COMP 7005
Computer Systems Technology
September 2023
Network Security

Project

This is a pair project.

Objective

Simulate a lossy network.

Tasks

- Create a reliable protocol based on UDP (the UDP data portion holds your protocol).
- ***You cannot use TCP at all (see below).***
- You must support IPv4 and IPv6.
- Create a sender that reads from the keyboard and writes to a UDP socket.
- Create a receiver that reads from a UDP socket and writes to the console.
- Create a proxy that sits between the two hosts that:
    - Randomly drops data from the sender
    - Randomly drops acks from the receiver
    - Randomly delays packets from either the sender or receiver.
- You can choose how to handle the delay (fixed, random, etc…)
- The receiver must send an acknowledgement back to the sender (via the proxy)
- If the sender does not get the acknowledgement in a reasonable time (as determined by you), it resends the packet.
- A GUI that graphs the data on the sender, receiver, and proxy (this part can use TCP, depending on how it is designed).

Constraints

- The sender must take command line arguments for the IP address of the proxy (or receiver if the proxy is removed), and the port.
- The receiver must take a command line argument for the port to receive from.
- The proxy must take command line arguments for the IP address of the receiver, the port to send to, and the port to listen on
- The proxy must take the % chance to drop data, drop acks, delay data, and delay acks on the command line.
- All 3 programs must maintain a list of statistics showing how many packets have been sent and received.

- The statistics must be stored in a file.
- You must provide a design for the sender, receiver, proxy, and (optional) grapher.
- You must provide videos (tmux us a good idea) of the sender, receiver, and proxy working.
- You can obtain bonus marks for:
    - Dynamically changing the % chance to drop or delay at runtime.
    - Implementing a window-based protocol instead of send and wait.

Submission Requirements

Use the following directory structure:

Directory	Purpose
source	Any source code files
report	Report files in .pdf format
video	Video(s) demonstration of your working project
pcap	Any relevant packet captures

Notes

- Follow the appropriate report format (samples).
- The demo video should cover each one of your test cases. In other words, it will be similar to an actual lab demo, except you will prepare a video of each test instead of me observing each test.
- You can have a separate video of all test cases.
- During the test, you will capture network traffic on any relevant machines and then submit the pcap files as specified above.
- Please set the appropriate packet capture filter to limit the size and scope of the data collected to be what is necessary.

Format

You must hand in a pax.Z file to the assignment submission folder on Learning Hub (https://lear.bcit.ca).

You can hand in as many versions as you like. The last submission, based on the timestamp, will be the one to be marked.

```
pax -w report/ source/ video/ pcap/ -f project-v#.pax
```

```
compress -f project-v-#.pax
```

Hand in the resulting project-v-#.pax.Z file.

Note: failure to follow the submission requirements may result in a loss of marks, up to 100%.

Evaluation

Item	Value
Client (sends data)	20
Server (sends acks)	10
Proxy	20
Graph	20
Report	20
Data (pcap, videos)	10
Total	**100**

Notes

- Please put a chart at the start of the report that clearly shows any bonus or “make it better” tasks.
- For graphing, the program can do it directly, or you can write a separate program.
- Using a separate program, you can send the data via TCP to another program that does the graphing.
- Each program: sender, receiver, proxy, and (optional) grapher can be written in different languages.
- For example, the sender could be C, the receiver could be Java, the proxy could be Go, and the (optional) grapher could be Python.
- From a learning perspective, the grapher program is far better than having the programs do the graphing.