Lab Exercise 4: Exploring TCP

Objectives:

- gain insights into the operation of TCP.
- get familiar with the ns-2 simulator (as preparation for the next two labs)

Prerequisites and Links:

- Week 4, 5 & 6 Lectures
- Relevant Parts of Chapter 3 of the textbook
- Introduction to Tools of the Trade
- Basic understanding of Linux. A good resource is here but there are several other resources online:
- tcp-ethereal-trace-1
- Overview slides
- simpleSim.tcl
- Explanation of script simpleSim.tcl

Marks: **10 marks.**

- Please attend the lab in your allocated lab time slot.
- We expect the students to go through as much of the lab exercises as they can at home and come to the lab for clarifying any doubts in procedure/specifications.

Deadline:

23:59:59 Sunday 31st March. You can submit as many times as you wish before the deadline. A later submission will override the earlier submission, so make sure you submit the correct file. Do not leave until the last moment to submit, as there may be technical, or communications error and you will not have time to rectify it.

Late Submission Penalty:

Late penalty will be applied as follows:

- 1 day after deadline: 20% reduction
- 2 days after deadline: 40% reduction
- 3 days after deadline: 60% reduction
- 4 or more days late: NOT accepted

Note that the above penalty is applied to your final mark. For example, if you submit your lab work 2 days late and your score on the lab is 8, then your final mark will be 8 - 3.2 (40% penalty) = 4.8.

Submission Instructions:

Submit a PDF document **Lab4.pdf** with answers to all questions for Exercise 1 & 2 only. Submit the pdf directly.

Original Work Only:

You are strongly encouraged to discuss the questions with other students in your lab. However, each student must submit his or her own work. You may need to refer to the material indicated above (particularly Tools of the Trade document) and also conduct your own research to answer the questions.

Exercise 1: Understanding TCP using Wireshark

For this particular experiment download the trace file: tcp-ethereal-trace-1 .

The following indicate the steps for this experiment:

Step 1: Start Wireshark by typing *wireshark* at the command prompt.

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 (gaia.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 using a HTTP POST message.

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 segments between the host in MIT and gaia.cs.umass.edu. The first three segments 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 in the 4th segment of the trace being sent from the host in MIT to gaia.cs.umass.edu (check the contents of the payload of this segment). You should observe that the text file is transmitted as multiple TCP segments (i.e. a single POST message has been split into several TCP segments) from the client to the server (gaia.cs.umass.edu). You should also see several TCP ACK segments been returned in the reverse direction.

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 (or Wireshark->Preferences->Protocols->TCP) and then disable the "Relative Sequence Numbers" 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. Also, set the time shown in the 2nd column as the "Seconds since beginning of capture" under view->Time display format.

Question 1. What is the IP address of gaia.cs.umass.edu? On what port number is it sending and receiving TCP segments for this connection? 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?

Question 2. 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.

Question 3. 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 relevant parts of Section 3.5 or lecture slides) after the receipt of each ACK? Assume that the initial value of *EstimatedRTT* is equal to the measured RTT (*SampleRTT*) for the first segment, and then is computed using the *EstimatedRTT* equation 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*. However, do not use this graph to answer the above question.

Question 4. What is the length of each of the first six TCP segments?

Question 5. 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?

Question 6. Are there any retransmitted segments in the trace file? What did you check for (in the trace) in order to answer this question?

Question 7. How much data does the receiver typically acknowledge in an ACK? Can you identify cases where the receiver is ACKing every other received segment (recall the discussion about delayed acks from the lecture notes or Section 3.5 of the text).

Question 8. What is the throughput (bytes transferred per unit time) for the TCP connection? Explain how you calculated this value.

Exercise 2: TCP Connection Management

Consider the following TCP transaction between a client (10.9.16.201) and a server (10.99.6.175).

