

Lab 4

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 The Script 'simpleSim.tcl'](#)

Marks: 4 marks

Questions to be marked: 1) Exercise 1. Questions 4, 2) Exercise 1. Questions 9, 3) Exercise 1. Questions 11

- Each lab comprises of a number of exercises. Not all the exercises for each lab are marked. Only those marked with (*) and written in bold will be marked.
- We expect the students to go through as much of the lab exercises as they can at home and come to the lab ready.
- Please attend your allocated lab and show/explain the answers of the marked exercises to your tutor.
- If lab exercise involves diagrams or plots, you require to show them to the tutor as well.
- Please make sure you **sign the marking form** once the tutor marked your lab. Signing this form implies that you agreed on the mark you received.
- There are 7 labs during this course. For each student, the 5 best performing labs will contribute to your final lab mark.

Exercise 1: Understanding TCP using Wireshark

For this particular experiment download the second 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 Carroll'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.

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?

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?

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?

***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? (1 marks, 0.25 each part)**

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?

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.

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 relevant parts of Section 3.5 or Week 6 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 8. What is the length of each of the first six TCP segments?

(*) 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? (1.5 marks)

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?

(*) 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 (recall the discussion about delayed acks from the Week 6 lecture notes or Section 3.5 of the text). (1.5 marks)

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

Exercise 2: Getting familiarised with ns-2 simulator

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 or vlab.

[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 [Explanation Of The Script 'simpleSim.tcl'](#). **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 SrcSaddr DestAddr  
SeqNum PktId
```

where:

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