Assignment 1: Sockets, Mininet, & Performance

Due: TBC

Overview

`iPerf` is a common tool used to measure network bandwidth. You will write your own version of this tool in C/C++ using sockets. You will then use your tools to measure the performance of virtual networks in Mininet and explain how link characteristics and multiplexing impact performance.

- Part 1: Write `iPerfer`
- Part 2: Mininet Tutorial
- Part 3: Measurements in Mininet
- Part 4: Real-World Latency (Bonus marks, optional)
- Submission Instructions
- Autograder

Learning Outcomes

After completing this programming assignment, students should be able to:

- Write applications that use sockets to transmit and receive data across a network
- Explain how latency and throughput are impacted by link characteristics and multiplexing

Part 1: Write `iPerfer`

For the first part of the assignment you will write your own version of `iPerf` to measure network bandwidth. Your tool, called `iPerfer`, will send and receive TCP packets between a pair of hosts using sockets.

> **NOTE:** You may refer to Beej's Guide to Network Programming Using Internet Sockets for socket programming. Tutorials will also review the some of the basics.

When operating in client mode, `iPerfer` will send TCP packets to a specific host for a specified time window and track how much data was sent during that time frame; it will calculate and display the bandwidth based on how much data was sent in the elapsed time. When operating in server mode, `iPerfer` will receive TCP packets and track how much data was received during the lifetime of a connection; it will calculate and display the bandwidth based on how much data was received and how much time elapsed during the connection.

Server Mode

To operate `iPerfer` in server mode, it should be invoked as follows:

```
./iPerfer -s -p <listen_port>
```

- `-s` indicates this is the `iPerfer` server which should consume data

- `listen_port` is the port on which the host is waiting to consume data; the port should be in the range `1024 ≤ listen_port ≤ 65535`

> For simplicity, you can assume these arguments will appear exactly in the order listed above.

You can use the presence of the `-s` option to determine `iPerfer` should operate in server mode.

If arguments are missing or extra arguments are provided, you should print the following and exit:

`Error: missing or extra arguments`

If the listen port argument is less than 1024 or greater than 65535, you should print the following and exit:

`Error: port number must be in the range of [1024, 65535]`

When running as a server, `iPerfer` must listen for TCP connections from a client and receive data as quickly as possible. It should then wait for some kind of message from the client indicating it is done sending data (we will call this a FIN message). The server should then send the client an acknowledgement to this FIN message. It is up to you to decide the format of these FIN and acknowledgement messages.

Data should be read in chunks of 1000 bytes. Keep a running total of the number of bytes received.

> You may set the chunks to 100000 bytes for mininet measurements.

After the client has closed the connection, `iPerfer` server must print a one-line summary in the following format:

`Received=X KB, Rate=Y Mbps`

where X stands for the total number of bytes received (in kilobytes), and Y stands for the rate at which traffic could be read in megabits per second (Mbps).
Note X should be an integer and Y should be a decimal with three digits after the decimal mark.

For example:
`Received=6543 KB, Rate=5.234 Mbps`

The `iPerfer` server should shut down gracefully after it handles one connection from a client.

> **Note:** Please use setsockopt to allow reuse of the port number, this will make your life easier for testing, which runs the `iPerfer` server with the same port number each time.

Client Mode

To operate `iPerfer` in client mode, it should be invoked as follows:

`./iPerfer -c -h <server_hostname> -p <server_port> -t <time>`

- `-c` indicates this is the `iPerfer` client which should generate data
- `server_hostname` is the hostname or IP address of the `iPerfer` server which will consume data
- `server_port` is the port on which the remote host is waiting to consume data; the port should be in the range 1024 ≤ `server_port` ≤ 65535

- `time` is the duration in seconds for which data should be generated. We will only test this with an integer value (i.e feel free to use time.h)

> For simplicity, you can assume these arguments will appear exactly in the order listed above.

You can use the presence of the `-c` option to determine that `iPerfer` should operate in the client mode.

If any arguments are missing or extra arguments are provided, you should print the following and exit:

```
Error: missing or extra arguments
```

If the server port argument is less than 1024 or greater than 65535, you should print the following and exit:

```
Error: port number must be in the range of [1024, 65535]
```

If the time argument ends up parsing to less than 0, you should print the following and exit:

```
Error: time argument must be greater than 0
```

If both the port and time argument are invalid, print only the port error message.