No	Source IP	Destination IP	Protocol	Info
295	10.9.16.201	10.99.6.175	TCP	50045 > 5000 [SYN] Seq=2818463618 win=8192 MSS=1460
296	10.99.6.175	10.9.16.201	TCP	5000 > 50045 [SYN, ACK] Seq=1247095790 Ack=2818463619 win=262144 MSS=1460
297	10.9.16.201	10.99.6.175	TCP	50045 > 5000 [ACK] Seq=2818463619 Ack=1247095791 win=65535
298	10.9.16.201	10.99.6.175	TCP	50045 > 5000 [PSH, ACK] Seq=2818463619 Ack=1247095791 win=65535
301	10.99.6.175	10.9.16.201	TCP	5000 > 50045 [ACK] Seq=1247095791 Ack=2818463652 win=262096
302	10.99.6.175	10.9.16.201	TCP	5000 > 50045 [PSH, ACK] Seq=1247095791 Ack=2818463652 win=262144
303	10.9.16.201	10.99.6.175	TCP	50045 > 5000 [ACK] Seq=2818463652 Ack=1247095831 win=65535
304	10.9.16.201	10.99.6.175	TCP	50045 > 5000 [FIN, ACK] Seq=2818463652 Ack=1247095831 win=65535
305	10.99.6.175	10.9.16.201	TCP	5000 > 50045 [FIN, ACK] Seq=1247095831 Ack=2818463652 win=262144
306	10.9.16.201	10.99.6.175	TCP	50045 > 5000 [ACK] Seq=2818463652 Ack=1247095832 win=65535
308	10.99.6.175	10.9.16.201	TCP	5000 > 50045 [ACK] Seq=1247095831 Ack=2818463653 win=262144

Answer the following questions:

Question 1. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection between the client computer and server?

Question 2. What is the sequence number of the SYNACK segment sent by the server to the client computer in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How did the server determine that value?

Question 3. 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 Acknowledgment field in this ACK segment? Does this segment contain any data?

Question 4. Who has done the active close? client or the server? how you have determined this? What type of closure has been performed? 3 Segment (FIN/FINACK/ACK), 4 Segment (FIN/ACK/FIN/ACK) or Simultaneous close?

Question 5. How many data bytes have been transferred from the client to the server and from the server to the client during the whole duration of the connection? What relationship does this have with the Initial Sequence Number and the final ACK received from the other side?

Exercise 3: Getting familiarised with ns-2 simulator (not marked, do not include in your report)

IMPORTANT NOTE: ns-2 and Nam are installed on all CSE lab machines. We do not recommend that you install ns-2 on your personal machines. This is not as straightforward as installing wireshark. Moreover, the provided scripts have ONLY been tested on CSE machines and may not work on other operating systems. We cannot offer support for running ns-2 natively on your local machines. It is possible to run ns-2 remotely via ssh.

ns-2 is a powerful simulator that provides substantial support for simulating TCP, routing and multicast protocols, among other things. It is capable of simulating the conditions that occur in wired or wireless networks. It is widely used in the research community and also in industry.

The simulator is written in C++. However, it uses OTcl as its command and configuration interface. In our lab exercises, we will use scripts written in OTcl. You will not be required to write any C++ code for any of the lab exercises. You will also not be required to write OTcl scripts from scratch. All scripts will be provided and at most ask you change a line or two in the scripts with proper instructions. To complete the lab exercises, you can safely assume ns-2 to be a black box. However, for those who are interested in finding out a bit more about ns-2, please refer to the following overview slides which offers a good introduction.

We will also use a network animator tool: Nam . This allows us to visualise the topology and the transmission of packets during the experiments.

Illustrative Example: 2 nodes communicating directly over UDP

We will use this example to get acquainted with ns-2. The OTcl script for the first experiment is simpleSim.tcl . It creates two nodes and simulates the sending of data packets from one node to the other over UDP. The full explanation of the script can be found here. **We strongly recommend** you read through it as it will be helpful for you for future exercises. It doesn't go over all details but rather gives you an overview of how to setup a topology, create traffic and run an experiment.

You can run the script by typing the following command:

```
$ ns simpleSim.tcl
```

This will start the nam tool and you will see the nam window . When you click on the 'play' button in the window, you will see that after 0.5 seconds of simulated time, node 0 starts sending data packets to node 1. The transmission stops at t = 1.5 seconds and the simulation stops at time t = 2 seconds. You might want to slow nam down by using the 'Step' slider at the top right.

Once the experiment has concluded close nam. You will find an output trace file named 'out.tr' in your current directory. The format is as follows:

```
event time src dest PktType PktSize Flags Fid SrcSddr DestAddr SeqNum PktId
```

where:

- event:
  - r: receive at dest
  - +: enqueue
  - -: dequeue
  - d: drop

- ScrAddr and DestAddr :
  - node.port (e.g., 0.1 means node 0 , port 1)

Example trace:

```
r 0.519 0 1 cbr 500 ------- 0 0.0 1.0 1 1

+ 0.52  0 1 cbr 500 ------- 0 0.0 1.0 4 4

- 0.52  0 1 cbr 500 ------- 0 0.0 1.0 4 4
```

You may wish to examine out.tr to get a better understanding of what happens in the experiment.

We will use ns-2 in the next lab to run some experiments with TCP flows.