

# COMP 3331/9331: Computer Networks and Applications

## Lab Exercise 3: TCP and UDP (Solutions)

### **AIM**

To investigate the behaviour of TCP and UDP in greater detail.

---

### **EXPERIMENT 1: Understanding TCP Basics**

#### *Tools*

For this experiment, we will use the *Wireshark* packet analyser that we used extensively in the previous lab. Before you begin go to the “Lab Traces” link on the course webpage and download the traces for Lab 3.

#### *Exercise*

Follow the steps described below. You will notice certain questions as you attempt the exercise. Write down the answers for your own reference. The solutions will be put up on the webpage at the end of the week. If you have any questions or are experiencing difficulty with executing the lab please consult your lab instructor.

Step 1: Open an xterm and run Wireshark.

Step 2: Load the trace file *tcp-ethereal-trace-1* by using the *File* pull down menu, choosing *Open* and selecting the appropriate trace file. This file captures the sequence of messages exchanged between a host and a remote server (gai.cs.umass.edu). The host transfers a 150 KB text file, which contains the text of Lewis Carrol’s *Alice’s Adventure in Wonderland* to the server. Note that the file is being transferred from the host to the server.

Step 3: Now filter out all non-TCP packets by typing “tcp” (without quotes) in the filter field towards the top of the Wireshark window. You should see a series of TCP and HTTP messages between the host in MIT and gaia.cs.umass.edu. The first three packets of the trace consist of the initial three-way handshake containing the SYN, SYN ACK and ACK messages. You should see an HTTP POST message and a series of “HTTP Continuation” messages being sent from the host in MIT to gaia.cs.umass.edu. Recall from the first lab exam that there is no such thing as an HTTP Continuation message – this is Wireshark’s way of indicating that there are multiple 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 host in MIT.

**IMPORTANT NOTE:** Do the sequence numbers for the sender and receiver start from zero? The reason for this is that Wireshark by default scales down all real sequence numbers such that the first segment in the trace file always starts from 0. To turn off this feature, you have to click Edit->Preferences>Protocols->TCP and then disable the “Relative Sequence Numbers and Window Scaling” option. Note that the answers in the

solution set will reflect this change. If you conduct the experiment without this change, the sequence numbers that you observe will be different from the ones in the answers.

*Question 1.* What is the IP address and TCP port number used by the client computer (source) that is transferring the file to gaia.cs.umass.edu? (Hint: To answer this question it's probably easiest to select an HTTP message and explore the details of the TCP packet used to carry this HTTP message, using the "details of the selected packet header window")

**Answer:** The IP address for the host (in MIT) is 192.168.1.102 and the port number is 1161.

*Question 2.* What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection?

**Answer:** The IP address of gaia.cs.umass.edu is 128.119.245.12 and the port number used for this connection is 80 (indicating a web server).

Step 5: Since this lab is about TCP rather than HTTP, change Wireshark'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 Wireshark do this, select *Analyze->Enabled Protocols*. Then uncheck the HTTP box and select *OK*.

*Question 3.* 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 in the segment that identifies the segment as a SYN segment?

**Answer:** The sequence number of the SYN segment is 232129012. Clicking on the Flags field in Ethereal reveals the details of all the flag fields. Notice that the SYN flag for this segment has been set to 1, implying that this segment is a SYN segment.

*Question 4.* 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?

**Answer:** The sequence number of the SYNACK segment is 883061785. The acknowledgement field is 232129013. The SYNACK packet is acknowledging the previously received SYN packet with the sequence number 232129012 and since this SYN segment does not contain any data, the TCP receiver on the web server simply increments this value by 1 and copies it to the acknowledgement field. The server is thus indicating to the client that the next segment, which it expects from the client, should contain the sequence number 232129013. On examining the flag fields of this TCP segment, we see that the SYN and ACK flag bits are both set to 1 implying that this is a SYNACK packet.

*Question 5.* What is the sequence number of the ACK segment sent by the client computer in response to the SYNACK? What is the value of the Acknowledgement field in this ACK segment? Does this segment contain any data? What is it in the segment that identifies this segment as a ACK segment?

