Davin Deol, Paul Li
Set: 5J

Test Document

Protocol Characteristics

- Configuration File
- Packet Loss
- Timeouts
- Half-Duplex
- Sliding Window

Configuration File

The first thing that must be done to run the program is to run the network emulator server and receiver server. Before the servers can start receiving packets however, they must access the configuration file that contains the IP address and port number that they must know in order to understand where to send/receive packets from.

The transmitter uses the configuration file to get the IP address and port number of the network emulator so that it knows where to send/receive packets from. The network emulator uses it to get its own IP address and port number so that it can be bounded, as well as the receiver's IP address and port number so that it knows where to send packets to. The receiver uses the configuration file for the same reason as the network emulator, except it needs the IP address and port number for the network emulator so that it knows where to receive packets from.

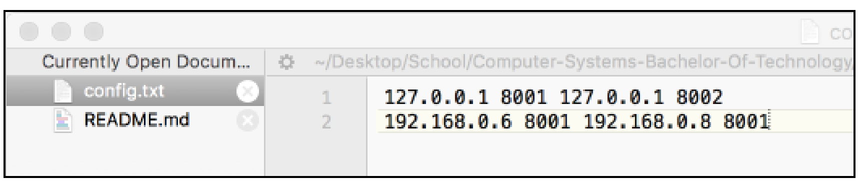
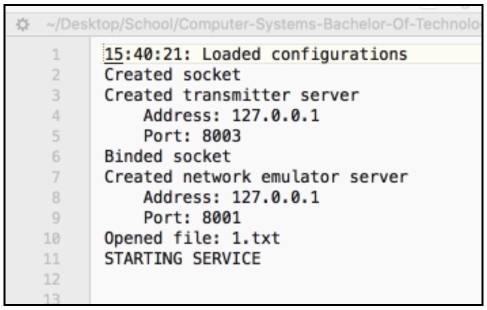


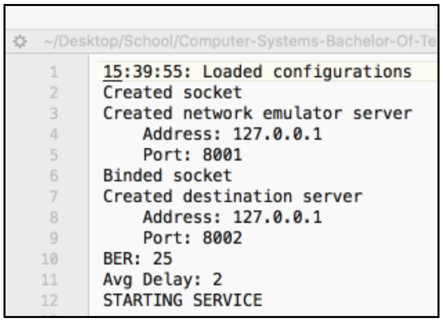
Figure 1: Sample configuration file text

To ensure that the program has accessed and knows the proper IP address and port numbers, logs are generated showcasing the server's IP address and the destination IP address. The logs should appear like the samples below:



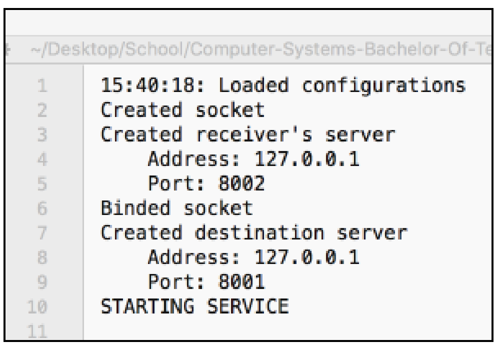
```
15:40:21: Loaded configurations
Created socket
Created transmitter server
    Address: 127.0.0.1
    Port: 8003
Binded socket
Created network emulator server
    Address: 127.0.0.1
    Port: 8001
Opened file: 1.txt
STARTING SERVICE
```

Figure 2: Transmitter logs. Note that the transmitter gets its IP address and port number from the user's inputs.



```
15:39:55: Loaded configurations
Created socket
Created network emulator server
    Address: 127.0.0.1
    Port: 8001
Binded socket
Created destination server
    Address: 127.0.0.1
    Port: 8002
BER: 25
Avg Delay: 2
STARTING SERVICE
```

Figure 3: Network Emulator logs



```
15:40:18: Loaded configurations
Created socket
Created receiver's server
    Address: 127.0.0.1
    Port: 8002
Binded socket
Created destination server
    Address: 127.0.0.1
    Port: 8001
STARTING SERVICE
```

Figure 4: Receiver logs

Packet Loss (Bit Error Rate)

Packet losses are a result of a packet being selected to not be forwarded to the destination (dropped). The way the selection occurs is first the network emulator server asks the user for the Bit Error Rate (BER) by entering a number from 0-99. Then when it comes time to forward a packet, a random number is generated from 0-100. If the random number is equal to or lesser than the BER, the packet is dropped.

The network emulator's logs specifically tells us which packets are dropped, including there data type.

ktop/School/Computer-Systems-Bachelor-Of-Tec

```
Waiting for DATA...
15:40:23: Dropped DATA[1]
DATA[2] >>> 127.0.0.1:8002
DATA[3] >>> 127.0.0.1:8002
EOT[4] >>> 127.0.0.1:8002

Waiting for ACKs...
127.0.0.1:8003 <<< ACK[2]
127.0.0.1:8003 <<< ACK[3]
127.0.0.1:8003 <<< EOT[4]
```

Figure 5: Network Emulator's logs. Note that only 3 packets were forwarded to the Receiver and only 3 packets were forwarded from the Receiver afterwards.

