INP111 Lab03 Week #4

Date: 2022-10-06

#1 How Many Data Bytes are Received?

The purpose of this lab is to implement a simple TCP client to interact with a remote server.

Procedure

1. Play with the server manually using the command `nc inp111.zoolab.org 10002`.
2. Implement your program to interact with the server. You may follow the instructions in the *Demonstration* section.

Demonstration

1. Your program should request the server to start sending data using the `GO` command
2. Your program should count how many bytes of data（alphabets and digits）are enclosed in the two lines `==== BEGIN DATA ====` and `==== END DATA ====` received from the server. You do not need to count the data bytes for the two lines.
3. Your program should answer the question requested by the server.
4. Please print out the last message you received from the server after sending the answer to the server.

#2 Constant BitRate TCP Flow

The purpose of this lab is to implement a simple TCP client to send data at a constant bitrate.

Procedure

> If you have difficulties connecting to the Internet, or the bandwidth in your network is unstable. You can download our sink server from here (x86-64 (http://inp111.zoolab.org/lab03.2/sink), arm64 (http://inp111.zoolab.org/lab03.2/arm64/sink)) and modify your client to connect to the local server instead of our Internet server.
>
> You have to enable the execute permission for the sink server and then run the server using the command `./sink 10003`. You can check whether the server works correctly by running `nc localhost 10003`.
>
> Note that if you run the sink server in your docker, you have to capture packets using `tcpdump` in your docker instead of Wireshark. The commands to run tcpdump is `sudo tcpdump -ni lo tcp and dst port 10003 -w output.pcap`. You can then open your pcap file using Wireshark and use the **I/O graphs** feature in Wireshark.

1. Play with the server manually using the command `nc inp111.zoolab.org 10003`. 140.113.213.213.

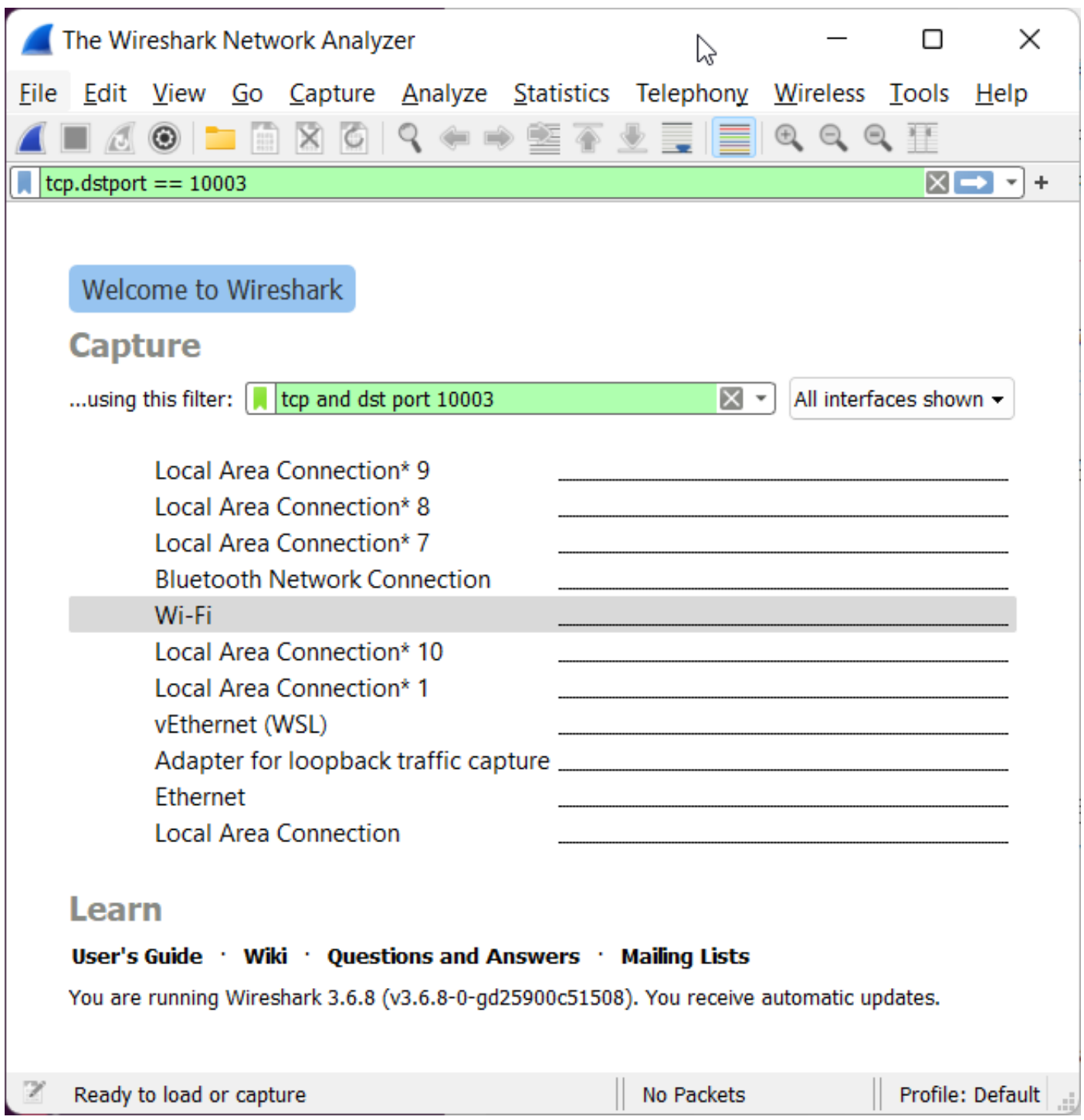
2. The server is a sink server that drops everything you send to the server.
3. Write a program to interact with the server. After receiving the first message from the server, your program should start sending data to the server until your program is terminated by the user.
4. Your program must control the data sending rate to the server, which is passed to your program via the first argument (in a unit of ***megabyte-per-second***)

