione TCP tra il computer client e `gaia.cs.umass.edu`. Il valore è 0 nella traccia memorizzata.

Il flag SYN flag è impostato ad 1 ed indica che questo è un segmento SYN.

3. Qual è il numero di sequenza del segmento *SYNACK* spedito da `gaia.cs.umass.edu` al computer client in risposta al *SYN*? Qual è il valore del campo di *ACKnowledgement* nel segmento *SYNACK*? Come `gaia.cs.umass.edu` ha determinato quel valore? Cosa c'è nel segmento che lo identifica come *SYNACK segment*?

Il numero di sequenza del segmento da `gaia.cs.umass.edu` al computer client in risposta al SYN ha il valore 0 in questa traccia.

Il valore del campo *ACKnowledgement* nel segmento *SYNACK* è 1. Tale valore è determinato da `gaia.cs.umass.edu` mediante l'aggiunta di 1 al numero iniziale di sequenza del segmento SYN dal computer client (ossia il numero di sequenza del segmento SYN iniziato dal client è 0).

Il flag SYN ed il flag *Acknowledgement* nel segmento sono impostati ad 1 ed essi indicano che quello è un segmento *SYNACK*.

4. Qual è il numero di sequenza del segmento TCP contenente il comando HTTP POST? Si osservi che per trovare il comando POST si avrà bisogno di guardare nel campo del contenuto del pacchetto posto in basso nella finestra Ethereal, alla ricerca di un segmento con "POST" all'interno del proprio campo DATA.

Il segmento numero 4 è il segmento che contiene il comando HTTP POST. Il numero di sequenza di tale segmento ha valore 1.

5. Si consideri il segmento TCP contenente l'HTTP POST come il primo segmento nella connessione TCP. Quali sono i numeri di sequenza dei primi sei segmenti nella connessione TCP (incluso il segmento che contiene l'HTTP POST)? In quale momento ogni segmento è spedito? In quale momento è ricevuto l'ACK per ogni segmento? Considerata la differenza tra il momento in cui ogni segmento TCP è stato spedito e quello in cui il proprio acknowledgement è stato ricevuto, qual è il valore di RTT per ognuno dei sei segmenti? Qual è il valore di EstimatedRTT (lucidi n° 23-24 da [1]) dopo la ricezione di ogni ACK? Si assuma che il valore dell'EstimatedRTT sia uguale all'RTT misurato per il primo segmento e poi sia calcolato usando l'equazione di EstimatedRTT di (lucidi n° 23-24 da [1]) per tutti i successivi segmenti.

Nota: Ethereal permette di tracciare l'RTT di ogni segmento inviato. Si selezioni un segmento TCP nella finestra "listing of captured packets" che è stato spedito dal client al server gaia.cs.umass.edu. Poi si selezioni: *Statistics->TCP Stream Graph->Round Trip Time Graph.*

Il segmento di HTTP POST è considerato come il primo segmento. I segmenti 1-6 sono, rispettivamente, i numeri 4, 5, 7, 8, 10, e 11 della traccia. Gli ACK dei segmenti 1-6 sono i numeri 6, 9, 12, 14, 15 e 16 della traccia.

Numero di sequenza del segmento 1: 1
 Numero di sequenza del segmento 2: 566
 Numero di sequenza del segmento 3: 2026
 Numero di sequenza del segmento 4: 3486
 Numero di sequenza del segmento 5: 4946
 Numero di sequenza del segmento 6: 6406

I tempi di invio e di ricezione degli ACK sono riportati nella seguente tabella.

	Istante di invio	Istante di ricezione ACK	RTT (secondi)
Segment 1	0.026477	0.053937	0.02746
Segment 2	0.041737	0.077294	0.035557
Segment 3	0.054026	0.124085	0.070059
Segment 4	0.054690	0.169118	0.11443
Segment 5	0.077405	0.217299	0.13989
Segment 6	0.078157	0.267802	0.18964

$$\text{EstimatedRTT} = 0.875 * \text{EstimatedRTT} + 0.125 * \text{SampleRTT}$$

EstimatedRTT dopo la ricezione dell'ACK del segmento 1:
 EstimatedRTT = RTT for Segment 1 = 0.02746 secondi

EstimatedRTT dopo la ricezione dell'ACK del segmento 2:
 EstimatedRTT = $0.875 * 0.02746 + 0.125 * 0.035557 = 0.0285$

EstimatedRTT dopo la ricezione dell'ACK del segmento 3:
 $\text{EstimatedRTT} = 0.875 * 0.0285 + 0.125 * 0.070059 = 0.0337$

EstimatedRTT dopo la ricezione dell'ACK del segmento 4:
 $\text{EstimatedRTT} = 0.875 * 0.0337 + 0.125 * 0.11443 = 0.0438$

