COMP 7005 Project Design

Sami Roudgarian, A01294122

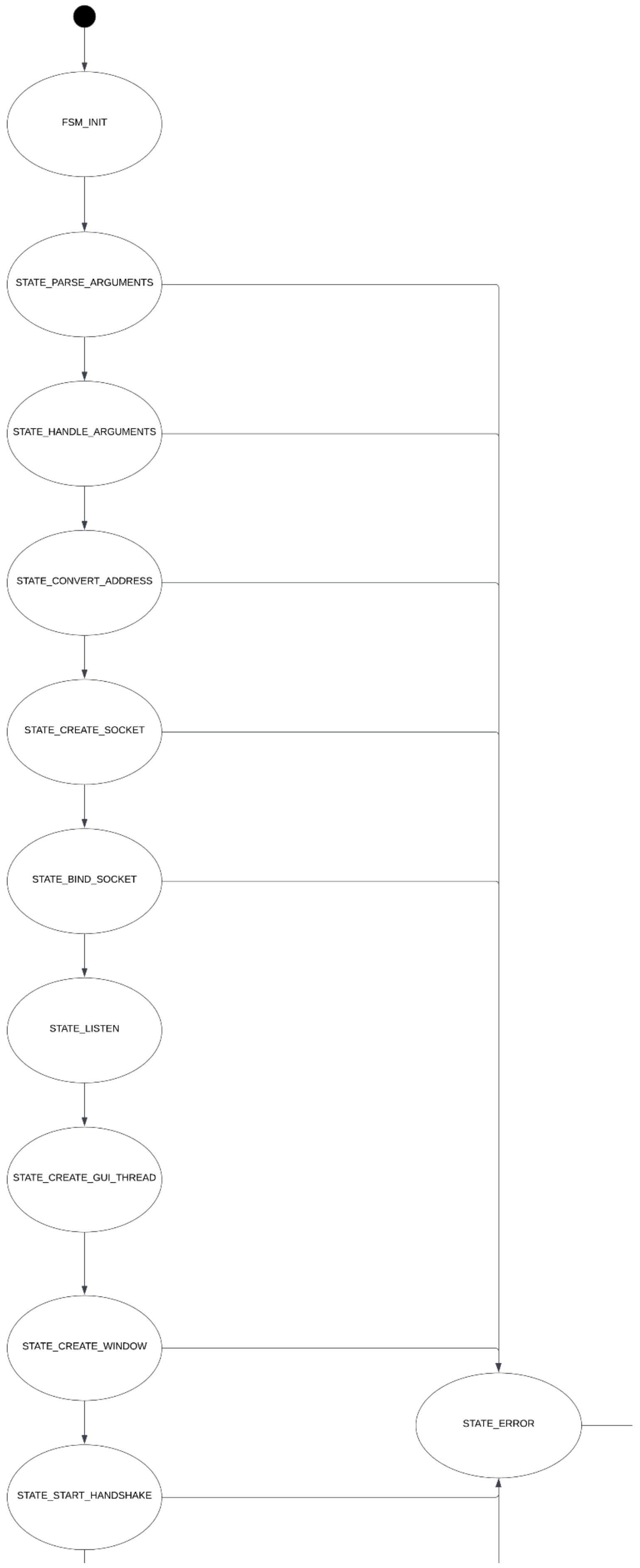
Harmanbir Dhillon, A00994245

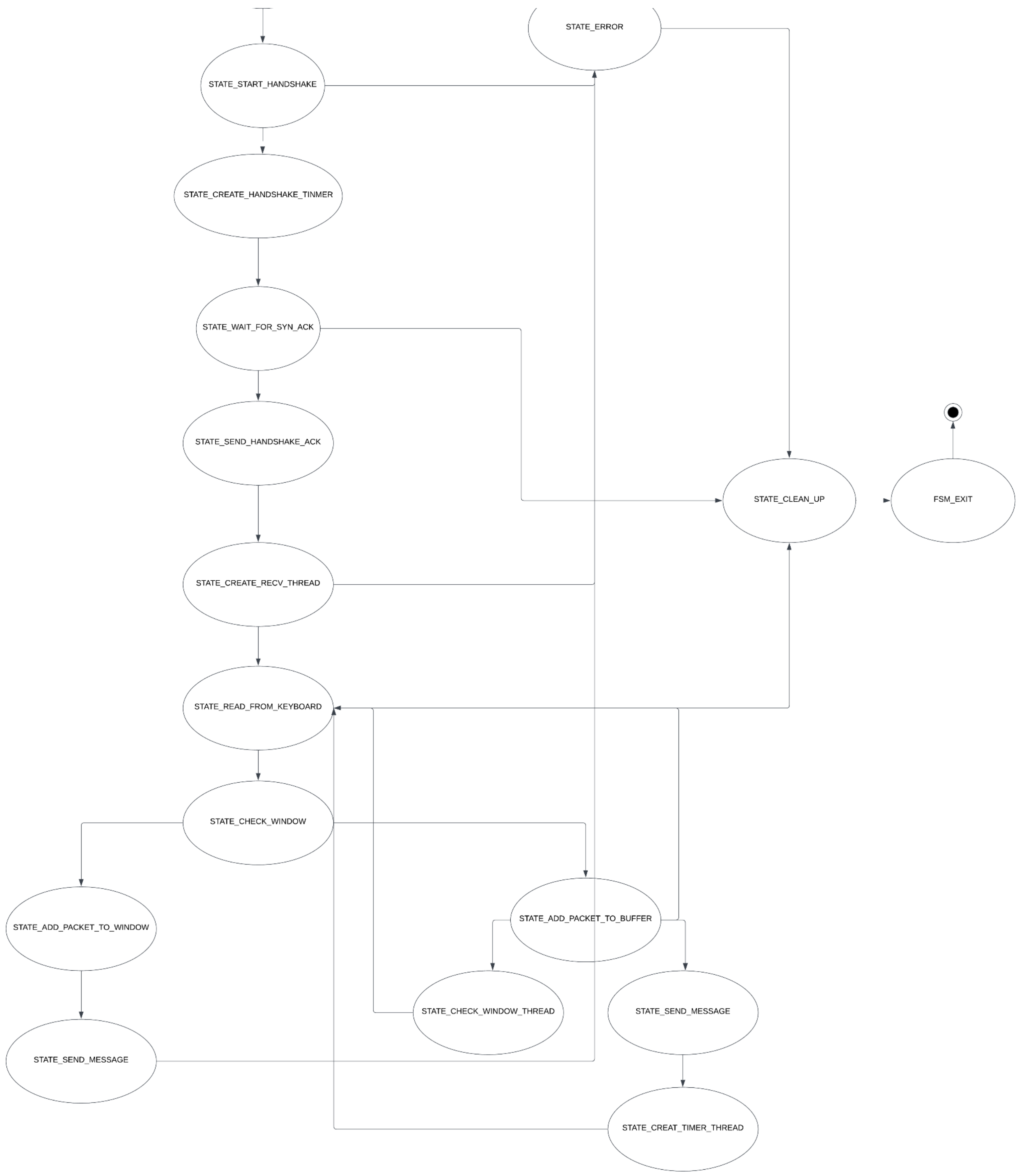
December 1, 2023

Client Main Table

From	**To**	**Handler**
FSM_INIT	STATE_PARSE_ARGUMENTS	parse_arguments_handler
STATE_PARSE_ARGUMENTS	STATE_HANDLE_ARGUMENTS	handle_arguments_handler
STATE_HANDLE_ARGUMENTS	STATE_CONVERT_ADDRESS	convert_address_handler
STATE_CONVERT_ADDRESS	STATE_CREATE_SOCKET	create_socket_handler
STATE_CREATE_SOCKET	STATE_BIND_SOCKET	bind_socket_handler
STATE_BIND_SOCKET	STATE_LISTEN	listen_handler
STATE_LISTEN	STATE_CREATE_GUI_THREAD	create_gui_thread_handler