When running as a client, `iPerfer` must establish a TCP connection with the server and send data as quickly as possible for `time` seconds. Data should be sent in chunks of 1000 bytes and the data should be all zeros. Keep a running total of the number of bytes sent. After the client finishes sending its data, it should send a FIN message and wait for an acknowledgement before exiting the program.

> You may set the chunks to 100000 bytes for mininet measurements.

`iPerfer` client must print a one-line summary in the following format:

```
Sent=X KB, Rate=Y Mbps
```

where X stands for the total number of bytes sent (in kilobytes), and Y stands for the rate at which traffic could be read in megabits per second (Mbps).
Note X should be an integer and Y should be a decimal with three digits after the decimal mark.

For example:
`Sent=6543 KB, Rate=5.234 Mbps`

You should assume 1 kilobyte (KB) = 1000 bytes (B) and 1 megabyte (MB) = 1000 KB. As always, 1 byte (B) = 8 bits (b).

> **NOTE:** When calculating the rate, do not use the `time` argument, rather measure the time elapsed from when the client first starts sending data to when it receives its acknowledgement message.

Testing

You can test `iPerfer` on any machines you have access to. However, be aware the certain ports may be blocked by firewalls on end hosts or in the network, so you may not be able to test your program on all hosts or in all networks.

The primary mode for testing should be using Mininet. You should complete Part 2 of this assignment before attempting that.

You should receive the same number of bytes on the server as you sent from the client. However, the timing on the server may not perfectly match the timing on the client. Hence, the bandwidth reported by client and server may be slightly different.

Instructions for submission are here. It is not meant to be your primary source of testing/debugging, but is rather intended for you to see your overall progress.

Part 2: Mininet Tutorial

To test `iPerfer`, you will learn how to use Mininet to create virtual networks and run simple experiments. According to the Mininet website, *Mininet creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM or native), in seconds, with a single command.* We will use Mininet in programming assignments throughout the semester.

Running Mininet

It is best advised to run Mininet in a virtual machine (VM). We will be using VirtualBox, which is a free and open-source hypervisor. Please download and install the latest version of VirtualBox.

You will be using VM image (link here) with Mininet 2.3 pre-installed. Please download and import the VM image into VirtualBox. To transfer files to/from your VM you can use the Shared Folder feature provided in VirtualBox. We will go over this in more detail in tutorials.

You are welcome to try to set up your own testing environment using the methods outlined in options 2 and 3 here, however we will only officially be supporting the provided VM above.

Mininet Walkthrough

Once you have a Mininet VM, you should complete the following sections of the standard Mininet walkthrough:

- All of Part 1, except the section "Start Wireshark"
- The first four sections of Part 2—"Run a Regression Test", "Changing Topology Size and Type", "Link variations", and "Adjustable Verbosity"
- All of Part 3

At some points, the walkthrough will talk about software-defined networking (SDN) and OpenFlow. We will discuss these during the second half of the semester, so you do not need to understand what they mean right now; you just need to know how to run and interact with Mininet.

> You do not need to submit anything for this part of the assignment.

Part 3: Measurements in Mininet

For the third part of the assignment you will use the tool you wrote (`iPerfer`) and the standard latency measurement tool `ping` (`ping` measures RTT), to measure the bandwidth and latency in a virtual network in Mininet. You must include the output from some of your experiments and the answers to the questions below in your submission. Your answers to the questions should be put in the file `answers.txt` **that we provide**. Please do **NOT** change the format of the `answers.txt` file (none of the answers to the questions should take more than one or two sentences).

Read the `ping` man page to learn how to use it.