EstimatedRTT dopo la ricezione dell'ACK del segmento 5:
 $\text{EstimatedRTT} = 0.875 * 0.0438 + 0.125 * 0.13989 = 0.0558$

EstimatedRTT dopo la ricezione dell'ACK del segmento 6:
 $\text{EstimatedRTT} = 0.875 * 0.0558 + 0.125 * 0.18964 = 0.0725$

6. Qual è la lunghezza di ognuno dei primi sei segmenti?

Lunghezza del primo segmento (contenente dell'HTTP POST): 565 byte
Lunghezza di ognuno degli altri 5 segmenti TCP: 1460 byte (MSS)

7. Qual è la quantità minima di spazio di buffer che può essere osservata su gaia.cs.umass.edu per l'intera traccia del file?

La quantità minima di spazio di buffer (finestra del ricevente) riscontrata su gaia.cs.umass.edu per l'intera traccia è di 5840 byte, visibile nel primo acknowledgement del server. Tale finestra del ricevente cresce costantemente fino ad un massimo di dimensione del buffer del ricevente di 62780 byte.

8. Ci sono alcuni segmenti ritrasmessi nel file di traccia? Cosa si controlla (nel file di traccia) al fine di rispondere a tale domanda?

Non c'è alcun segmento ritrasmesso. E' possibile verificarlo controllando i numeri di sequenza nel file di traccia. Nel grafo Sequence-Graph (Stevens) di tale traccia, tutti i numeri di sequenza del sorgente (192.168.1.102) alla destinazione (128.119.245.12) sono monotonicamente crescenti rispetto al tempo. Se ci fosse un segmento ritrasmesso, il numero di sequenza del segmento ritrasmesso dovrebbe essere più piccolo di quello dei segmenti vicini.

9. Di solito quanti dati il ricevente riconosce in un ACK? Si potrebbero identificare casi in cui il ricevente stà riconoscendo un qualunque altro segmento ricevuto (lucido 25 da [1])?

I numeri di sequenza degli ACK sono i seguenti:

	Numero di sequenza ricevuto	Dati riconosciuti
ACK 1	566	566
ACK 2	2026	1460
ACK 3	3486	1460

ACK 4	4946	1460
ACK 5	6406	1460
ACK 6	7866	1460
ACK 7	9013	1147
ACK 8	10473	1460
ACK 9	11933	1460
ACK 10	13393	1460
ACK 11	14853	1460
ACK 12	16313	1460

La differenza tra di due numeri di sequenza di 2 ACK consecutivi indica i dati ricevuti tra questi due ACK. Ispezionando i dati riconosciuti da ogni ACK, ci sono casi in cui il ricevente stà riconoscendo ogni altro segmento. Ad esempio il segmento numero 80 i cui dati riconosciuti sono: 2920 byte = 1460*2 byte.

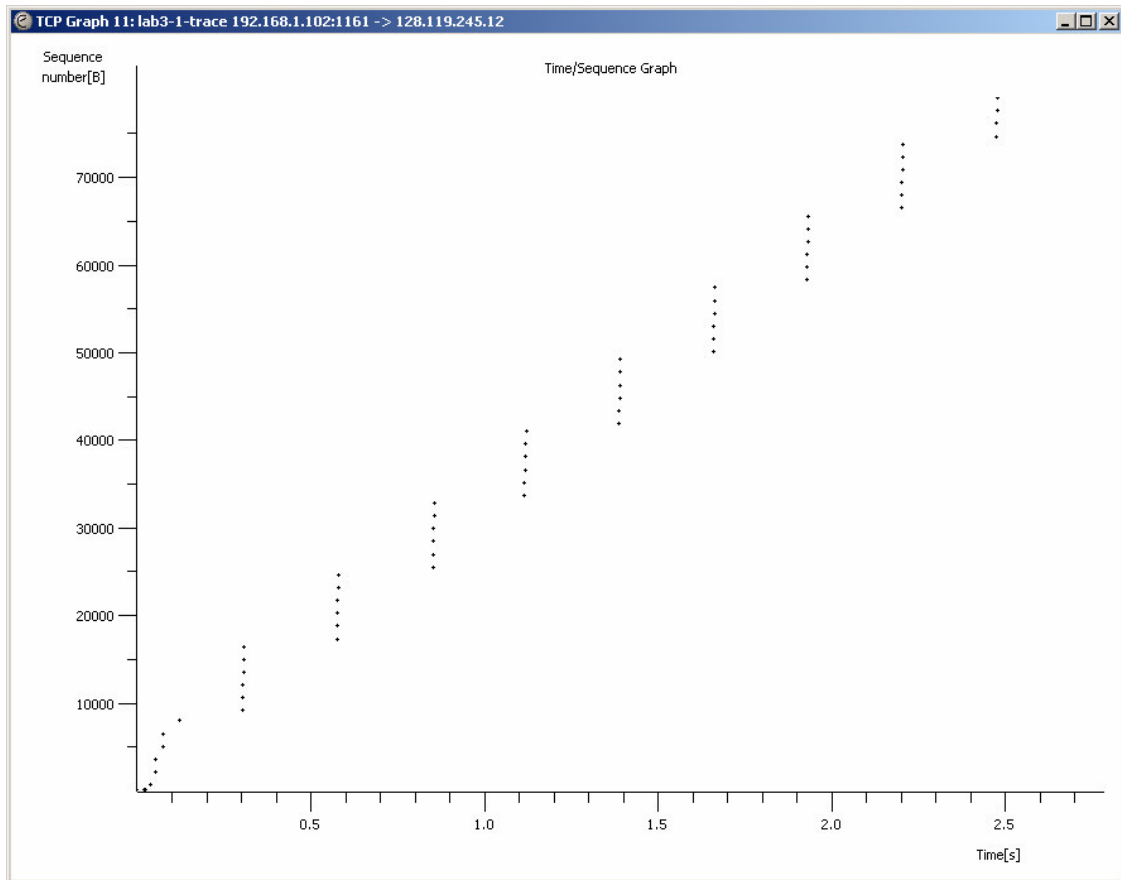
10. Qual è il *throughput* (byte trasferiti per unità di tempo) per la connessione TCP? Fornire una spiegazione di come si è calcolato tale valore?

