UCMerced Fall 2024 - CSE160 - Computer Networks
Project 3 - Link State Routing
Project Team: CJ Cox, Aaron Nam
Professor Cerpa

Design Process & Decisions

In Project 3, For our design process, we created a separate data structure for “tcp_pack” by making its own TCP header along with the socket header, it was a good decision to organize TCP packets and the data we utilize later. We used the socket structures given to us for specifically handling the socket states, socket_addr_t, and socket_store_t. We decided on modularizing most functions to make the stages of setup/teardown easier to see and made protocols for handshake and handling the sendPacks for SYN, SYN-ACK, and ACK. Using our design from Project 2 of the Link state, we utilize “Linkstate.send()”, so that when a socket is used it would have all the information it needs to know in its topology.

In our design decision for creating “tcpPacket.h”’ We created a typedef struct that will take in the data sequence that the sender sends within the window along with the data being transferred. This ensured that the TCP packets were encapsulated with all the important information and labeled with a TCP flag to determine the stages for the three-way handshake.

Once the socket states have been initialized and the connection is set to established. The next procedure was sending data. We created two functions “sendData” which took in the socket file descriptor and the other was “receivedData,” which took in the socket, data, and seq. It would utilize the sliding window we integrated and utilize the socket_store_t functions for the sender and receiver portion. Finally, when the data is fully processed, the Transport.release function would reset all the attributes of the socket.

Discussion Questions

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

Choosing a random value is better. While it does increase complexity a little, it’s much better for security, as it makes it much more difficult for any interceptors to be able to read the message being sent.

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

The buffer size should be the same size as the maximum the window size can be, such that the buffer can capture all the data from one window's worth of packages. Then it can read from the buffer, send an ACK, and start again from an empty buffer.

3. Our connection setup protocol is vulnerable to the following attack. The attacker sends a large number of connection requests (SYN) packets to a particular node but never sends any data. (This is called a SYN flood.) What would happen 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?
If our implementation were attacked this way, all of the sockets would get used up and the node wouldn't be able to receive any more SYNs. If we were to redesign our handshake protocol to be more robust to this attack, we would make it so that a node can only receive one SYN per interval of time from any one source. We could also make it so that we add each SYN to a buffer, and they only get a socket once we finish the handshake.

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 manage this case?
Our implementation would just keep the socket open indefinitely. To avoid this, we could make it so that the sender must first tell us how much data is going to be sent, and we would add a cap so that they can't send a ludicrous amount of data. We could also make it so that if they don't transmit any packages that contain data that we can add to our buffer (they have to be within the window and not be a repeat), the connection automatically closes after a while.