TÉCNICO LISBOA

RC 17/18	LAB ASSIGNMENT	Number:	6
Wireshark Lab: Protocol Analysis		Issue Date:	23 Oct 2017
TCP and UDP		Due Date:	03 Nov 2017

Preliminary Notes

The instructions in this document are applicable to Computers at the IST Labs, which are already configured with the required tools. Nevertheless, one nice feature of the software stack we are going to use is that it is portable to many platforms including **YOUR OWN** personal computers. The instructions are suitable for machines running the following Operating Systems:

- Microsoft Windows from version 7 up to 10
- Apple macOS from versions 10.8 'Mountain Lion' up to 11.13 'High Sierra'
- Debian-based Linux, such as Ubuntu (recommended) from versions 12.04 'Precise' to 16.04 'Xenial Xerus'.

Note: Avoid copying text strings from the command line examples or configurations in this document, as pasting them in your system or files may introduce/modify some characters, leading to errors or inadequate results.

1 Intro

In this lab, we'll take a quick look at the UDP transport protocol. UDP is a streamlined, no-frills protocol. Because UDP is simple, we'll be able to cover it pretty quickly in this lab. We'll also investigate the behavior of the celebrated TCP protocol in detail. We'll do so by analyzing a trace of the TCP segments sent and received in transferring a 150KB file from a client to your server. We'll study TCP's use of sequence and acknowledgement numbers for providing reliable data transfer; we'll see the mechanism of fast retransmission. We'll also briefly consider TCP connection setup and we'll investigate the performance (throughput and round-trip time) of the TCP connection between your computer and the server.

2 Experimental Environment

Run up your C.O.R.E. machine and access to the GUI by issuing:

```
:~$ vagrant ssh -c core-gui -- -X
```

In C.O.R.E. we need to create a simple network between two machines and start your simulation. This simple network will consist of one PC and one Server connected by a router, as illustrated in Figure 1. In order to create you network use the tools available at the left toolbar.

RC 17/18	LAB ASSIGNMENT	Number:	6
Wireshark Lab: Protocol Analysis		Issue Date:	23 Oct 2017
TCP and UDP		Due Date:	03 Nov 2017

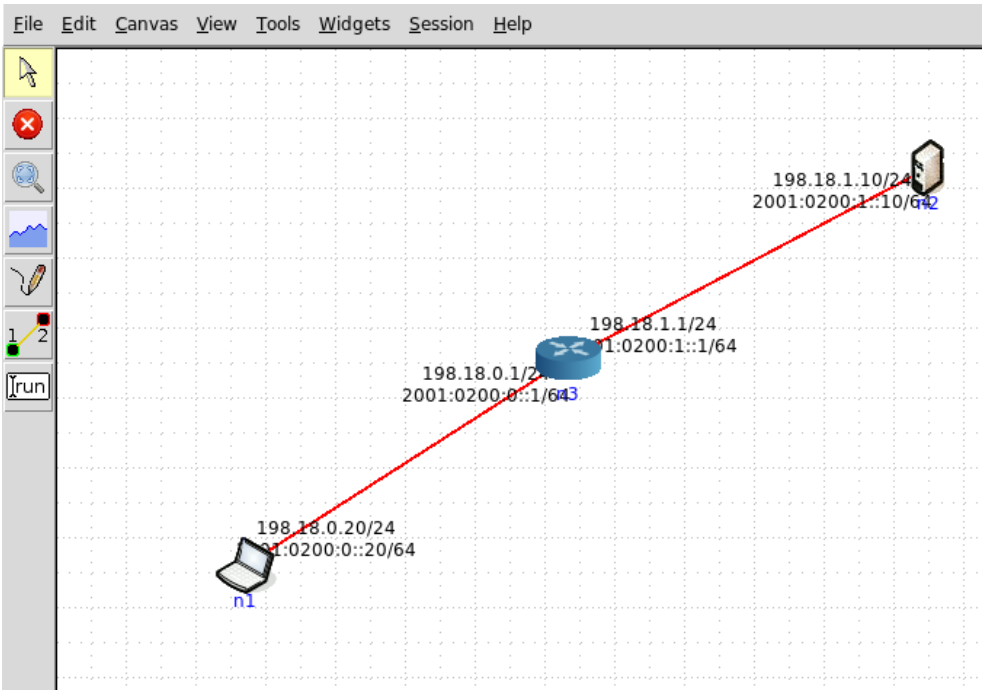

Figure 1: Simple network created on C.O.R.E.