Il calcolo del throughput TCP in gran parte dipende dalla selezione del periodo medio. Per il calcolo del throughput di tale domanda si seleziona come periodo medio l'intero tempo di connessione. Perciò il throughput medio per questa connessione TCP è calcolato come rapporto tra la quantità totale di dati ed il tempo totale di trasmissione. La quantità totale di dati può essere calcolata mediante la differenza tra il numero di sequenza del primo segmento TCP (ossia 1 byte per il segmento numero 4) ed il numero di sequenza dell'ultimo ACK riconosciuto (164091 byte per il numero 202). Perciò la quantità di dati complessiva è $164091 - 1 = 164090$ byte. L'intero tempo di trasmissione corrisponde all'istante di tempo del primo segmento TCP (ossia 0.026477 secondi per il segmento numero 4) e l'istante di tempo dell'ultimo ACK (ossia 5.455830 secondi per il segmento numero 202). Perciò il tempo totale di trasmissione è $5.455830 - 0.026477 = 5.4294$ secondi. Di conseguenza il throughput per la connessione TCP è calcolato come $164090/5.4294 = 30.222$ KByte/sec.

4. TCP: controllo della congestione in azione

Si esamini ora la quantità di dati spediti per unità di tempo dal client al server. Piuttosto che calcolarlo dai dati delle righe presenti nella finestra Ethereal si usi una delle utilità grafiche di Ethereal - *Time-Sequence-Graph(Stevens)* – per tracciare i dati.

- Si selezioni un segmento TCP nella finestra di “elenco dei pacchetti catturati”. Quindi si selezioni il menù : *Statistics->TCP Stream Graph-> Time-Sequence-Graph(Stevens)*. Si dovrebbe ottenere un grafico che sembri simili a quello sottostante:



In tale grafico ogni punto rappresenta un segmento TCP spedito, in corrispondenza del numero di sequenza del segmento e del momento in cui è stato spedito. Si osservi che un insieme di punti impilato l'uno sull'altro rappresenta una sequenza di pacchetti che sono stati spediti uno dopo l'altro dal mittente.

Rispondere alle seguenti domande:

11. Usare il tool *Time-Sequence-Graph(Stevens)* per vedere il numero di sequenza rispetto al momento in cui i segmenti sono stati spediti dal client al server `gaia.cs.umass.edu`. E' possibile identificare quando la fase "partenza lenta" inizia e finisce e quando termina la fase di "prevenzione della congestione"? Si osservi che nella realtà non tutto è abbastanza accurato e chiaro come si potrebbe pensare dal tool usato.

La fase di partenza lenta del TCP inizia alla partenza della connessione, cioè quando il segmento di HTTP POST è spedito. L'identificazione della fase di partenza lenta del TCP ed il controllo della congestione dipendono dal valore della dimensione della finestra di congestione del mittente TCP.