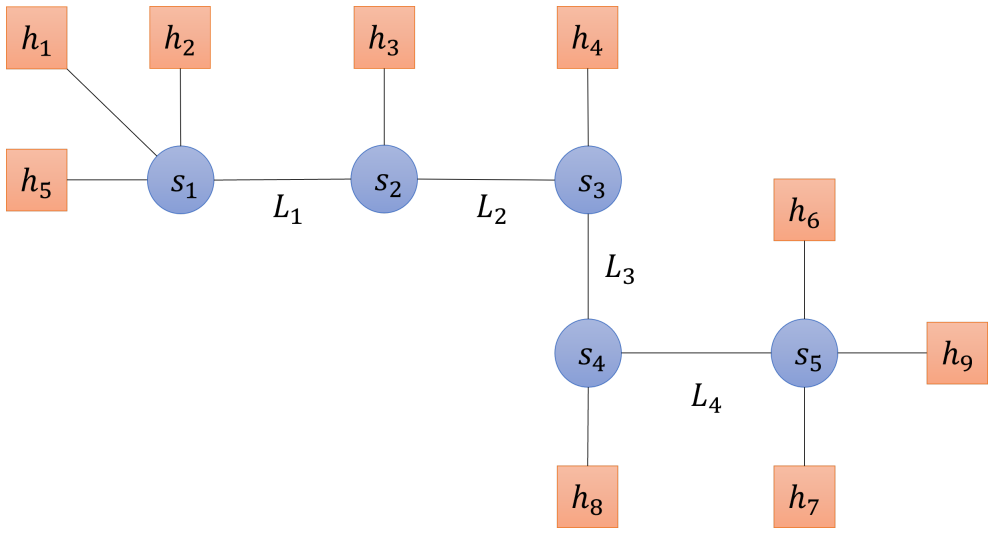
A python script to run Mininet with the topology described below is provided here along with other files that you will find useful in completing this assignment.

To run Mininet with the provided topology, run the Python script `assignment1_topology.py` using sudo:

```
sudo python assignment1_topology.py
```

> Note: You should use **Python 2.7** in this assignment.

This will create a network with the following topology:



If you have trouble launching the script, a common fix is to first try running `sudo mn -c`, and then try launching the script again.

Hosts (`h1` to `h9`) are represented by squares and switches (`s1` to `s5`) are represented by circles; the names in the diagram match the names of hosts and switches in Mininet. The hosts are assigned IP addresses 10.0.0.1 through 10.0.0.9; the last number in the IP address matches the host number.

> **NOTE:** When running ping and `iPerfer` in Mininet, you must use IP addresses, not hostnames. Also, if you are not confident your `iPerfer` is working correctly, feel free to use `iperf` for any throughput measurements noted below. Output using either program will be accepted.

Q1: Link Latency and Throughput

First, you should measure the RTT and bandwidth of each of the four individual links between switches (`L1` - `L4`). You should run ping with 20 packets and store the output of the measurement on each link in a file called `latency_L#.txt`, replacing # with the link number from the topology diagram above. You should run `iPerfer` for 20 seconds and store the output of the measurement on each link in a file called `throughput_L#.txt`, replacing # with the link number from the topology diagram above.

Q2: Path Latency and Throughput

Now, assume `h1` wants to communicate with `h9`. What is the expected latency and throughput of the path between the hosts? Put your prediction in the `answers.txt` file under question 2.

Measure the latency and throughput between `h1` and `h9` using `ping` and `iPerfer`. It does not matter which host is the client and which is the server. Use the same parameters as above (20 packets / 20 seconds) and store the output in files called `latency_Q2.txt` and `throughput_Q2.txt`. Put the average RTT and measured throughput in the `answers.txt` file and explain the results. If your prediction was wrong, explain why.

Q3: Effects of Multiplexing

Next, assume multiple hosts connected to `s1` want to simultaneously talk to hosts connected to `s5`. What is the expected latency and throughput when two pairs of hosts are communicating simultaneously? Put your predictions in your `answers.txt` file under question 3.1.

Use `ping` and `iPerfer` to measure the latency and throughput when there are two pairs of hosts communicating simultaneously; it does not matter which pairs of hosts are communicating as long as one is connected to `s1` and one is connected to `s5`. Use the same parameters as above. You do not need to submit the raw output, but you should put the average RTT and measured throughput for each pair in your `answers.txt` file under question 3.1 and explain the results. If your prediction was wrong, explain why.

Repeat for three pairs of hosts communicating simultaneously and put your answers in `answers.txt` under question 3.2.