In order to analyze the packets exchanged between, we will use the Wireshark. Open one Wireshark window in one of the end systems:

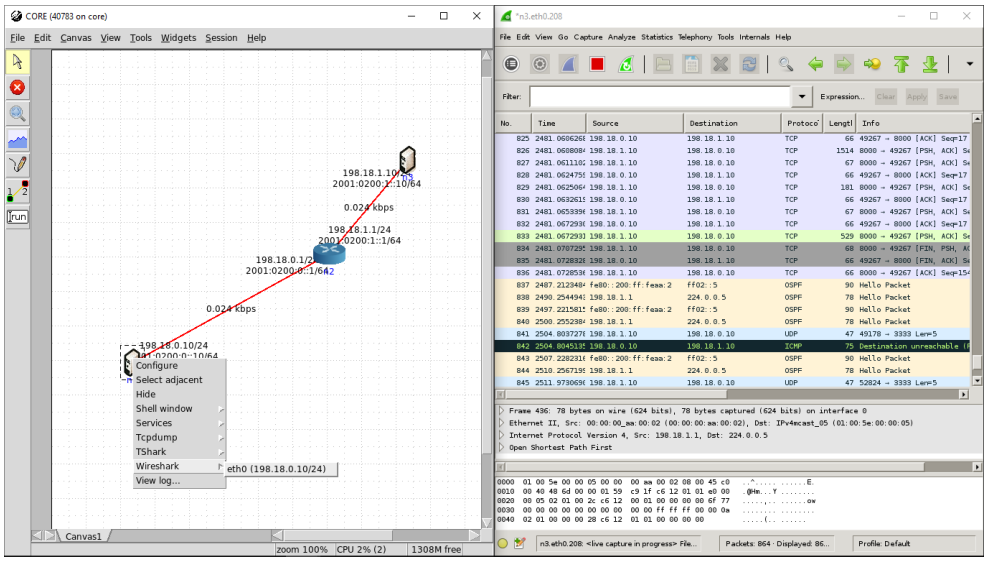

Figure 2: Wireshark window example

In order to have a realistic you should double click in the connection and add some delay and lost probability.

You should also have one 150KB file available inside the machine. TO do that in your shared directory.

2.1 UDP Analysis

In order to analyze one UDP packet we will use netcat to transfer one message. Open two bash windows, one in each one of the end systems.

On receiver we will wait at port 3333:

```
root@n3:~$ nc -ul 3333 >> t.n
```

0n sender we will send the message ”hello”:

```
root@n1:~$ echo -n "hello" | nc -u 198.18.1.10 3333
```

We should get one UDP packet in our wireshark capture. You can use the filter in order to isolate UDP packets, by filling ”udp”:

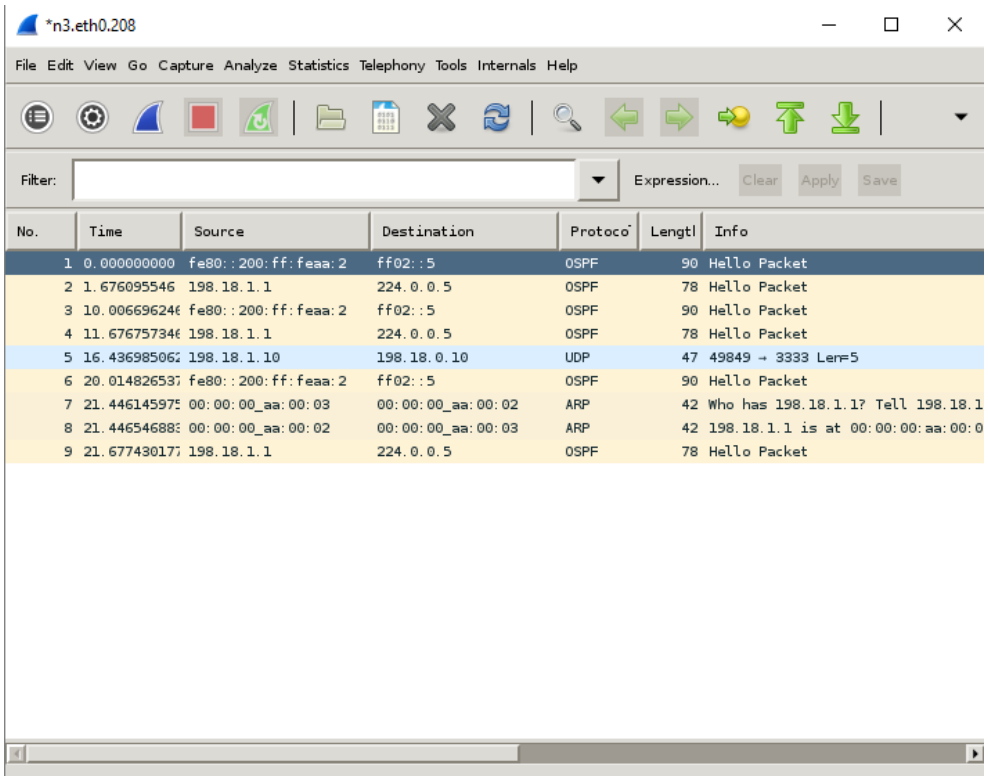

Figure 3: Wireshark UDP example

To print a packet, use File-¿Print, choose Selected packet only, choose Packet summary line, and select the amount of packet detail.

