CSC/ECE 573: Internet Protocols
Take Home Mini Exam Projects

1. Packet Sniffer (60 Points: 50 Implementation and Experiment, 10 Report)

1.1 Objective:

This project aims at giving students an understanding of how raw socket interface and packet header parsing works. A raw socket allows an application to directly access lower-layer protocols, which means a raw socket receives un-extracted packets. There is no need to provide the port and IP address to a raw socket, unlike in the case of the stream and the datagram sockets. In this project, the students will get the experience of: 1) working with protocol headers at datalink, network, and transport layer, 2) designing data structures for the header of each protocol, and 3) mapping the header fields from the buffer in which packet has been received on to the data structure. After completing this project, students should have developed some confidence in understanding the network stack code of any open-source OS.

1.2 Description:

In this project, you are going to develop a packet sniffer tool. Your tool should be able to continuously read packets from the raw socket and parse their headers depending on the protocol. You can take help from tutorials on raw sockets available online. However, you are expected to develop your tool without using pre-existing codes on the internet (changing variable names does not amount to originality and it is very easy to spot). The minimum requirements for this project are:

1. Sniffing of all the packets that are received on the raw socket and parsing of headers for ethernet, IP, TCP, UDP, and DNS protocols. Extraction of data from packet is not required. You should only parse headers such that you can identify the protocol used on each layer.
2. Your tool should be able to categorize packets based on protocols (IP, TCP, UDP, and DNS) i.e., for the duration of execution of your tool, it should keep the count of all the packets received for these protocols.
    - **Points distribution**
        - **Successfully implementing this part will secure you 9 points for each protocol. So total points for getting correct packet count for four protocols is 36 out of 50.**
        - **Incorrect counts for each protocol will result in deduction [-4 points/protocol].**

3. When your program is executed, it should run for 30 seconds and then exist gracefully. There should be no error or exception at the time of compilation, execution, during runtime, and at the time of exit.
    - **Points distribution**
        - **Correct implementation of 30 seconds runtime and graceful exit [4 points].**
        - **Incorrect implementation of 30 seconds runtime [-4 points].**
        - **Failure to compile code due to any issue [-20 points].**
        - **Error at the time of execution, during runtime, or exit [-15 points].**
        - **Exception at the time of execution, during runtime, or exit [-15 points].**
        - **Not implementing code for any protocol [-10 points/protocol].**

4. When your program exits after 30 seconds, it should generate a .csv file name as sniffer_<unityid>.csv e.g., sniffer_hiqbal.csv. This file should have the count of packets of each of the fours protocols as mentioned in point (2). The format for .csv file *must* be as follows:

```
protocol,count
ip,192
tcp,61
udp,47
dns,22
```

   For example: if you receive a DNS packet, you will increment count for:
   - DNS as this is the protocol to identify
   - UDP because DNS uses UDP as transport
   - and IP because at network layer IP is used by DNS.

   Note: Every alphabet in .csv file is in lower case. There are no spaces before and after the comma. The counts represent the number of packets received for that protocol. The numbers above are shown as examples. Your counts will vary.

   - **Points distribution**
     - **Each protocol count in the file gives you 2.5 points. So, correct file generation in the specified format with counts for the four protocols will secure you [10 points].**
     - **No file generated after program exits [- 30 points].**
     - **File generated but it is empty [-25 points].**
     - **File generated but has incorrect format [-15 points].**

1.3 Bonus Task: (10 points)

To get bonus points, apart from categorizing protocols as mentioned in point (2) of section 1.2, you should also keep count of these four protocols: ICMP, HTTP, HTTPS, and QUIC.

Your output in a file, as mentioned in point (4) of section 1.2, now looks like as follows:

```
protocol,count
ip,250
tcp,64
udp,23
dns,20
icmp,46
http, 12
https,65
quic,98
```

For example: if you receive an HTTPS packet, you will increment count for:
- HTTPS as this is the protocol to identify
- TCP if HTTPS uses TCP as transport
- and IP because at network layer IP is used by HTTPS.

Note: Every alphabet in .csv file is in lower case. There are no spaces before and after comma. The counts represent the number of packets received for that protocol. The numbers above are shown as examples. Your counts will vary.

