Jeff Foreman

Nimitt Tripathy

CSE 160

18 November 2017

Project 3 Discussion Questions

1: Your transport protocol implementation picks an initial sequence number when establishing a new connection. This might be 1, or it could be a random value. Which is better, and why?

The best sequence number would be to start with a set number, such as 1. This is better because that way the sequence numbers are more organized. This sort of organization can make it where a node can recognize if a new node has been established, or that a node has restarted. With random numbers, this is impossible to tell.

2: Your transport protocol implementation picks the size of a buffer for received data that is used as part of flow control. How large should this buffer be, and why?

This buffer size should be picked as the difference between the Last Written or Last Read and the Maximum Buffer Size (The total size reserved for the buffer). This is so that data that is being written or read does not overload the buffer.

3: Our connection setup protocol is vulnerable to the following attack. The attacker sends a large number of connection request (SYN)packets to a particular node, but never sends any data. (This is called a SYN flood.) What happens to your implementation if it were attacked

in this way? How might you have designed the initial handshake protocol (or the protocol implementation) differently to be more robust to this attack?

What will happen is that the node being attacked will continuously open new sockets to handle each SYN request. Very quickly, the node will run out of sockets to use, and thus all communication with the node will be exhausted, effectively shutting out the node from the network. One possible way to combat this is to implement a relatively quick SYN timeout. After approximately 2 RTT + Safety, the socket should timeout and close.

4: What happens in your implementation when a sender transfers data but never closes a connection? (This is called a FIN attack.) How might we design the protocol differently to better handle this case?

In this case, the socket is never closed, and is thus open to receiving more data from various sources (if they know the socket port and address). A simple way to combat this is after the node finishes reading data, it immediately closes the socket regardless of what the sender does.