STATE_CREATE_GUI_THREAD	STATE_CREATE_WINDOW	create_window_handler
STATE_CREATE_WINDOW	STATE_START_HANDSHAKE	start_handshake_handler
STATE_START_HANDSHAKE	STATE_CREATE_HANDSHAKE_TIMER	start_handshake_handler
STATE_CREATE_HANDSHAKE_TIMER	STATE_WAIT_FOR_SYN_ACK	wait_for_syn_ack_handler
STATE_WAIT_FOR_SYN_ACK	STATE_SEND_HANDSHAKE_ACK	send_handshake_ack_handler
STATE_WAIT_FOR_SYN_ACK	STATE_CLEANUP	cleanup_handler
STATE_SEND_HANDSHAKE_ACK	STATE_CREATE_RECV_THREAD	create_recv_thread_handler
STATE_CREATE_RECV_THREAD	STATE_READ_FROM_KEYBOARD	read_from_keyboard_handler
STATE_READ_FROM_KEYBOARD	STATE_CHECK_WINDOW,	check_window_handle
STATE_CHECK_WINDOW	STATE_ADD_PACKET_TO_WIND	add_packet_to_window_handler
STATE_CHECK_WINDOW	STATE_ADD_PACKET_TO_BUFFER	add_packet_to_buffer_handler
STATE_ADD_PACKET_TO_BUFFER	STATE_READ_FROM_KEYBOARD	read_from_keyboard_handler
STATE_ADD_PACKET_TO_BUFFER	STATE_CHECK_WINDOW_THREAD	check_window_thread_handle
STATE_ADD_PACKET_TO_WINDOW	STATE_SEND_MESSAGE	send_message_handler
STATE_CHECK_WINDOW_THREAD	STATE_READ_FROM_KEYBOARD	read_from_keyboard_handler
STATE_SEND_MESSAGE	STATE_CREATE_TIMER_THREAD	create_timer_thread_handler
STATE_CREATE_TIMER_THREAD	STATE_READ_FROM_KEYBOARD	read_from_keyboard_handler
STATE_READ_FROM_KEYBOARD	STATE_CLEANUP	cleanup_handler

