Data Structures in msocket.h:

`struct message_header_send`:

```
struct message_header_send{
    int is_ack; // 1 for ack 0 for data
    int number; // sequence number

};
```

- Purpose: Represents the header of a message to be sent.

`struct message_send`:

```
struct message_send{
    char data[1024];
    struct message_header_send header;
};
```

- Purpose: Represents a message to be sent, including its data and header.

`struct message_receive`:

```
struct message_receive{
    char data[1024];
    int num; // sequence number of message received
};
```

- Purpose: Represents a received message.

`struct swnd` (Sender Window):

```
struct swnd{
        int window_size; // Window size
       time_t time[10]; //saves the time of the message sent

       struct message_send send_messages[10]; // Sequence numbers of
messages sent but not acknowledged
       int next_sequence_number; // next sequence number expected
       int index_to_write; // pointer in the buffer array to write the next
message
    }; // Sender window
```

- Purpose: Represents the sender window for managing sent messages.

`struct rwnd` (Receiver Window):

```
struct rwnd{
        int window_size; // Window size
        struct message_receive receive_messages[5]; // Sequence numbers of messages received but not acknowledgement not sent

        int next_sequence_number;//sequence number of the first message in buffer
        int nospace; // flag to see if there is space in the buffer
    };
```

- Purpose: Represents the receiver window for managing received messages.

`struct MTPSocketInfo`:

```
struct MTPSocketInfo {
    int is_allocated; // Flag indicating if the MTP socket is free or allocated
    int process_id; // Process ID for the process that created the MTP socket
    int udp_socket_id; // UDP socket ID
    char other_end_ip[16]; // IP address of the other end of the MTP socket
    unsigned short other_end_port; // Port of the other end of the MTP socket
    struct swnd senders_window;
    struct rwnd receivers_window;
 };
```

`struct SOCK_INFO`:

```
struct SOCK_INFO {
    int sock_id;
    char IP[50];
    unsigned short port;
    int err_;
};
```

Functions in msocket.c:

M_socket:

```
int m_socket(int domain, int type, int protocol);
```

Function to create a new socket. Semaphores are used to synchronize the creation of sockets.The sock info values are set to 0. This function signals the sem1 semaphore which is waiting in initmsocket. After creation of socket, initmsocket signals sem2 semaphore where msocket is waiting.

M_bind:

```
int m_bind(int sockfd,char source_ip[50],unsigned short source_port,char
dest_ip[50],unsigned short dest_port);
```

Function to bind the socket to the given IP and port. Semaphores are used to synchronize. The sockinfo sockid is set to sockfd, ip is set to source_ip and port is set to source_port. This function signals the sem1 semaphore which is waiting in initmsocket. After creation of socket, initmsocket signals sem2 semaphore where msocket is waiting.

M_sendto:

```
int m_sendto(int sockfd, const void* data, int len, int flags, const struct
sockaddr *servaddr, socklen_t addrlen );
```

Create a function that places a message into the sender buffer only if the buffer is currently empty. If the destination port and address are not specified, return an ENOTBOUND error. If the sender buffer is already full, return an ENOBUFS error. Otherwise, return the length of the message successfully written to the sender buffer. This function operates in a non-blocking manner.

M_recvfrom:

```
int m_recvfrom(int sockfd, void *buffer, int len, int flags, struct sockaddr
*src_addr, socklen_t *addrlen);
```

Function to retrieve the message from the receive buffer. If there is no message present in the buffer then the error number is set to ENOMSG. The first message of the buffer is copied to the (void*) buffer passed in the argument. The first message is removed from the receivers window.

dropMessage:

```
int dropMessage(float p)
```

This function drops the message with a probability p. This function is called in the receivers thread.

- Purpose: Simulates dropping a message with a certain probability.

`M_close`:

```
int m_close(int sockfd)
```

This function sets the is_allocated field in shared memory to -1. Then the garbage collector when it checks that a particular socket has is_allocated set to -1 it closes the socket.

Functions in initmsocket.c:

`signal_handler`:

- Purpose: Handles the signal generated by pressing Ctrl+C. It detaches shared memory segments, destroys semaphores, and exits the program gracefully.

`get_ip_port`:

- Parameters: `client_addr` (pointer to `struct sockaddr_in`), `ip_str` (char array to store IP address), `ip_str_len` (size of `ip_str`), `port` (pointer to store port number).
- Purpose: Extracts IP address and port number from a given `struct sockaddr_in`.

`send_ack`:

- Parameters: `i` (index indicating MTP socket,shared Memory pointer).
- Purpose: Sends an acknowledgment message for the given MTP socket index.

`receiver_thread`:

- Parameters: `arg` (unused).
- Purpose: Implements the behavior of the receiver thread. It listens for incoming messages, processes them, and sends acknowledgments accordingly.

`sender_thread`:

- Parameters: `arg` (unused).

- Purpose: Implements the behaviour of the sender thread. It sends messages stored in the sender window at specific intervals after timeout of T seconds

`garbage_collector`:

- Parameters: `arg` (unused).
- Purpose: Cleans up inactive MTP sockets, freeing resources associated with them.

`main`:

- Purpose: The main function of the program. It initialises shared memory, semaphores, and SOCK_INFO structure. Then, it creates receiver and sender threads, and it waits for m_socket() or m_bind() calls.

TABLE : Probability vs Number of Transmission per message

Probability	Ratio
0.05	1.330097
0.10	1.485437
0.15	1.441176
0.20	1.685437
0.25	2.067961
0.30	2.288461
0.35	2.474227
0.40	2.814286
0.45	2.789020
0.50	3.249596