

# Ethereal Lab: TCP

In this lab, we'll investigate the behavior of TCP in detail. We'll do so by analyzing a trace of the TCP segments sent and received in transferring a 150KB file (containing the text of Lewis Carroll's *Alice's Adventures in Wonderland*) from a computer to a remote server. We'll study TCP's use of sequence and acknowledgement numbers for providing reliable data transfer; we'll see TCP's congestion control algorithm – slow start and congestion avoidance – in action; and we'll look at TCP's receiver-advertised flow control mechanism. We'll also briefly consider TCP connection setup and we'll investigate the performance (throughput and round-trip time) of the TCP connection between the client computer and the server.

Before beginning this lab, you'll probably want to review sections 3.5 and 3.7 in the text.

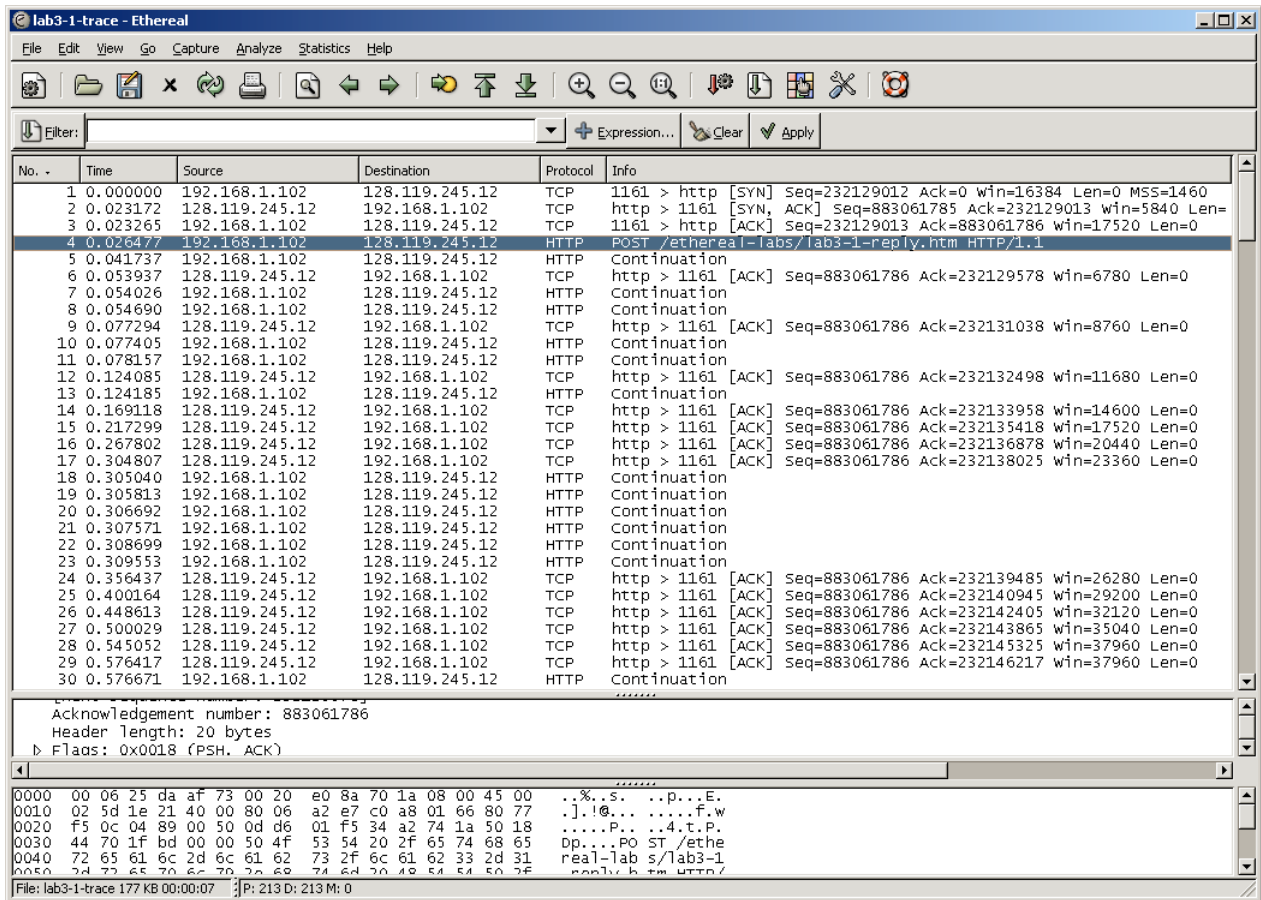
## 1. Capturing a bulk TCP transfer from the client computer to a remote server

To allow you to investigate TCP's behavior, Ethereal was used to obtain a packet trace of the TCP transfer of a file from a client computer to a remote server. The trace was created by accessing a Web page that will allowed the user to enter the name of a file stored on the client computer (which contains the ASCII text of *Alice in Wonderland*), and then transfer the file to a Web server using the HTTP POST method (see section 2.2.3 in the text). The POST method was used rather than the GET method because we'd like to transfer a large amount of data *from* the client computer to another computer. During the transfer, Ethereal was running to obtain the trace of the TCP segments sent and received from the client computer.

