TÉCNICO LISBOA

Computer Networks — 2021/22		Assignment:	Lab 6
TCP protocol, sockets & congestion control		Issued:	2021-12-15
Transmission Control Protocol (TCP)		Submission Due:	2021-12-19
Authors:	João Romeiras Amado Prof. Luis D. Pedrosa	Comments Due:	2021-12-23
		Version:	1.0

Submission note. You will need to later submit all the answers to this lab's questions in Moodle. You should create a PDF with all the answers, and submit it into Moodle as an attachment to the submission form.

1 Goals

- Implement a simple TCP client/server communication using sockets.
- Understand the TCP protocol handshake mechanism and explore TCP packet headers.
- Identify instances of TCP congestion control.

2 Exercises

Reminder. Execute the command `git pull origin master` in `/home/rc/lab-files` to get the most up to date files.

2.1 TCP Sockets

TCP is a transport layer protocol that provides a reliable streaming service between two end-points. It provides effective reliability guarantees by detecting lost or corrupt data and retransmitting and reconstructing the data stream.

In this section, you will explore TCP connections through sockets, using a python-based server and client. The provided file `server.py` is already implemented, while the `client.py` file is incomplete, with only a skeleton implementation (highlighted with TO-DO comments). Once the connection between the client and server is established, they can exchange simple text messages with up to 1024 bytes. To close the connection, type in `end` on the client.

The first step is to open the `client.py` file in your prefered text editor and complete its implementation, following both the instructions in the comments and also considering the complete `server.py` code.

Q1. Finish the `client.py` implementation, following the instructions in the comments and also considering the complete `server.py` code. In your answer, show the complete `client.py` implementation using a screenshot.

After completing the implementation, to test the client and server, start the CORE emulator and open the `tcp1.mn` topology file.

After activating the network, open a terminal on `n2` and start `server.py`:

```
python3 /home/rc/lab-files/lab-tcp/server.py
```

Computer Networks — 2021/22		Assignment:	Lab 6
TCP protocol, sockets & congestion control		**Issued:**	2021-12-15
Transmission Control Protocol (TCP)		**Submission Due:**	2021-12-19
Authors:	João Romeiras Amado Prof. Luis D. Pedrosa	**Comments Due:** **Version:**	2021-12-23 1.0

On `n3`, start `client.py`:

```
python3 /home/rc/lab-files/lab-tcp/client.py
```

Test the program and confirm that your socket implementation is correct. Afterwards, retest the connection while observing the traffic in wireshark.

Q2. Show, using a message sequence diagram (e.g. from draw.io), the negotiation steps required to establish a TCP connection on this particular network, detailing the IP addresses, ports, and relevant protocol information such as flags.

Q3. The server implementation allows two `client.py` instances to estabilish a connection simultaneously. Test this by connecting both clients to port 50001 on the server. How does the server distinguish between the two client connections?

2.2 TCP Header Analysis

Every TCP packet segment consists of a header followed by an optional data portion. On this section, the objective is to analyze the TCP protocol header and understand how its various fields are modified during a network connection.

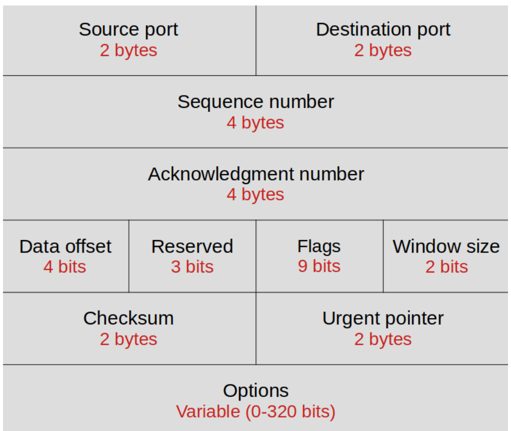

Figure 1: TCP Header

Open the trace file `tcp2.pcap` in Wireshark. Analyze the various connections in the trace, considering in particular the TCP protocol section in the packet details.

Q4. What is the procedure to gracefully terminate a TCP connection between two endpoints? Can you identify any such instance in this trace?

Q5. Observe that in the beginning of the trace, the connection attempts result in an error. Describe the error and identify a possible reason for its occurence.

> **Hint:** Analyze the TCP flags for this particular connection and compare them with a successful TCP connection attempt.

Throughout the trace, there are several packets identified with duplicate acknowledgements (DupACK).

Q6. What can cause these DupAcks and how should the sender react to them?

2.3 Congestion Control

In the TCP protocol, congestion control provides the ability to limit the sending rate in response to packet congestion in the network, while aiming to achieve an overall high throughput rate. TCP's approach to congestion control is multifold, happening across the various phases of a transmission.

In this section, you will use iperf to simulate a TCP connection using the **Reno congestion control algorithm**.

Open the following topology `tcp3.imn` in CORE. On n2, run the following command:

```
iperf3 -s
```

Open wireshark on `n3` and start a capture with the `tcp` filter.

On `n3`, run the following command, which will send TCP packets over 5 flows using the Reno congestion control algorithm. It will run for 60 seconds. Afterwards, stop the wireshark capture.

```
iperf3 -c 10.0.0.10 -B 10.0.1.10 --cport 50000 -P 5 -t 60 -C reno
```

On Wireshark, go to *Statistics* -> *TCP Stream Graphs* -> *Time Sequence (tcptrace)*. The tcptrace graph is a time-sequence graph that shows a TCP stream, representing a single direction at a time (i.e., sender->receiver or receiver->sender). On the graph window options there is a selection button that allows you to quickly switch between the various streams in the trace. Additionally, you can switch the selected direction for a given stream using the *Switch Direction* option.

Analyze the graphs, selecting streams 1-5, with the view corresponding to **10.0.1.10 as the sender**.

Q7. Take a screenshot of one of the stream graphs. Annotate it in your report to highlight an instance of the slow start phase and congestion avoidance (AIMD), highlighting the transition moment.

Fast retransmission, a property of the TCP Reno congestion control algorithm, is initiated when a sender receives a 3rd duplicate ACK, after which it assumes that the corresponding packet is lost, and immediately retransmits it, without waiting for the expiration of the retransmission timer.

Observe the trace packets and the graph and look for retransmission and fast retransmission occurences.

Hint. You can use the Wireshark filters `tcp.analysis.retransmission` and `tcp.analysis.fast_retransmission`.

Q8. Considering the subsequent phase after a congestion event, what is the main difference between performing a fast retransmission, instead of a regular retransmission (after a timeout), in terms of congestion control?