**Answer:** The sequence number of the ACK segment sent by the client is 232129013. The acknowledgment field contains 883061786 (1 greater than the sequence # of the SYNACK). No, this segment does not contain any data. The ACK flag is set to 1 indicating that this is an ACK segment.

**Question 6.** 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.

**Answer:** The sequence number of the TCP segment containing the POST method is 232129013. Notice that the sequence number is same as the sequence number for the ACK segment that was sent prior to this segment. The reason for this is that the ACK segment contained no data and hence the sequence number was not incremented.

**Question 7.** 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) sent from the client to the web server (Do not consider the ACKs received from the server as part of these six segments)? 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, Section 3.5.3) after the receipt of each ACK? Assume that the value of the *EstimatedRTT* is equal to the measured RTT (*SampleRTT*) for the first segment, and then is computed using the *EstimatedRTT* equation on page 237 for all subsequent segments. Set  $\alpha$  to 0.125.

**Note:** Wireshark 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*.

**Answer:** The details for the first six segments and the corresponding ACKs are provided below along with the calculation of the *EstimatedRTT*:

No.	Sequence #	Length (in bytes)	Time sent	Time ACK received	Sample RTT	Estimated RTT
1	232129013	565	0.026477	0.053937	0.02746	0.02746
2	232129578	1460	0.041737	0.077294	0.035557	0.028472
3	232131038	1460	0.054026	0.124085	0.070059	0.03367
4	232132498	1460	0.054690	0.169118	0.114428	0.043765
5	232133958	1460	0.077405	0.217299	0.139894	0.055781
6	232135418	1460	0.078157	0.267802	0.189645	0.072514

Be careful when you note down the times. Note that the ACK always contains the next expected sequence number in the acknowledgement field.

**Question 8.** What is the length of each of the first six TCP segments?

**Answer:** See the above table for the TCP segment lengths. Note that the length in the table indicates the payload contained in the TCP header. It does not include the 20 byte TCP header.

*Question 9.* 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?

**Answer:** To answer this question examine all the ACK (and SYNACK) packets sent by the server to the client. The minimum advertised window is 5840 and this is advertised in the SYNACK segment. The receiver window does not seem to throttle the sender at all. Even when the advertised receiver window is at its lowest value (5840: equivalent to 4 MSS packets of 1460 bytes), the sender seems to be constrained by congestion window as opposed to the flow control window. Notice that in the later part of the trace, when the congestion window has grown to a reasonable size, the receiver advertised window is very large (mostly 62780 bytes: 43 MSS segments). Hence the lack of receiver buffer space does not seem to be an issue with this connection.

*Question 10.* Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

**Answer:** There are no retransmitted segments in the trace file. If at all a packet is retransmitted there would be a repeat entry for that segment which is retransmitted with the same sequence number. However there is no such occurrence in the entire trace file.

*Question 11.* 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, Section 3.5.4).

