CS 118 - Project 2

Alex Crosthwaite – Jacob Nisnevich – Jason Yang

June 7, 2016

1 Design

As a group, we implemented two classes to create our TCP packet abstraction, `Packet` and `Packet_info`. We also made the decision to *not* implement classes for the client and server for this project. Rather, we decided to simply execute the client and server code line-by-line in `client.cpp` and `server.cpp` respectively.

1.1 Packet

In our project, we used the `Packet` class for as our TCP packet abstraction. The class contained `SYN`, `ACK`, and `FIN` flags, in addtion to member variables for the sequence number, acknowledgment number, recieve window, and the data contained in the packet. The class could either be initialized with each of these values or with default values.

1.2 Packet_info

We also implemented a `Packet_info` class that we used to keep track of data length, the time the packet was sent, and the timeout time for the packet.

1.3 Client

Overall, our implementation of the client was very much by-the-book and straightforward. As such, this will be a general run-through of our implementation of the client.

The first step in our client execution is to set-up the UDP socket. To do this, we created a helper function, `set_up_socket`, that takes in the command-line arguments and returns the socket file descriptor. This helper function sets up an `addrinfo` struct and finds and connects to an available socket.

The client then selects a random initial sequence number, initializes a `Packet` with this sequence number and a `SYN` flag, and sends it along the UDP socket that was previously set up and increments the sequence number. The client then sits in a loop until a `Packet` with `SYN` and `ACK` flags arrives, and then increments the sequence number. Then, the client sends a `Packet` with an `ACK` flag and again increments the sequence number.

Now that the connection has been established, the client initializes a window, which is implemented as an `unordered_map` of `uint16_t` to `Packet_info`, and enters a loop that continues until a `Packet` with a `FIN` flag has been received. Within the loop, the client discards all invalid `ACK` packets, updates the window with new packets, and either sends an acknowledgement if a data packet was recieved or a `FIN/ACK` if a `FIN` packet was received. The received data is written to the "received.data" file and the socket is closed when the client recieves the final `ACK` packet.

1.4 Server

2 Problems and Solutions

2.1 Binary File Transfer Issues

2.2 Bit Vector Issues

2.3 Congestion Control Issues

2.4 Debugging Issues

2.5 Extra Bits at the End of Packets

3 Extra Credit

For the extra credit portion of the project, our group chose to implement TCP Reno fast retransmission and fast recovery as well as adaptive RTO.

4 Build Instructions

For Project 2 we did not modify the Makefile or Vagrantfile and just used the suggested virtual machine and g++ version. To build the project, simply initiate the two Ubuntu-based virtual machines with:

```
vagrant up
```

Then, `ssh` into both the client and server machines with:

```
vagrant ssh client
vagrant ssh server
```

Finally, build the project with:

```
make client
make server
```

5 Test Cases

5.1 No Loss

For the no packet loss testing, we tried sending four different files:

- `small.txt`
- `server.cpp` (medium-size)
- `large.txt`
- `client` (binary)

5.2 10% Loss on Client/Server

We also tested the TCP connection with 10% loss on both the client and the server. We tested this using the given command to emulate a network with 10% loss as follows:

```
tc qdisc add dev eth1 root netem loss 10%
```

Using this emulated network, we sent the same files as with the no packet loss testing.

5.3 20% Loss on Client/Server

In addition to testing with no loss and 10% loss, we tested with an emulated network with 20% loss. This was done with the following command:

```
tc qdisc add dev eth1 root netem loss 20%
```

Again, we sent the same four files as with the previous two testing scenarios.

5.4 Packet Re-ordering

6 Contributions

6.1 Alex Crosthwaite

6.2 Jacob Nisnevich

6.3 Jason Yang