Tale valore, comunque, non può essere ottenuto direttamente dal grafico di *Time-Sequence-Graph (Stevens)*. Tuttavia, è possibile stimare il limite inferiore della dimensione della finestra, per mezzo della quantità di dati di tipo *outstanding* poiché questi costituiscono la totalità dei dati senza *acknowledgement*. Inoltre è noto che la

finestra TCP è vincolata dalla dimensione della finestra del ricevente e che il buffer del ricevente può agire come limite superiore della dimensione della finestra. Nella traccia memorizzata il buffer del ricevente non è il collo di bottiglia; pertanto tale limite superiore non è sufficientemente utile per inferire la dimensione della finestra TCP. Quindi, ci si focalizzerà sul limite inferiore della dimensione della finestra TCP.

Dalla seguente tabella non è possibile vedere che la totalità dei dati *outstanding* si incrementa velocemente alla partenza del flusso TCP e che, comunque, non eccede mai gli 8192 Byte. Perciò è possibile essere certi che la dimensione della finestra TCP è più grande di 8192 Byte. Tuttavia non si può determinare la fine della fase di partenza lenta e l'inizio del controllo di congestione per tale traccia. La principale ragione è che il mittente TCP non sta inviando i dati abbastanza aggressivamente da pervenire allo stato di congestione. Ispezionando la totalità dei dati *outstanding* si può osservare che l'applicazione al massimo, invia blocchi di dati di 8192 byte. Prima che esso riceva l'acknowledgement per l'intero blocco di questi 8192 byte, l'applicazione non invierà altri dati. Questo indica, prima della fine della fase di partenza lenta, che l'applicazione già ferma temporaneamente la trasmissione.

Type.	No.	Seq	ACKed seq.	Outstanding data
Data	4	1		565
Data	5	566		2025
ACK	6		566	1460
Data	7	2026		2920
Data	8	3486		4380
ACK	9		2026	2920
Data	10	4946		4380
Data	11	6406		5840
ACK	12		3486	4380
Data	13	7866		5527
ACK	14		4096	4917
ACK	15		6006	3007
ACK	16		7866	1147
ACK	17		9013	0
Data	18	9013		1460
Data	19	10473		2920
Data	20	11933		4380
Data	21	13393		5840
Data	22	14853		7300
Data	23	16313		8192
ACK	24		10473	6732
ACK	25		11933	5272
ACK	26		13393	3812
ACK	27		14853	2352

ACK	28		16313	892
ACK	29		17205	0
Data	30	17205		1460
Data	31	18665		2920
Data	32	20125		4380
Data	33	21585		5840
Data	34	23045		7300
Data	35	24505		8192
ACK	36		18665	6732
ACK	37		20125	5272
ACK	38		21585	3812
ACK	39		23045	2352
ACK	40		24505	892
ACK	41		25397	0
Data	42	25397		1460
Data	43	26857		2920
Data	44	28317		4380
Data	45	29777		5840
Data	46	31237		7300
Data	47	32697		8192
ACK	48		26857	
ACK	49		28317	
ACK	50		29777	
ACK	51		31237	
ACK	52		33589	
Data	53	33589		6732
Data	54	35049		5272
Data	55	36509		3812
Data	56	37969		2352
Data	57	39429		892
Data	58	40889		0
ACK	59		35049	6732
ACK	60		37969	3812
ACK	61		40889	892
ACK	62		41781	0
Data	63	41781		1460
Data	64	43241		2920
Data	65	44701		4380
Data	66	46161		5840
Data	67	47621		7300
Data	68	49081		8192
ACK	69		44701	5272
ACK	70		47621	2352
ACK	71		49973	0
Data	72	49973		1460

Data	73	51433		2920
Data	74	52893		4380
Data	75	54353		5840
Data	76	55813		7300
Data	77	57273		8192
ACK	78		52893	5272
ACK	79		55813	2352
ACK	80		58165	0
Data	81	58165		

12. Si commentino i modi in cui i dati misurati differiscono dal comportamento teorico del TCP.

Il comportamento “idealizzato” del TCP assume che i mittenti TCP siano aggressivi nello spedire i dati; perciò, i mittenti TCP dovrebbero seguire l’algoritmo AIMD in modo che quando rilevano la congestione della rete (ossia, perdita di pacchetto), la dimensione della loro finestra di invio dovrebbe diminuire. Nella pratica, il comportamento del TCP dipende in gran parte dall’applicazione. In questo esempio, quando il TCP può spedire i dati, non ci sono dati disponibili per la trasmissione. Nell’applicazione web, alcuni degli oggetti web hanno dimensioni molto piccole. La trasmissione è finita prima della fine della fase di partenza lenta; quindi, la trasmissione di questi piccoli oggetti web risente del lungo e non necessario ritardo a causa della fase di partenza lenta del TCP.

Riferimenti

[1] Lo strato di trasporto (1/2): <http://www.di.uniba.it/%7Elisi/courses/reti/Reti09.pdf>