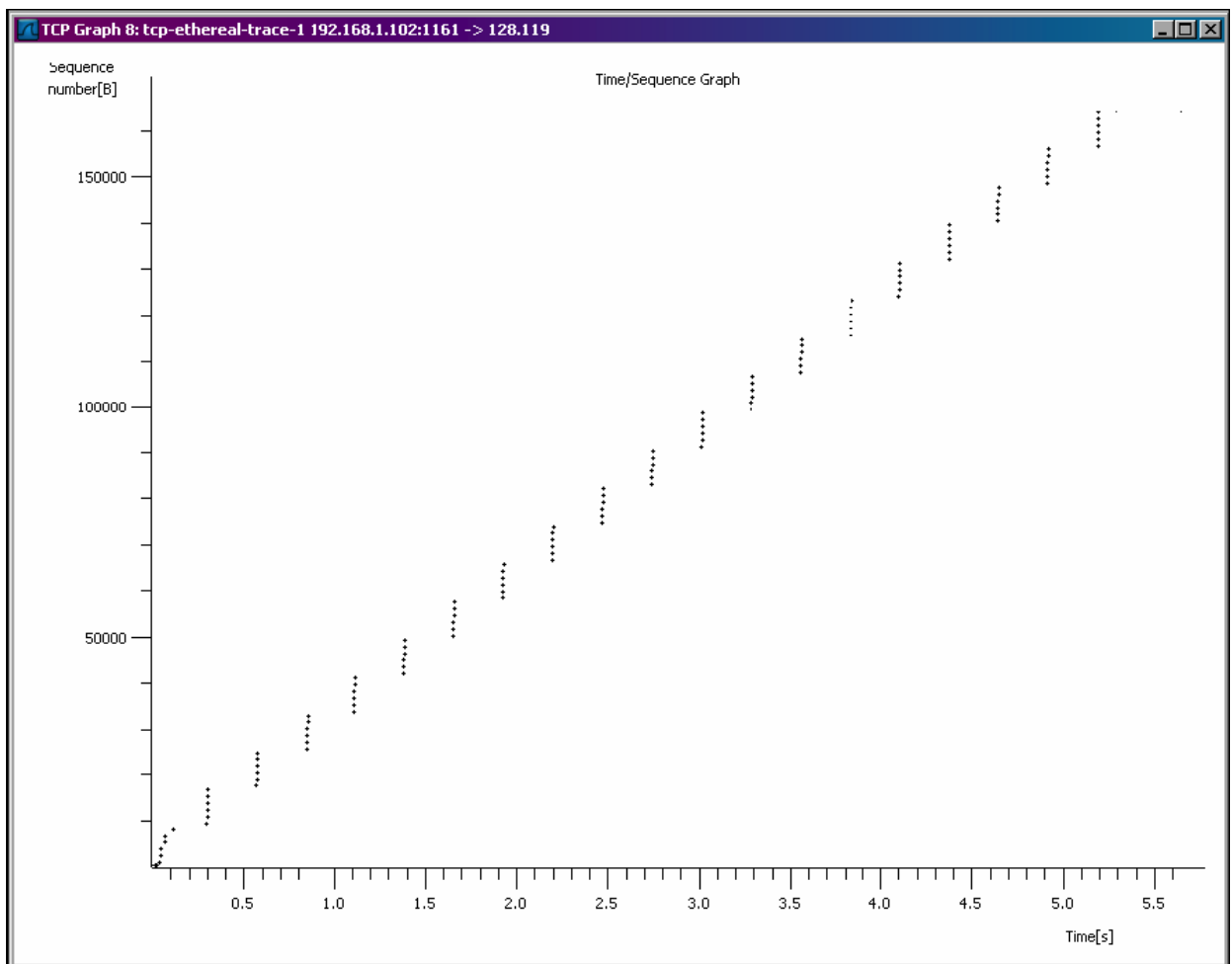
**Answer:** In the early part of the trace file (probably slow-start phase), we notice that each packet is individually being ACKed by the receiver. We observe behaviour where the sender transmits a burst of packets and the receiver then sends back ACKs for each of them. However later on in the trace, in particular at segment # 60, you will notice that the ACK with the acknowledgement field as 232166981 is actually acknowledging two segments with sequence #232164061 and # 232165521. From this point on there are several instances when the receiver sends an ACK for every other received segment. The receiver is typically sending a cumulative ACK for two TCP segments that it receives. As explained earlier this is due to the fact that TCP uses Delayed ACKs where the receiver waits for up to 500 msec for the arrival of another in-order segment and then sends a cumulative ACK for both of the received segments. (Refer to Table 3.2 on page 245 in Section 3.5.4 of the text).

*Question 12.* What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

**Answer:** The total size of the data file being transmitted is 150KB. The total duration required for transmitting this file (neglecting the three-way handshake process) is 5.624664 seconds. So the average throughput for the TCP connection is  $150 \times 8 / 5.624664 = 213.346$  Kbits/second.

An alternate way to compute the throughput could also involve looking at the amount of data that the TCP sender sends back-to-back as part of one window and dividing this by the total time since the transmission of the first packet in this window till the last ACK (for the last packet in this window) is received. This throughput will be slightly greater than that calculated above since it does not take into account the slow start phase.