The ethereal packet trace was captured using the following steps:

- The user visited the URL <http://gaia.cs.umass.edu/ethereal-labs/TCP-ethereal-file1.html>
- The user entered the name of the file containing the text of *Alice in Wonderland*.
- Before pressing the Upload button, the packet capture with Ethereal was started.
- Then the user pressed the “*Upload alice.txt file*” button to upload the file to the [gaia.cs.umass.edu](http://gaia.cs.umass.edu) server. Once the file had been uploaded, a short congratulations message was displayed in the browser window.
- Ethereal packet capture was then stopped.

- If you load the `ethereal_tcp_trace` file into Ethereal, your Ethereal window should look like the window shown below.



## 2. A first look at the captured trace

Before analyzing the behavior of the TCP connection in detail, let's take a high level view of the trace. First, filter the packets displayed in the Ethereal window by entering "tcp" into the display filter specification window towards the top of the Ethereal window.

What you should see is series of TCP and HTTP messages between the client computer and `gaia.cs.umass.edu`. You should see the initial three-way handshake containing a SYN message. You should see an HTTP POST message and a series of "HTTP Continuation" messages being sent from the client computer to `gaia.cs.umass.edu`. Note that there is no such thing as an HTTP Continuation message – this is Ethereal's way of indicating that there are multiple TCP segments being used to carry a single HTTP message. You should also see TCP ACK segments being returned from `gaia.cs.umass.edu` to the client computer.

Whenever possible, when answering a question you should hand in a printout of the packet(s) within the trace that you used to answer the question asked. Annotate the printout to explain your answer. To print a packet, use *File->Print*, choose *Selected packet only*, choose *Packet summary line*, and select the minimum amount of packet detail that you need to answer the question.

Since this lab is about TCP rather than HTTP, let's change Ethereal's "listing of captured packets" window so that it shows information about the TCP segments containing the HTTP messages, rather than about the HTTP messages. To have Ethereal do this, select *Analyze->Enabled Protocols*. Then uncheck the HTTP box and select *OK*. Also, in this lab we would like to see TCP's sequence numbers (and not the relative sequence numbers that Ethereal may instead display). To see the sequence numbers, go to *Edit>Preferences>Protocols>IP* and uncheck "relative sequence numbers". You should now see an Ethereal window that looks like:

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.102	128.119.245.12	TCP	1161 > http [SYN] Seq=232129012 Ack=0 win=16384 Len=0 MSS=1460
2	0.023172	128.119.245.12	192.168.1.102	TCP	http > 1161 [SYN, ACK] Seq=883061785 Ack=232129013 win=5840 Len=0
3	0.023265	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232129013 Ack=883061786 win=17520 Len=0
4	0.026477	192.168.1.102	128.119.245.12	TCP	1161 > http [PSH, ACK] Seq=232129013 Ack=883061786 win=17520 Len=0
5	0.041737	192.168.1.102	128.119.245.12	TCP	1161 > http [PSH, ACK] Seq=232129578 Ack=883061786 win=17520 Len=0
6	0.053937	128.119.245.12	192.168.1.102	TCP	http > 1161 [ACK] Seq=883061786 Ack=232129578 win=6780 Len=0
7	0.054026	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232131038 Ack=883061786 win=17520 Len=1460
8	0.054690	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232132498 Ack=883061786 win=17520 Len=1460
9	0.077294	128.119.245.12	192.168.1.102	TCP	http > 1161 [ACK] Seq=883061786 Ack=232131038 win=8760 Len=0
10	0.077405	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232133958 Ack=883061786 win=17520 Len=1460
11	0.078157	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232135418 Ack=883061786 win=17520 Len=1460
12	0.124085	128.119.245.12	192.168.1.102	TCP	http > 1161 [ACK] Seq=883061786 Ack=232132498 win=11680 Len=0
13	0.124185	192.168.1.102	128.119.245.12	TCP	1161 > http [PSH, ACK] Seq=232136878 Ack=883061786 win=17520 Len=0
14	0.169118	128.119.245.12	192.168.1.102	TCP	http > 1161 [ACK] Seq=883061786 Ack=232133958 win=14600 Len=0
15	0.217299	128.119.245.12	192.168.1.102	TCP	http > 1161 [ACK] Seq=883061786 Ack=232135418 win=17520 Len=0
16	0.267802	128.119.245.12	192.168.1.102	TCP	http > 1161 [ACK] Seq=883061786 Ack=232136878 win=20440 Len=0
17	0.304807	128.119.245.12	192.168.1.102	TCP	http > 1161 [ACK] Seq=883061786 Ack=232138025 win=23360 Len=0
18	0.305040	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232138025 Ack=883061786 win=17520 Len=1460
19	0.305813	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232139485 Ack=883061786 win=17520 Len=1460
20	0.306692	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232140945 Ack=883061786 win=17520 Len=1460
21	0.307571	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232142405 Ack=883061786 win=17520 Len=1460
22	0.308699	192.168.1.102	128.119.245.12	TCP	1161 > http [ACK] Seq=232143865 Ack=883061786 win=17520 Len=1460
23	0.309553	192.168.1.102	128.119.245.12	TCP	1161 > http [PSH, ACK] Seq=232145325 Ack=883061786 win=17520 Len=0