> ***Note:*** Here, the definition of megabytes is 1,000,000 bytes.
>
> 10 bytes.

> A sample code for running an infinite loop, measuring elapsed time, and handling signals (program termination and user interruption [Ctrl-C]) is available here (https://inp111.zoolab.org/lab03.2/template.c) for your reference.

Demonstration

1. Run Wireshark on your host machine. Select the interface you used to connect to the Internet and start capturing packets. You can use a Wireshark filter `tcp.dstport == 10003` (or tcpdump filter `tcp and dst port 10003`) to capture packets. A sample Wireshark startup screenshot is shown below.



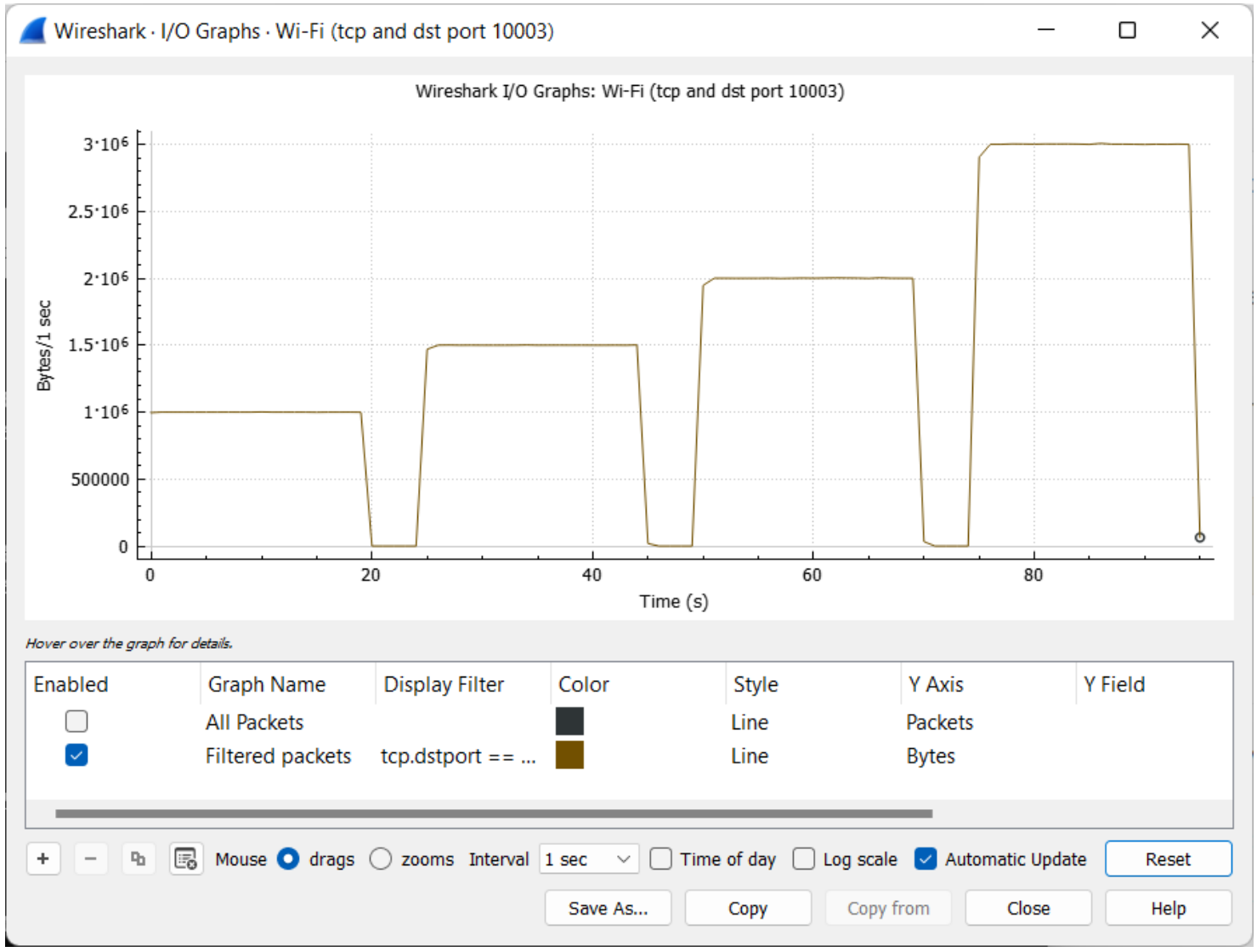
2. After you start capturing packets, activate the "I/O Graphs" window. It can be activated via the menu "Statistics" -> "I/O Graphs". You should add a new line that plots *byte-per-second* numbers for the filtered traffic. Note that the default settings for a line plot *packet-per-second* numbers. You should change it manually by yourself. You may look at the sample figure below to see how it should be appropriately configured.

> ***Note*** that you can only run your program using a regular user's permission. Run it with root permission is prohibited.

3. Suppose the program you implemented is `tcpcbr` and the first argument is the expected constant bitrate configured to send data. You can test your implementation with the following shell script.