Step 6: We shall now examine the amount of data sent per unit time from the client to the server. Rather than calculating this from the raw data in the Wireshark window, we shall make use of Wireshark's TCP graphing utilities to plot our data. To do this select from the menu: *Statistics->TCP Stream Graph->Time-Sequence-Graph(Stevens)*. You should see a plot that looks similar to the picture below.



Here each dot represents a TCP segment sent, plotting the sequence number of the segment versus the time at which it was sent (the y-axis is the sequence number and the x-axis is time).

Question 13. Using this plot can you identify where TCP's slowstart phase begins and ends and where congestion avoidance takes over?

**Answer:** Observing the plot it appears that the slow-start phase only lasts for the first 0.1 second. Following this it appears that the TCP session is always in congestion avoidance. However, we do not observe the expected linear increase behaviour, i.e. the

TCP transmit window does not grow linearly during this phase. In fact, it appears that the sender transmits packets in batches of 5. This does not seem to be caused by flow control since the receiver advertised window is significantly larger than 5 packets. The reason for this behaviour is not obvious; perhaps the HTTP server has enforced some sort of rate-limit.

---

## EXPERIMENT 2: Understanding UDP Basics

### *Tools*

For this experiment, we will use the *Wireshark* packet analyser that we used extensively in the previous lab. Before you begin go to the “Lab Traces” link on the course webpage and download the traces for Lab 3.

### *Exercise*

Follow the steps described below. You will notice certain questions as you attempt the exercise. Write down the answers for your own reference. The solutions will be put up on the webpage at the end of the week. If you have any questions or are experiencing difficulty with executing the lab please consult your lab instructor.

Step 1: Open an xterm and run Wireshark.

Step 2: Load the trace file *dns-ethereal-trace-2* by using the *File* pull down menu, choosing *Open* and selecting the appropriate trace file. Recall that we had used this trace file in Lab Exercise 2 for studying the behaviour of DNS. This file captures the sequence of messages exchanged between a host and its default DNS server while using the *nslookup* utility. Recall that DNS uses UDP as the transport layer protocol.

Step 3: Now filter out all non-UDP packets by typing “udp” (without quotes) in the filter field towards the top of the Wireshark window.

Question 1. Select the first packet in the trace. Determine, how many fields there are in the UDP header (Do not look in the textbook! Answer these questions directly from what you observe in the packet trace). Name these fields.

**Answer:** The UDP header consists of 4 fields: Source Port, Destination Port, Length and Checksum.

Question 2. From the packet content field (click on any header and observe the display at the bottom of the Wireshark window), determine the length (in bytes) of each of the UDP header fields.

**Answer:** The length in bytes of each of the header fields is as follows: (i) Source Port: 2 bytes, (ii) Destination Port: 2 bytes, (iii) Length: 2 bytes and (iv) Checksum: 2 bytes.

Question 3. The value in the Length field is the length of what? Verify your claim using the selected packet.

**Answer:** The value of the Length field (28 bytes) is the total length of the UDP segment, i.e. including the header and payload. Each of the 4 header fields is 2 bytes long as discussed above. Further, the payload of the UDP segment is 20 bytes. Thus,  $20 + 4(2) = 28$  bytes.

Question 4. What is the maximum number of bytes that can be included in a UDP payload?

**Answer:** The maximum size of a UDP segment (including headers) is 65535 bytes ( $2^{16} - 1$  bytes). Thus the maximum size of the UDP payload can be 65527 bytes. Of course in a real network, the MTU of the underlying network will limit the maximum size to a much smaller value.

Question 5. What is the largest possible source port number?

**Answer:** The largest possible source port number is 65535 ( $2^{16} - 1$ ).

Question 6. Examine the first two DNS packets in the trace (neglect the first HSRP packet). Describe the relationship between the port numbers in the two packets.

**Answer:** The source and destination port numbers for the first UDP packet in the trace are 3740 and 53 (used for DNS) respectively. The second UDP packet has the source port number as 53 and the destination port number as 3740, which is reverse of the first packet. This indicates that the two packets are part of a query-response exchange, which is indeed the case – the first packet is a DNS query and the second packet is a DNS response.

**END OF LAB**