Frame 7 (1514 bytes on wire (1514 bytes captured))  
 Ethernet II, Src: 00:20:e0:8a:70:1a, Dst: 00:06:25:da:af:73  
 Internet Protocol, Src Addr: 192.168.1.102 (192.168.1.102), Dst Addr: 128.119.245.12 (128.119.245.12)  
 Transmission Control Protocol, Src Port: 1161 (1161), Dst Port: http (80), Seq: 232131038, Ack: 883061786, Len: 1460  
 Source port: 1161 (1161)  
 Destination port: http (80)  
 Sequence number: 232131038  
 [Next sequence number: 232132498]  
 Acknowledgement number: 883061786  
 Header length: 20 bytes  
 Flags: 0x0010 (ACK)  
 Window: 17520  
 Length: 1460  
 Options: 0  
 Raw packet data (hex): 0000 00 06 25 da af 73 00 20 e0 8a 70 1a 08 00 45 00 ...%.s. .p...E.  
 Raw packet data (ASCII): 0010 05 dc 1e 23 40 00 80 06 9f 66 c0 a8 01 66 80 77 ...#...f..f.w  
 Raw packet data (ASCII): 0020 f5 0c 04 89 00 50 0d d6 09 de 34 a2 74 1a 50 10 ....P...4.t.p.  
 Raw packet data (ASCII): 0030 44 70 b9 8e 00 00 0d 0a 0d 0a 57 65 20 61 72 65 dp.....we are  
 Raw packet data (ASCII): 0040 20 6e 6f 77 20 74 72 79 69 6e 67 20 64 6f 20 72 now try ing to r  
 Raw packet data (ASCII): 0050 65 6c 65 61 72 65 20 61 6c 6c 20 6f 75 72 20 62 al...ll our h

This is what we're looking for - a series of TCP segments sent between the client computer and gaia.cs.umass.edu. We will use the packet trace to study TCP behavior in the rest of this lab.

### 3. TCP Basics

Answer the following questions for the TCP segments:

1. What is the IP address and TCP port number used by the client computer (source) to transfer the file to gaia.cs.umass.edu? What is the IP address and port number used by gaia.cs.umass.edu to receive the file?
2. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and gaia.cs.umass.edu? What is it in the segment that identifies the segment as a SYN segment?
3. What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN? What is the value of the ACKnowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?
4. What is the sequence number of the TCP segment containing the HTTP POST command? Note that in order to find the POST command, you'll need to dig into the packet content field at the bottom of the Ethereal window, looking for a segment with a "POST" within its DATA field.
5. Consider the TCP segment containing the HTTP POST as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection (including the segment containing the HTTP POST)? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value (see page 237 in text) after the receipt of each ACK? Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation on page 237 for all subsequent segments.  
*Note:* Ethereal has a nice feature that allows you to plot the RTT for each of the TCP segments sent. Select a TCP segment in the "listing of captured packets" window that is being sent from the client to the gaia.cs.umass.edu server. Then select: *Statistics->TCP Stream Graph->Round Trip Time Graph*.
6. What is the length of each of the first six TCP segments?<sup>1</sup>
7. What is the minimum amount of available buffer space advertised at the receiver for the entire trace? Does the lack of receiver buffer space ever throttle the sender?
8. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?
9. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (see Table 3.2 on page 245 in the text).
10. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

