TÉCNICO LISBOA

Computer Networks — 2023/24, Q2		**Assignment:**	Lab 6
Transmission Control Protocol (TCP)		**Version:**	1.0
Issued:	2023-11-27	**Submission Due:**	2023-12-03

Submission note. You will need to submit all the answers to this lab's questions in Moodle, under "Lab 6: TCP".

1 Goals

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

2 Exercises

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. For each TO-DO comment, you must write a **one-line command** to perform the intended procedure.

Q1. Finish the `client.py` implementation, following the instructions in the comments and also considering the complete `server.py` code.

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
```

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, 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. You can test this by connecting two 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.

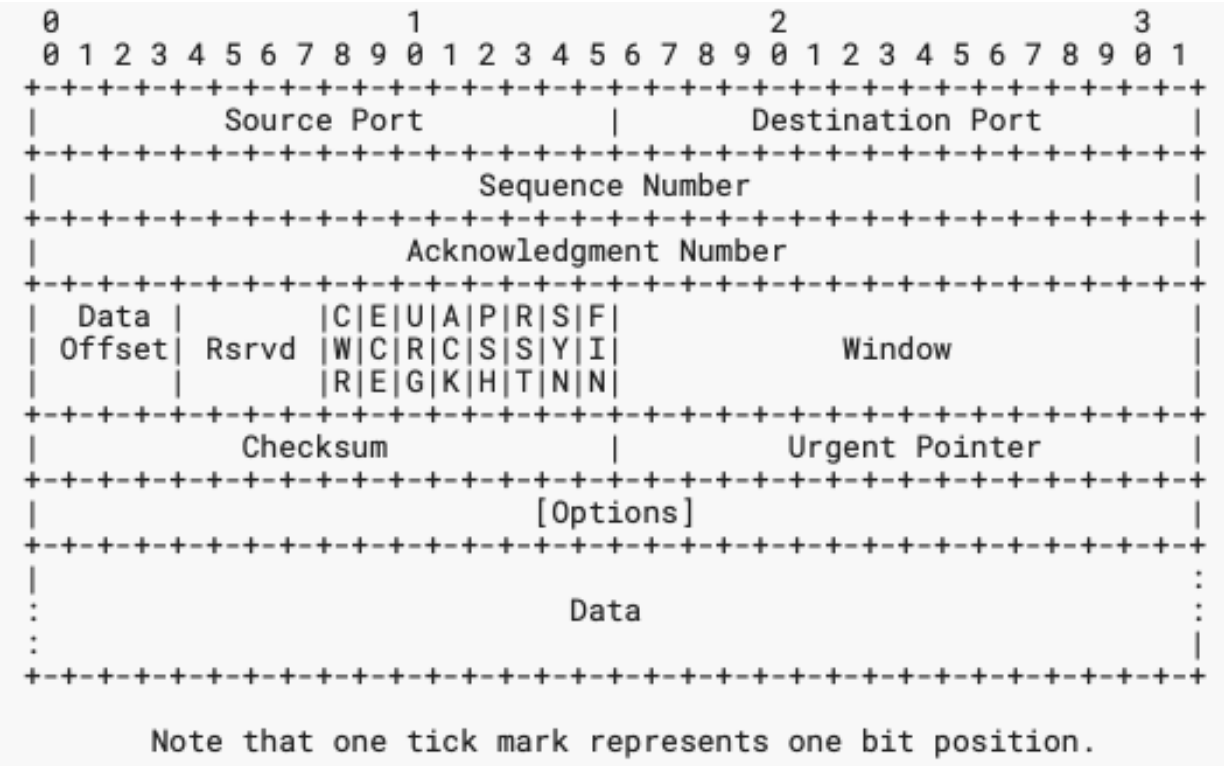

Figure 1: TCP Header

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 A and B (consider that the procedure is started by A)? Can you determine any instance of a graceful termination in this trace?

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

> **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. Identify a possible cause for these DupAcks. 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 analyze a packet trace, previously generated with iperf, that simulates a TCP connection using the **Reno congestion control algorithm**.

Open the following trace file `tcp-congestion.pcap` in Wireshark. Afterwards, 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.

To answer the following question, analyze the graph corresponding to stream 1, selecting the direction of **10.0.1.10 as the sender**.

Q7. Write down the approximate time (in seconds) where the first transition between the slow start and congestion avoidance phases occurs.

> **Hint.** To answer the previous question, you must use the provided trace file `tcp-congestion.pcap`. However, if you're curious, you can generate a similar simulation using the provided `tcp3.imn` topology in CORE. After starting an iperf3 server on `n2`, open Wireshark on n3 and start a capture with the tcp filter. Afterwards, run the following command on `n3`, which will send TCP packets over 5 flows using the Reno congestion control algorithm: 'iperf3 -c 10.0.0.10 -B 10.0.1.10 –cport 50000 -P 5 -t 60 -C reno'. It will run for 60 seconds. Afterwards, stop the wireshark capture. NOTE: The output of this

simulation will likely differ from the provided packet trace. Don't use this output to answer Q7!

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?