- **Points distribution**
  - **Each of these four protocols packet count will gives you 2.5 points per protocol. Therefore, correct file generation in the specified format with additional counts for these four protocols will earn you extra [10 points].**

1.4 Experiment:
Once your development is complete, please carryout following experiments with your tool.

Exp 1: In this experiment, you first play any YouTube video at highest resolution possible in a browser. Once video starts playing, run your tool. Once your tool exits itself after 30 seconds, you will get an output .csv file.

Exp 2: In this experiment, first run your tool. Then open a browser, go to YouTube website and quickly click on any video. Let this video play until the tool exits. Once your tool exits itself after 30 seconds, you will get an output .csv file.

Exp 3: In this experiment, run your tool. Then open a browser and randomly search stuff on google and open different websites until the tool exits. Once your tool exits itself after 30 seconds, you will get an output .csv file.

You are going to compare the results of these three experiments and discuss your observations in a report. You will also have to plot the results of these three experiments on a single figure. The format of the report and plot are provided in the template.

1.5 Submission Instructions:
We will test your code for successful compilation, execution, termination, and .csv file generation, which will contain the counts of packets for the protocols as mentioned earlier. Your results will be compared with our tool for correctness.

Your program will be tested on Ubuntu 16.04 LTS operating system. It is highly recommended to use Ubuntu 16.XX or 18.XX for this project. If you are using Windows or MAC machines, then install VMware workstation or virtual box on your machine and setup a virtual machine (VM) for Ubuntu 16.04 LTS.

If you are using VCL then make sure you do the following:

1. Take backup before turning off VM or before its reservation time expires. Your data *will* be lost if you do not take the backup.
2. If there are packages or dependencies required for this project, then you will have to install those of the VM every time you reserve it. Therefore, maintain a list of packages and dependencies which you have to install every time so that it may save you time.

For submission, students must make a folder named **project1** and place the following three files in it:

1. Code file named as sniffer_<unityid>.py, OR sniffer_<unityid>.c, OR sniffer_<unityid>.cc
2. Csv file named as sniffer_<unityid>.csv. Submit the file generated in experiment 2.

3. Readme file as readme_<unityid>.txt
    - Readme file should contain clear instructions about the following:
        - How to compile the code? Provide command.
        - How to execute the compiled code? Provide command.
        - Which packages and dependencies are required by your code for the successful compilation and execution of the code? Provide a list of all those packages and dependencies. Failure to provide this may lead to unsuccessful compilation and execution of your code due to which your points will be deducted as per the criteria mentioned in point (4) of section 1.2.

Once you have placed the three files in folder project1, go to section 4 below for final submission instructions.

Hints:

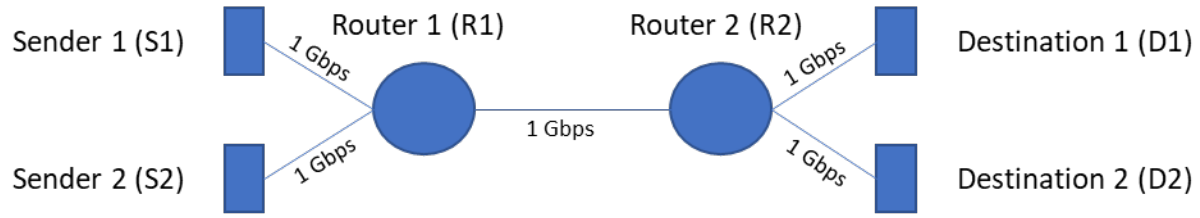
- *Take raw sockets, libpcap, or rawsocketpy as a starting point.*
- *Implement in python, C, or C++. Choose from only these three programming languages.*
- *The header of the lower layer has a field that identifies the protocol of the higher layer.*
- *Implementing this tool is easier than you think :)*

2. TCP Analysis (60 Points: 50 Implementation and Experiment, 10 Report)

2.1 Objective:
This project aims at giving students an understanding of the behaviors of two different TCP protocols when congestion occurs in network. These two TCP protocols are Cubic and DCTCP. Student will learn how each protocol behaves individually in the network and how both coexist with each other. The performance of these protocols will be evaluated in a simple network topology. At the end of this projects, students will have basic understanding of how performance of these two TCP protocols changes in different network condition and they will also get hands-on experience on ns-3.