Timeouts

Timeouts only occur at the transmitter. It occurs when the transmitter doesn't receive an EOT packet within a specified time. When this occurs, the transmitter sends all the packets it can within its scrolling window. It also logs the timeout for the purposes of debugging.

logTransmitter.txt

~/Desktop/School/Computer-Systems-Bachelor-Of-Technology/Level-5/Computer-Net

```

Window After Creating Packets { EOT[1], -2-, -3-, -4- }
Sent EOT[1]

15:40:32: ===Timeout occurred===

Window After ACKs { EOT[1], -2-, -3-, -4- }


Window After Creating Packets { EOT[1], -2-, -3-, -4- }
Resent EOT[1]
```

Half-Duplex

The servers in the program can only receive OR send packets at a time. This rule is implemented by having all the servers only send packets if an EOT packet has been received, with the exception of the transmitter which has a timeout. The key player in keeping this half-duplex is the network emulator, which keeps this rule in check by only sending or receiving. We can ensure that this is working by looking at the network emulator's logs, mainly the switch between "Waiting for DATA..." and "Waiting for ACKs...".

```
Sent DATA[9]
Sent DATA[10]
Sent DATA[11]
Sent EOT[12]

19:25:04: Received
19:25:11: ===Timeo

Window After ACKs


Window After Creat
Resent DATA[9]
Resent DATA[10]
Resent EOT[12]

19:25:15: Received
Received ACK[12]

Window After ACKs

Created EOT[13], L

Window After Creat
Sent DATA[10]
Sent EOT[13]

19:25:19: Received
Received ACK[13]
```

```
19:25:00: Waiting for DATA...
19:25:02: Dropped DATA[9]
DATA[10] >>> 127.0.0.1:8002
DATA[11] >>> 127.0.0.1:8002
EOT[12] >>> 127.0.0.1:8002

Waiting for ACKs...
19:25:04: Dropped ACK[10]
127.0.0.1:8080 <<< ACK[11]
Dropped EOT[12]

Waiting for DATA...
DATA[9] >>> 127.0.0.1:8002
DATA[10] >>> 127.0.0.1:8002
EOT[12] >>> 127.0.0.1:8002

19:25:13: Waiting for ACKs...
127.0.0.1:8080 <<< ACK[9]
19:25:15: Dropped ACK[10]
127.0.0.1:8080 <<< EOT[12]

Waiting for DATA...
DATA[10] >>> 127.0.0.1:8002
EOT[13] >>> 127.0.0.1:8002

19:25:17: Waiting for ACKs...
127.0.0.1:8080 <<< ACK[10]
127.0.0.1:8080 <<< EOT[13]
```

```
19:25:02: Received
Received DATA[11],
Received EOT[12],

Sent ACK[10]
Sent ACK[11]
Sent ACK[12]

19:25:13: Received
Received DATA[10],
Received EOT[12],

Sent ACK[9]
Sent ACK[10]
Sent ACK[12]

19:25:17: Received
Received EOT[13],

Sent ACK[10]
Sent ACK[13]
```

The timestamps also help us as they tell us what occurred at a certain second. As you can see by the screenshots, there is no point where the emulator receives and sends packets from the same direction. Therefore, the half-duplex rule is being following.

Sliding Window

The sliding window is kept on the transmitter and is implemented using an array of packets. A typical process goes as follows:

1. Create packets if the last packet number plus the amount of empty spaces at the end is greater than the next packet(s) to be created.
2. Send the packets and wait for ACKs.
3. Set packets that have been ACK'd to uninitialised, signifying that they can be replaced.
4. Shift packet(s) to the left if there's room to the left of it. This occurs on the first initialised packet only. Other packets must then move the same distance as the first packet to be moved.
5. Loop from 1-4 until all packets have been sent and ACK'd.

Each of these steps are logged to keep track of the window's progress.

```
Window After Creating Packets { DATA[5], DATA[6], DATA[7], EOT[8] }
Resent DATA[5]
Resent DATA[6]
Resent DATA[7]
Resent EOT[8]

15:40:57: Received ACK[5]
Received ACK[6]
Received ACK[7]
Received ACK[8]

Window After ACKs { -5-, -6-, -7-, -8- }

Created DATA[9], LEN: 255, ACK: 2296
Created DATA[10], LEN: 255, ACK: 2551
Created DATA[11], LEN: 255, ACK: 2806
Created EOT[12], LEN: 255, ACK: 3061

Window After Creating Packets { DATA[9], DATA[10], DATA[11], EOT[12] }
Sent DATA[9]
Sent DATA[10]
Sent DATA[11]
Sent EOT[12]
```