Implementation of a microTCP protocol

Names:
Gavriela Koutsikou
4917

Nikoletta Arvaniti
4844

Course:
Project Networking - CS335a

This project is all about creating a simplified version of the Transmission Control Protocol(TCP) called microTCP. In the first phase we set up basic functions like accept, connect and shutdown in order to successfully complete the 3 way handshake between the server and the client. In this phase we are improving these functions based on the additional needs of the protocol, creating two new functions called microTCP_send and microTCP_recv and adding important features like flow control (to make sure the data does not overwhelm the receiver), congestion control (managing traffic in the network), and retransmissions (resending data if something goes wrong). To test all of these functions, we are using the bandwidth test. This report will guide you through how we designed and implemented microTCP, its key features and how it performed during our tests.

microtcp.c

Starting with the implementation of the header file, in the first 70 lines of the code we define functions for creating and binding a microTCP socket. The ***microtcp_socket*** function initializes a microTCP socket by opening a standard socket and setting up a corresponding structure. The ***microtcp_bind*** function associates the socket with a specified address, updating global variables and the socket's state. It performs input validation and provides error messages with appropriate exits in case of failure. Global variables "cl" and "sr" store client and server addresses, while "cl_len" and "sr_len" represent the lengths. Overall, these functions establish the initial configuration of the microTCP socket and enable it to be bound to a specific address for communication.

In the next lines of code is the implementation of the ***microtcp_connect*** function, which manages the process of establishing a connection between a client and a server. It begins by initializing a random sequence number for each run and setting the destination address for the client's socket. A message structure (message_t) is dynamically allocated to facilitate communication. The function then creates a SYN message with the created random sequence number, sends it to the server, and waits for a SYN ACK response. If successful, it updates the client's window and sends an ACK message to complete the three way handshake. The function transitions the socket's state to "ESTABLISHED" and allocates memory for the receive buffer, initializing window-relating values.

The ***microtcp_accept*** function manages the server-side process of accepting a connection request from a client. It begins by initializing random values for sequence numbers and allocating memory for a message structure (message_t). The function waits for a SYN message from the client, verifies its control flags, and responds with a SYN ACK message. It then waits for the final ACK message from the client to complete the three way handshake. Upon successful completion, the server updates its socket state to "ESTABLISHED" , and allocates memory for the receive buffer as well.

The ***microtcp_shutdown*** function handles the termination of a microTCP connection, adapting its behavior based on whether it is called by the server or the client. If the socket is in the "CLOSING_BY_PEER" state, meaning that the server called the function, indicating it has received a FIN ACK from the client during the ***microtcp_rcv*** process, the function proceeds to send an ACK back to the client, acknowledging the termination request. It then sends a FIN ACK to initiate the termination process, waits for the client's final ACK, and transitions to "CLOSED" state.

On the other hand, when the client calls the shutdown, the function sends a FIN ACK to the server, indicating its intention to terminate. It then waits for the server's ACK, updates sequence and ack numbers, and transitions to "CLOSING_BY_HOST" state. The client continues to wait for the server's FIN ACK, acknowledging the termination, and sends a final ACK to complete the process, moving to the "CLOSED" state.

The ***microtcp_send*** function is responsible for transmitting data from the sender (client or server) to the receiver in the microTCP protocol. The function divides the data into manageable chunks and sends them individually to the receiver. It utilizes various parameters and mechanisms for flow control, congestion control, and error handling. Now let's get into the details. At first, the function initializes essential variables and data structures, this includes the "sendmssg" structure, which represents the message to be sent, a checksum variable for CRC32 checksum calculation, and buffers such as "data_per_chunk" and "temp" to manage data during the transmission process. Additionally, the function configures a timeout for receiving acknowledgments using the "setsockopt" function.

The core of this function resides inside the data transmission loop. Within this loop, the data to be sent are divided into chunks. The size of each chunk is determined by the minimum of the current window size, the congestion window, and the remaining data to be sent. For each chunk, a message header is constructed, including sequence numbers, ack numbers, window sizes(size of receive buffer - how many stuff are inside the receive buffer) and data lengths(bytes to be sent). A CRC32 checksum is calculated for each packet to ensure data reliability.

The function sends each chunk individually using the "sendto" function. With each successful transmission the function updates various socket values, such as "bytes_sent"(adding the bytes we sent to the total), "packets_sent"(incremented by one), "seq_number"(incremented by the number of the bytes we sent) and "curr_win_size"(also incremented by the number of the bytes we sent). The cwnd is also influenced by the successful transmission, if it is still in the slow start phase, it is increased by the size of the MICROTCP_MSS. Once the cwnd surpasses the slow start threshold, the function transitions to congestion avoidance making the appropriate calculations for the cwnd.

The loop continues until all the data to be sent are processed. In cases where the last chunk of data is not a complete multiple of the MSS, we add one more chunk and send the remaining data.

After the data transmission loop, the function is ready to receive the acknowledgments. It waits for ACKs to arrive within a specified timeout period using the "recvfrom" function. If an ACK is not received within the timeout, adjustments are triggered at the cwnd and threshold, and retransmission of packets starting from the one that triggered the timeout.

In situations where three duplicate ACKs are received, indicating potential packet loss, the function initiates fast recovery. This involves adjusting the slow start threshold and congestion window, followed by the retransmission of packets starting from the one associated with the triple duplicate ACK. The handling of ACKs also accounts for scenarios where the receiver's window size is zero. In such cases, the function sends special packets with zero payload to prevent congestion at the receiver's end until a non-zero window size packet is received.

Retransmission logic: whenever there is a need for retransmission, we retransmit all the packets that were not ACKed starting from the packet with sequence number the last ACK number we got. In order to do that we use the array called *chunk_seq_number,* which keeps track of the sequence numbers of each chunk, and the array called *bytes_per_chunk* that keeps the number of bytes of each chunk. We send the remaining packages inside a for loop, starting from the number of chunk that caused the need for retransmission and we decrease by one the outside for-counter in order to receive the remaining ACKs.

The ***microtcp_recv*** function is responsible for receiving data on the receiver's end. The function begins by configuring a receive timeout, allowing it to handle cases where messages are not received within a specified time frame. Upon receiving a message, the function checks for potential errors using the CRC32 checksum (initializing two new variables to keep the return value of the checksum function and the checksum inside the received message, also we set to zero the checksum inside the received message), and if errors occur, duplicate ACKs are sent to prompt retransmission. In case of a correctly received package, the function updates socket values, such as bytes_received and packets received.

The function further checks for specific control flags in the received message. If a FIN ACK flag is detected, indicating the end of the communication, the socket transitions to CLOSING_BY_PEER state. Otherwise, the function ensures the correct sequencing of received data and sends an ACK back to the sender. If a sequence number is unexpected, a duplicate ACK is also sent to signal retransmission.