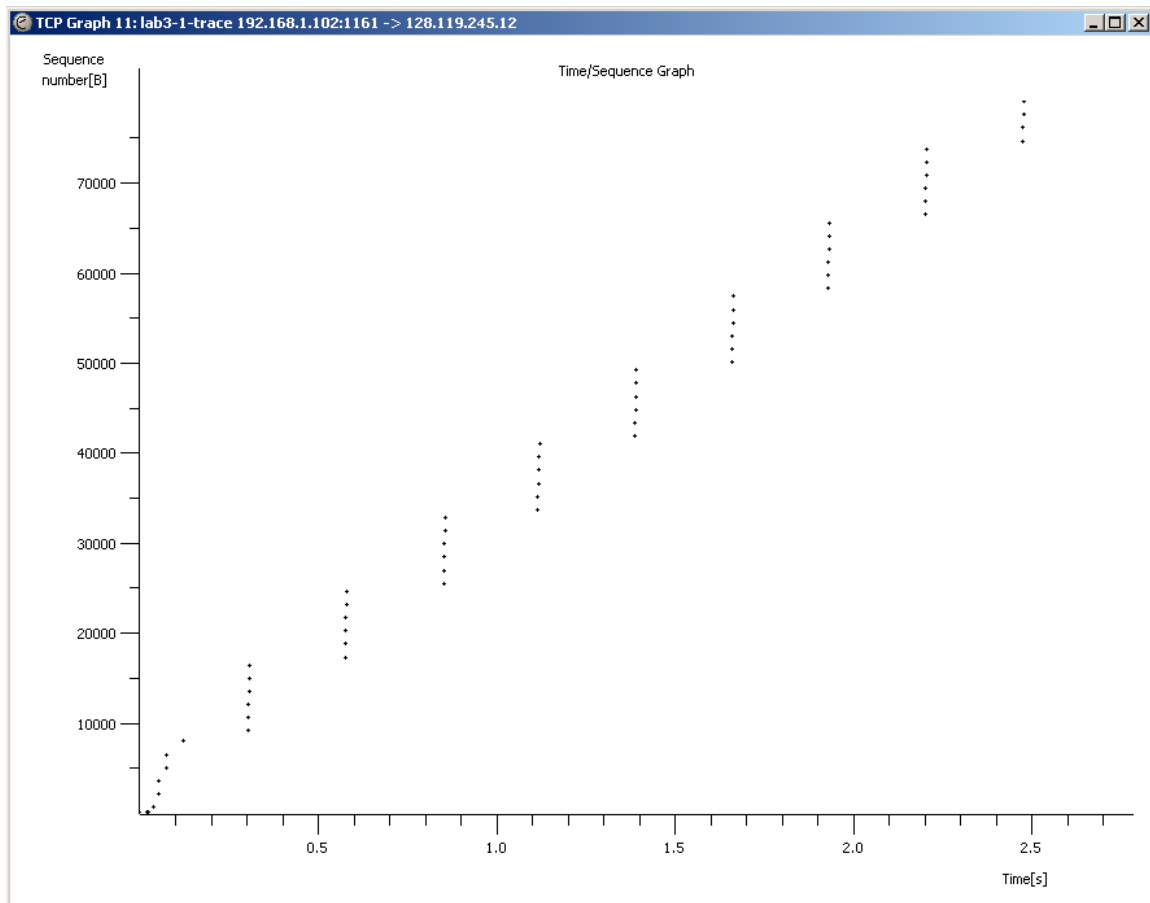
---

<sup>1</sup> The TCP segments in the tcp-ethereal-trace-1 trace file are all less than 1460 bytes. This is because the computer on which the trace was gathered has an Ethernet card that limits the length of the maximum IP packet to 1500 bytes (40 bytes of TCP/IP header data and 1460 bytes of TCP payload). This 1500 byte value is the standard maximum length allowed by Ethernet.

## 4. TCP congestion control in action

Let's now examine the amount of data sent per unit time from the client to the server. Rather than (tediously!) calculating this from the raw data in the Ethereal window, we'll use one of Ethereal's TCP graphing utilities - *Time-Sequence-Graph(Stevens)* - to plot out data.

- Select a TCP segment in the Ethereal's "listing of captured-packets" window. Then select the menu : *Statistics->TCP Stream Graph-> Time-Sequence-Graph(Stevens)*. You should see a plot that looks similar to the following plot:



Here, each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent. Note that a set of dots stacked above each other represents a series of packets that were sent back-to-back by the sender.

Answer the following questions:

11. Use the *Time-Sequence-Graph(Stevens)* plotting tool to view the sequence number versus time plot of segments being sent from the client to the `gaia.cs.umass.edu` server. Can you identify where TCP's slowstart phase begins

and ends, and where congestion avoidance takes over? Note that in this “real-world” trace, not everything is quite as neat and clean as in Figure 3.51 (also note that the y-axis labels for the *Time-Sequence-Graph(Stevens)* plotting tool and Figure 3.51 are different).

12. Comment on ways in which the measured data differs from the idealized behavior of TCP that we’ve studied in the text.