```
#!/bin/sh
timeout 20 ./tcpcbr 1;   sleep 5  # send at 1 MBps
timeout 20 ./tcpcbr 1.5; sleep 5  # send at 1.5 MBps
timeout 20 ./tcpcbr 2;   sleep 5  # send at 2 MBps
timeout 20 ./tcpcbr 3
```

4. Check the "I/O Graphs" window to ensure that your program sends at the bitrate we expected. For your reference, a sample screenshot for the "I/O Graphs" window is shown below.



5. Grading Policy:

You must print out the data sending rate of your implementation. However, the grading rule is based only on the graph output from Wireshark.

- You get 100% of the points if your program sends precisely at the expected bitrate, just like the sample figure shown above.
- You get 95% of the points if your program sends at a rate within ± 0.1 MBps of the expected bitrate.
- You get 90% of the points if your program sends at a rate within ± 0.3 MBps of the expected bitrate.
- You get 80% of the points if your program sends at a rate within ± 0.6 MBps of the expected bitrate.
- Well, if you can send data to the server consistently at any rate higher than 1 MBps, you get 60% of the points.

> ***Hint***: Wireshark measures bitrate based on the entire packets captured on the wire. Therefore, you have to estimate how many header bytes are added to each packet to ensure that your program can send at a precise bitrate.