1. 1. dpinit(): initializes and returns a dp_connection struct with zeroed values to provide a default dp_connp value to work with

2. dpclose(dp_connp dpsession): frees the memory allocated to dpsession struct, removing use of the provided session's struct.

3. dpmaxdgram(): returns the max size a datagram can have in the drexel transport protocol

4. dpServerInit(int port): creates a UDP socket to listen for requests from, prepares the necessary server information and returns the dp_connp structure when all is initialized correctly, establishing the server.

5. dpClientInit(char *addr, int port): Creates udp socket client (not bound to a port), and provides necessary details to dp_connp structure to be able to connect to a server.

6. dprecv(dp_connp dp, void *buff, int buff_sz): Takes data received, interprets it as a pdu and extracts the data sent to the buff, returning the number of bytes of data received from the pdu payload

7. dprecvdgram(dp_connp dp, void *buff, int buff_sz): Takes bytes received and interprets as pdu and sends back acknowledge or closes connecton

8. dprecvraw(dp_connp dp, void *buff, int buff_sz): Verifies incoming connection struct is initialized, received bytes from sender UDP socket, and marks the sender's address as verified after receiving the data. Returns number of bytes received for validation.

9. dpsend(dp_connp dp, void *sbuff, int sbuff_sz): sends datagram to receiver and returns the amount of bytes sent.

10. dpsenddgram(dp_connp dp, void *sbuff, int sbuff_sz): builds protocol pdu with required information and sends pdu with the datagram payload to the receiver. Returns number of bytes sent.

11. dpsendraw(dp_connp dp, void *sbuff, int sbuff_sz): sends built pdu and datagram payload to the receiver over udp socket and returns number of bytes sent

12. dplisten(dp_connp dp): listens for connection and receives data from sender, upon successful receive, sends acknowledgement to the sender. And marks connection to sender as established

13. dpconnect(dp_connp dp): Sends connection request to receiver, if successful send, receives data back from receiver and if receive and message type looks good, mark connection as established.

14. dpdisconnect(dp_connp dp): Send disconnect message to receiver, receives acknowledgement and closes socket.

15. dp_prepare_send(dp_pdu *pdu_ptr, void *buff, int buff_sz): clears buffer with zeros and copies what is in marks buffer as what pdu_ptr points to, returns pointer to where standard pdu ends and payload can begin

16. print_out_pdu(dp_pdu *pdu): prints out contents of pdu to be sent

17. print_in_pdu(dp_pdu *pdu): prints out contents of pdu received

18. print_pdu_details(dp_pdu *pdu): prints out inner contents of any pdu (in or out)

19. pdu_msg_to_string(dp_pdu *pdu): print the type of a specific message based on mtype attribute of pdu

20. dprand(int threshold): generates random number given a threshold number and if the random number is less than threshold, return true, if greater than threshold, return false.

2. This is useful design because it helps abstract certain low level details that may not be needed by a developer using the drexel university protocol as well as separating out how the protocol works vs standard use case. For example dpsend is very easy to understand for someone using the function, dpsenddgram is more protocol specific and dpsendraw is the basic low level socket logic for getting the data across to the receiver. This effectively separates concerns/responsibilities despite resulting in the same outcome of sending data.
3. seqNums are used to keep track of which request is being handled and what is occurring. We update the sequence number for things that must be acknowledged to maintain correct ordering and keeping a timeline of events, this allows some sense of verification of what we are receiving.
4. This is limited because if you send a lot of data at the once, it is not like TCP that is this huge pipe/stream of bytes that can be ACK'd all at once while du-proto waits before sending any message to be ACK'd making performance a little limiting even compared to TCP with all of its ACKs. This also simplifies du-proto because we aren't worried about if we can send a message, if we have an acknowledgement we know we are ready to send a new message because the previous is acknowledged.
5. One of the main differences when setting up a UDP socket is the use of SOCK_DGRAM for datagram rather than SOCK_STREAM that TCP uses. When managing, you don't use an accept call to listen for connections in UDP, you just recvfrom. You also use sendto and recvfrom instead of send and recv. This is because UDP's connectionless architecture doesn't require acknowledgements, it is send and forget based.