Do not worry too much about starting the clients at the exact same time. So long as the connections overlap significantly, you should achieve the correct results. One simple way is to open up terminals for each of the hosts you will use, start the iPerfer servers, type in the iPerfer client command on each of the client hosts without hitting ENTER, and then quickly hit ENTER on all client hosts so that they start at roughly the same time.

Q4: Effects of Latency

Lastly, assume `h1` wants to communicate with `h9` at the same time `h3` wants to communicate with `h8`. What is the expected latency and throughput for each pair? Put your prediction in your `answers.txt` file under question 4.

Use `ping` and `iPerfer` to conduct measurements, storing the output in files called `latency_h1-h9.txt`, `latency_h3-h8.txt`, `throughput_h1-h9.txt`, and `throughput_h3-h8.txt`. Put the average RTT and measured throughput in your `answers.txt` file and explain the results. If your prediction was wrong, explain why.

Part 4: Real-World Latency (Optional, Bonus Marks)

For the fourth part, you should measure the RTT of three domain names in mainland China, Hong Kong SAR, and the USA. You should run ping with 20 packets and store the outputs of the measurements in files called `latency_CN.txt`, `latency_HK.txt`, and `latency_US.txt`. Explain the differences in the results and put your answers in answers.txt under question 5.

> **Note**: You can check the IP address in the ping feedback message to ensure the it definitely comes from the desired destination.
> You can pick any domain name in the three regions. Some domains have configured their switches to ignore ping packets, so you won't get any measurements in that case. Just try other domain names.
> You will notice that where you launch the ping matters a lot. The results are likely to be

> different when you ping from a machine inside the CSE or CUHK network and from your home network. Try to think about why, and share your thoughts on Piazza.

Submission Instructions

Your assigned repository must contain:

- The source code for `iPerfer`: all source files for `iPerfer` should be in a folder called `iPerfer`; the folder should include a `Makefile` to compile the sources.
- Your measurement results and answers to the questions from Part 3 and Part 4: all results and a text file called `answers.txt` should be in a folder called `measurements`.

Example final structure of repository:

```
$ tree ./p1/
./p1/
├── README.md
├── assignment1_topology.png
├── assignment1_topology.py
├── iPerfer
|   ├── ** source files ** <- either an iPerfer.c file or an iPerfer.cpp file
|   ├── Makefile <- supports "make clean" and "make"
|   └── iPerfer <- executable from running make
└── measurements
    ├── answers.txt
    ├── latency_L1.txt
    ├── latency_L2.txt
    ├── latency_L3.txt
    ├── latency_L4.txt
    ├── latency_Q2.txt
    ├── latency_h1-h9.txt
    ├── latency_h3-h8.txt
    ├── latency_CN.txt
    ├── latency_HK.txt
    ├── latency_USA.txt
    ├── throughput_L1.txt
    ├── throughput_L2.txt
    ├── throughput_L3.txt
    ├── throughput_L4.txt
    ├── throughput_Q2.txt
    ├── throughput_h3-h8.txt
    └── throughput_h1-h9.txt
```

Autograder

The autograder tests the following aspects of `iPerfer`

1. Incorrect argument handling
2. Format of your iPerfer output
3. Correctness of iPerfer output (both the `Sent` and `Received` values as well as `Rate`).

Because of the guarantees of TCP, both Sent and Received should be the same. The `Rate` is tested by first running `iperf` over a link, then comparing your `iPerfer` output to the result given a reasonable margin of error.

Submission methods:
Log in with your CUHK link email and password (will be sent by email after add/drop period) to this URL under the connection of CSE VPN.

You need to finish **both** the "Performance metrics" and the "Socket programming" parts.

Please find this link for general use of the autograder.

Our autograder runs the following versions of gcc/g++, please make sure your code is compatible

```
$ gcc --version
gcc (Ubuntu 11.2.0-16ubuntu1) 11.2.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

$ g++ --version
g++ (Ubuntu 11.2.0-16ubuntu1) 11.2.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Acknowledgements

This programming assignment is for CSCI4430 Fall 2023 of CUHK, designed by @Shaofeng Wu.

This programming assignment is based on Mosharaf Chowdhury's Assignment 1 from Umich EECS 489: Computer Networks.