2.2 Description:
In this project, you will assess the performance of TCP Cubic and DCTCP in terms of throughput and average flow completion time. You will implement a dumbbell topology in the ns-3 simulator and run multiple experiments to evaluate the performance of both TCP versions for the given metrics. You will demonstrate how well these two protocols coexist with each other and which one performs better in terms of responding to the congestion in the network. The simulator allowed in this project is ns-3. Details of network topology, measurement metrics, experiments, and deliverables are explained below.

2.3 Topology: [5 points]
You are required to set up a simulation of dumbbell topology as shown in the figure below:



There are two sender servers S1 and S2 which are connected to router R1 with 1 Gbps link each. There are two destination servers D1 and D2 which are connected to router R2 with 1 Gbps link each. Both R1 and R2 are connected with 1 Gbps link as well. Please note that for the given simple topology, the term router and switch can be treated as same.

- **Point distribution**
  - **Correct topology setup [5 points].**
  - **Any incorrect parameter [-5 points].**

2.4 Configuration: [5 points]
- S1 will send traffic to D1 only
- S2 will send traffic to D2 only
- All the servers i.e., S1, S2, D1, and D2 should be configured to support TCP Cubic (also known as TCP BIC in ns-3) and DCTCP. This parameter can be set according to the requirements of the experiments as described in section 2.6.

- **Point distribution**
  - **Correct configuration setup [5 points].**
  - **Any incorrect configuration [-5 points].**

2.5 Measurement Metric:

There are two metrics that you will study:

1. ***Throughput*** tells how much data can be transferred from a source to destination at any given time. Its unit is bits per second (bps).
2. ***Average Flow Completion Time*** measures the time taken by the sender to send the desired amount of data to the sender. It is measured in seconds.

The methodology of measuring these metrics is open-ended. Develop your way to measure them as accurately as possible. However, for both metrics, take the results of three runs for each experiment and then calculate the average and standard deviation for each experiment. More details on experiments are in the section 2.6.

2.6 Experiments: [25 points]

Once the topology is set and configurations are done, run the following experiments and measure *throughput* and *average flow completion time*. In each experiment, use bulk sender application in ns-3 to generate the traffic. Use bulk sender to send 50 GB of data from source to destination in each of the following experiments:

Exp-1: S1 sends traffic to D1 using TCP Cubic. S2 and D2 are not used in this experiment.

Exp-2: S1 sends traffic to D1 and S2 sends traffic to D2. Both senders will use TCP Cubic and start sending data to respective destinations simultaneously.

Exp-3: S1 sends traffic to D1 using DCTCP. S2 and D2 are not used in this experiment.

Exp-4: S1 sends traffic to D1 and S2 sends traffic to D2. Both senders will use DCTCP and start sending data to respective destinations simultaneously.

Exp-5: S1 sends traffic to D1 using TCP Cubic whereas S2 sends traffic to D2 using DCTCP. Both senders will start sending data to respective destinations simultaneously.

Run each experiment three times and then calculate average and standard deviation for the two metrics (mentioned in section 2.5) per sender.

Note: Implement you code such that it executes all the above five experiments one after the other and generates the results in a .csv file as specified in section 2.7. This is mandatory because we are going to test your code and it must generate results for all the experiments that you have implemented.

- **Point distribution**
  - **Each experiment has 5 points. Conducting all the experiment will give you [25 points].**
  - **Deduction for any experiment not performed [-5 points].**
  - **If the 'Note' given above is not followed [-15 points].**
  - **Failure to compile code due to any issue [-20 points].**
  - **Error at the time of execution, during runtime, or exit [-15 points].**
  - **Exception at the time of execution, during runtime, or exit [-15 points].**

2.7 Output Format: [15 marks]
You will have to provide the results in a csv file. The title of csv file *must* be tcp_<unityid> e.g., tcp_hiqbal.csv. The format of the file *must* be as follows:

exp,r1_s1,r2_s1,r3_s1,avg_s1,std_s1,unit_s1,r1_s2,r2_s2,r3_s2,avg_s2,std_s2,unit_s2,
th_1,520.12,547.85,690.98,586.32,91.70,Mbps,,,,,,
th_2,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>
th_3,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>
th_4,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>
th_5,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>
afct_1,20,17,26,21,4.58,sec,,,,,,
afct_2,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>
afct_3,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>
afct_4,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>
afct_5,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>,<>