Questions:

1. From this packet, determine how many fields there are in the UDP header and name these fields.
2. By consulting the displayed information in Wireshark's packet content field for this packet, determine the length (in bytes) of each of the UDP header fields.
3. The value in the Length field is the length of what?
4. What is the maximum number of bytes that can be included in a UDP data?
5. What is the largest possible source port number?

2.2 TCP Basics Analysis

In order to analyze some TCP packets we will use the web Server that you implemented last class to get one file. Open two bash windows, one in each one of the end systems.

On our server, we will wait for request at port 8000:

```
root:n1~$ cd /home/vagrant/web_shared
root:n1~$ python webServer.py
Ready to serve in port 8000...
```

0n client we will use telnet to get our file:

```
root:n3~$ telnet 198.18.1.10 8000
Trying 198.18.1.10 ...
Connected to 198.18.1.10.
Escape character is '^]'.
GET / HTTP/1.1
```

In our Wireshark we will have a lot of TCP packets. You can use the filter in order to isolate TCP packets, by filling "tcp":

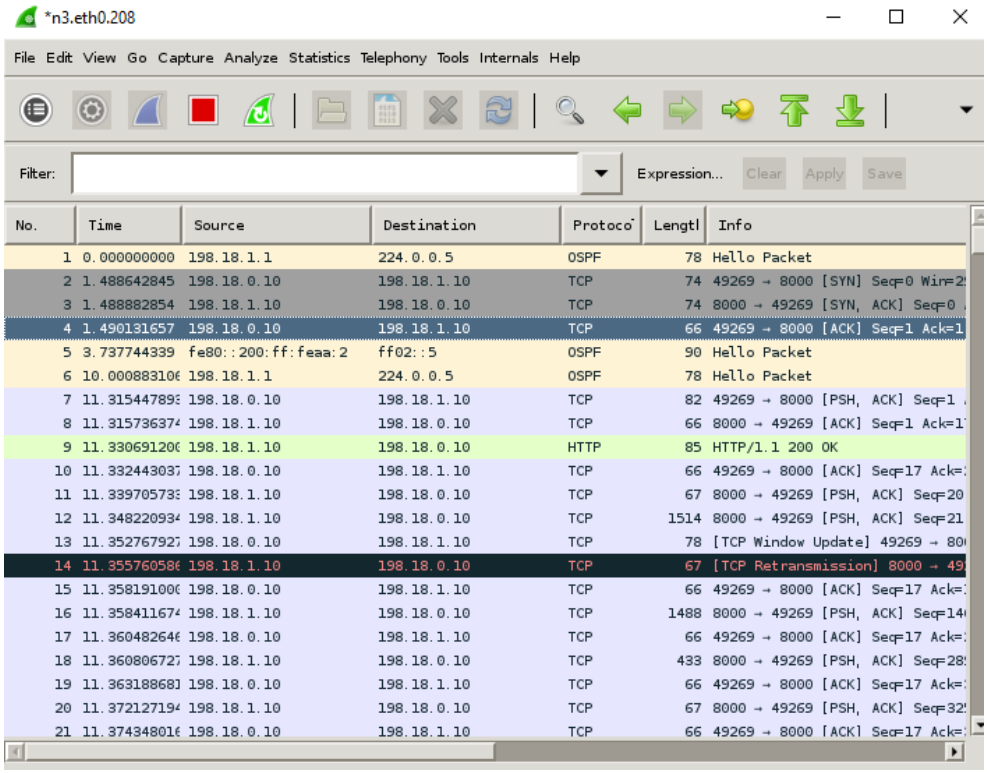

Figure 4: Wireshark TCP example

Questions

1. What is the sequence number of the TCP SYN segment that is used to initiate the TCP connection? What is it in the segment that identifies the segment as a SYN segment?

2. What is the sequence number of the SYNACK segment sent in reply to the SYN? What is the value of the Acknowledgement field in the SYNACK segment? How this value was determined? What is it in the segment that identifies the segment as a SYNACK segment?

3. Consider the TCP segment not SYN as the first segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection? 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 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.

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

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

3 Finishing your Experiments

In order to stop the Virtual Machines and to verify the global state of all active Vagrant environments on the system, we can issue the following commands:

Confirm that the statuses of the VMs is 'powered off'.

In Lab computers, also do the following (answering "y"), in order to ensure the system is cleaned from your experiments. Note however that the configuration files remain stored in your private area in AFS:

```
:~$ vagrant destroy
default: Are you sure you want to destroy the 'default' VM? [y/N]
==> default: Destroying VM and associated drives...
:~$ vagrant global-status
```

Confirm that there are no VMs listed.