Written Assignment 4

Adam Terhaerdt

CS-472

1. **DU-proto.c function definitions:**

**Function:** dpinit()

**Description:** This function is responsible for initializing a new Drexel Procotcol connection. It allocates memory for the connection and sets the memory to zero. After, it initializes the connection struct and returns the connection pointer.

**Returns:** The connection pointer created.

**Function:** dpclose(dp_connp dpsession)

**Description:** This function is responsible for closing the Drexel Procotcol connection. This function frees the memory allocated for the connection.

**Returns:** Nothing.

**Function:** dpmaxdgram()

**Description:** This function is responsible for returning the maximum buffer size of the Drexel Procotcol.

**Returns:** The maximum buffer size of the Drexel Protocol.

**Function:** dpServerInit(int port)

**Description:** This function is responsible for initializing the Drexel Procotcol server. It creates a socket and binds it to the given port. It returns errors if the socket creation fails or the binding fails.

**Returns:** The connection pointer created.

**Function:** dpClientInit(char *addr, int port)

**Description:** This function is responsible for initializing the Drexel Procotcol client. It creates a socket and binds it to the given port. It returns errors if the socket creation fails or the binding fails. The `memcpy` function is used to ensure that the inbound address and port are the same as the outbound address and port.

**Returns:** The connection pointer created.

**Function:** dprecv(dp_connp dp, void *buff, int buff_sz)

**Description:** The purpose of this function is to receive data from a connection and place it into a buffer to which the is provided by the caller.

**Returns:** The size of the data payload.

**Function:** dprecvdgram(dp_connp dp, void *buff, int buff_sz)

**Description:** This function is used to implement the receiver side of the protocol and handle incoming messages while maintaining the state. It validates the buffer and receives raw data. After, it extracts and validates the PDU, increments the seqNum, and then sends back data.

**Returns:** The number of bytes received or an error code.

**Function:** dprecvraw(dp_connp dp, void *buff, int buff_sz)

**Description:** This function is used to receive raw data from the network and perform validation. It checks if the inbound socket is initialized. Then it receives data from the socket and there are any errors. It updates the initialized state, prints the PDU information, and then returns the number of bytes received.

**Returns:** The number of bytes received or an error code.

**Function:** dpsend(dp_connp dp, void *sbuff, int sbuff_sz)

**Description:** This function checks that the size of the data does not exceed the maximum size, then passes the data to another function to send it. Finally it returns the result in sndSz.

**Returns:** The result from dpsenddgram.

**Function:** dpsenddgram(dp_connp dp, void *sbuff, int sbuff_sz)

**Description:** This function is used to build the PDU, send the data, update the seqNum, and wait for acknowledgement that the data was received correctly.

**Returns:** The number of bytes send excluding the PDU header,

**Function:** dpsendraw(dp_connp dp, void *sbuff, int sbuff_sz)

**Description:** This function is used to give direct access to the socket API to send data. It checks if the socket address was initialized, then sends the data to the socket, and returns the number of bytes sent.

**Returns:** The number of bytes sent or an error.

**Function:** dplisten(dp_connp dp)

**Description:** This function is used to wait for and accept an incoming connection request from the client. Once it receives a request, it accepts it, and then updates the connection state.

**Returns:** True to show successful connection.

**Function:** dpconnect(dp_connp dp)

**Description:** This function is used to establish a connection with the server. It sends a connection request to the server, waits for the server to respond, and then updates the connection state.

**Returns:** True to show successful connection.

**Function:** dpdisconnect(dp_connp dp)

**Description:** This function is used to initiate the closing of an established connection. It implements graceful termination of the DP connection. Sends a request to close connection, waits for response, and then disconnects.

**Returns:** DP_CONNECTION_CLOSED

**Function:** dp_prepare_send(dp_pdu *pdu_ptr, void *buff, int buff_sz)

**Description:** This function is used to help contract a formatted message buffer by copying the PDU header into the buffer.

**Returns:** A pointer to after the PDU header.

2. I personally think that the three-layer design is a good approach for the Drexel Protocol implementation. This is because it follows the principle of separation of concerns and provides boundaries for functions. The first layer, or the top layer, dpsend(), is the layer in which the application would interact with. It performs basic validation and then sends it to the middle layer. The top layer is used as an abstraction. Then the middle layer, dpsenddgram(), manages the data structure of the PDU without directly interfacing with the socket API. It handles protocol specific logic, validation, and the data structure without touching the implementation of the socket API. The third layer, dpsendraw(), is the lowest level level. This function directly interfaces with the socket API and performs little validation. This layer actually handles sending the data. Again, I think this is a good design because it abstracts details to lower levels and each function has its clear responsibility.

3. Sequence numbers are used in du-proto to keep track of messaging ordering and is also an acknowledgment mechanism to tell the sender that the receiver has processed the data up until a certain point. We update the sequence number of things that must be acknowledged because it ensures reliable delivery, helps maintain flow so the sender knows how much data the receiver has processed, and it helps keep the state synchronized.

4. Yes, I can think of a limitation this approach might provide vs a traditional TCP approach. The key limitation is that it could significantly reduce throughput on high

latency networks. If we have to send and wait for each protocol and our network is experiencing high latency, then it could take much longer than a traditional TCP approach. This approach simplified du-proto's implementation because we did not need to handle out-of-order packets. The approach that du-proto took, ensured that the packets were always in sync between the sender and receiver, so we did not need to do hard calculations to handle out-of-order packets.

5. A difference between setting up and managing UDP and TCP sockets is how the connection is set up. In a TCP connection, it requires an explicit connection establishment, but in a UDP connection, it requires no formal connection establishment. TCP connections are a continuous stream while UDP sends messages with boundaries attached. TCP has a higher overhead with connection management but has more mechanisms in place to ensure it is more reliable. UDP has a lower overhead, as it does not require connection setup.