- ✓ Column 'exp' represents 5 experiments. Rows th_# implies data for five throughput experiments. Row afct_# implies data for average flow completion time for five experiments.
- ✓ Columns 'r1_s1', 'r2_s1', 'r3_s1', 'avg_s1', 'std_s1', 'unit_s1' represents the 3 runs, average, standard deviation, and unit for each of the experiments involving sender S1. These columns will be filled for all the five experiments because S1 is part of all the experiments.
- ✓ Columns 'r1_s2', 'r2_s2', 'r3_s2' 'avg_s2', 'std_s2', 'unit_s2' represents the 3 runs, average, standard deviation, and unit for each of the experiments involving sender S2. These columns will be only filled for Exp-2, Exp-4, and Exp-5 because sender S2 is only used in these experiments. For experiment Exp-1 and Exp-3, keep these columns blank.
- ✓ For columns r1_, r2_, r3_, avg_, and std_, replace <> with the numerical values obtained from the experiments. Do not write unit in these columns.
- ✓ In column unit_, replace <> with appropriate units. For throughput use Gbps, Mbps, Kbps, or bps. For average flow completion time, use millisec or sec.
- ✓ There are no spaces before and after the commas.
- ✓ For example, two rows th_1 and afct_1 are filled to show how to present your data. The values shown are hypothetical. Your values will vary. Rows th_1 and afct_1 are for 'Exp 1' which only includes sender S1. Therefore, values for sender S2 will be blank.

- **Points distribution**
  - **Correct file generated according to the given format [15 points].**
  - **No file generation after execution of code [- 30 points].**
  - **File generated but it is empty [-25 points].**
  - **File generated but has incorrect format [-15 points].**

2.8 Submission Instruction:
We will test your code for successful compilation, execution, termination, and csv file. Your results will be compared with our implementation for correctness.

Your program will be tested on Ubuntu 16.04 LTS operating system. It is highly recommended to use Ubuntu 16.XX or 18.XX for this project. If you are using Windows and MAC machines, then install VMware workstation or virtual box on your machine and setup a virtual machine (VM) for Ubuntu 16.04.

VCL is not recommended for this project as this will require you to install ns-3 every time you make a reservation for a VM. This is time consuming process. Therefore, it is better to setup environment on your local machine.

For submission, students must make a folder named **project2** and place the following three files in it:

1. Code file named as tcp_<unityid>.cc
2. Csv file named as tcp_<unityid>.csv as explained in 2.7
3. Readme file as readme_<unity_id>.txt.
    - Readme file should contain clear instructions about the following:
        - How to compile the code? Provide command.
        - How to execute the compiled code? Provide command.
        - It should contain any extra information regarding your implementation such as delay used for point-to-point links etc.
        - Which packages and dependencies are required by your code for the successful compilation and execution of the code? Provide a list of all those packages and dependencies. Failure to provide this may lead to unsuccessful compilation and execution of your code due to which your points will be deducted as per the criteria mentioned in section 2.6.

Once you have placed the three files in folder project2, go to section 4 below for final submission instructions.

2.9 Documentation and Help:

ns-3:

- Installation Guide
    - https://www.nsnam.org/wiki/Installation
- Documentation
    - https://www.nsnam.org/documentation/
- Code Search and details
    - https://www.nsnam.org/doxygen/
- For a quick start, after installation, look for examples in folder *examples/tutorials/ and example/tcp/*
- You can search for quick setup tutorials on YouTube as well.

3. Report (20 points: 10 for each project)

The template for the report is provided in the attached file. Please follow the instructions below while writing the report:

1. The report must be to-the-point. There is no need of long details.
2. Only follow the format of the template and fill in the required data.
3. There are 10 points for each project in the report.
4. Report must be submitted in .pdf format. Name the report as <unityid>.pdf.
5. Once the report is complete, go to section 4 below for final submission instructions.

4. Final Submission

Once you have created both folders project1 and project2 and added the respective files in them, zip these two folders along with the report into a single .zip file. Name that .zip file as <unityid>.zip and submit it on Moodle. Your submitted .zip file contains following:

- ✓ project1
- ✓ project2
- ✓ <unityid>.pdf

Good Luck!