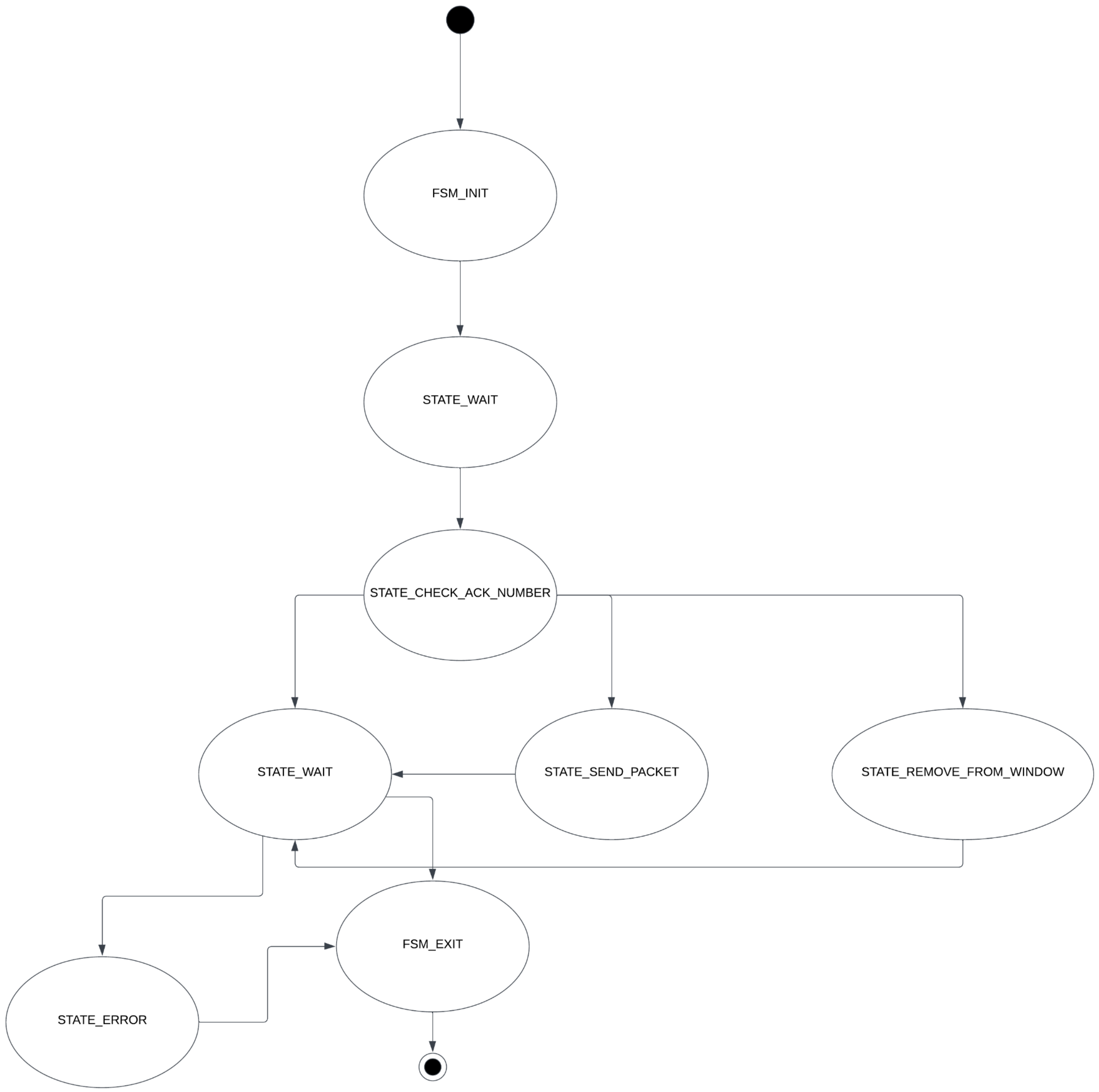
From	**To**	**Handler**
STATE_ERROR	STATE_CLEANUP	cleanup_handler
STATE_PARSE_ARGUMENTS,	STATE_ERROR	error_handler
STATE_HANDLE_ARGUMENTS	STATE_ERROR	error_handler
STATE_CONVERT_ADDRESS	STATE_ERROR	error_handler
STATE_CREATE_SOCKET	STATE_ERROR	error_handler
STATE_BIND_SOCKET	STATE_ERROR	error_handler
STATE_CREATE_WINDOW	STATE_ERROR	error_handler
STATE_CREATE_RECV_THREAD	STATE_ERROR	error_handler
STATE_START_HANDSHAKE	STATE_ERROR	error_handler
STATE_SEND_MESSAGE	STATE_ERROR	error_handler
STATE_CLEANUP	STATE_ERROR	error_handler
STATE_CLEANUP	FSM_EXIT	-


FSM_INIT
STATE_PARSE_ARGUMENTS
STATE_HANDLE_ARGUMENTS
STATE_CONVERT_ADDRESS
STATE_CREATE_SOCKET
STATE_BIND_SOCKET
STATE_LISTEN
STATE_CREATE_GUI_THREAD
STATE_CREATE_WINDOW
STATE_ERROR
STATE_START_HANDSHAKE


STATE_ERROR
STATE_START_HANDSHAKE
STATE_CREATE_HANDSHAKE_TINMER
STATE_WAIT_FOR_SYN_ACK
STATE_SEND_HANDSHAKE_ACK
STATE_CLEAN_UP
FSM_EXIT
STATE_CREATE_RECV_THREAD
STATE_READ_FROM_KEYBOARD
STATE_CHECK_WINDOW
STATE_ADD_PACKET_TO_WINDOW
STATE_ADD_PACKET_TO_BUFFER
STATE_CHECK_WINDOW_THREAD
STATE_SEND_MESSAGE
STATE_SEND_MESSAGE
STATE_CREAT_TIMER_THREAD

Client Receive Table

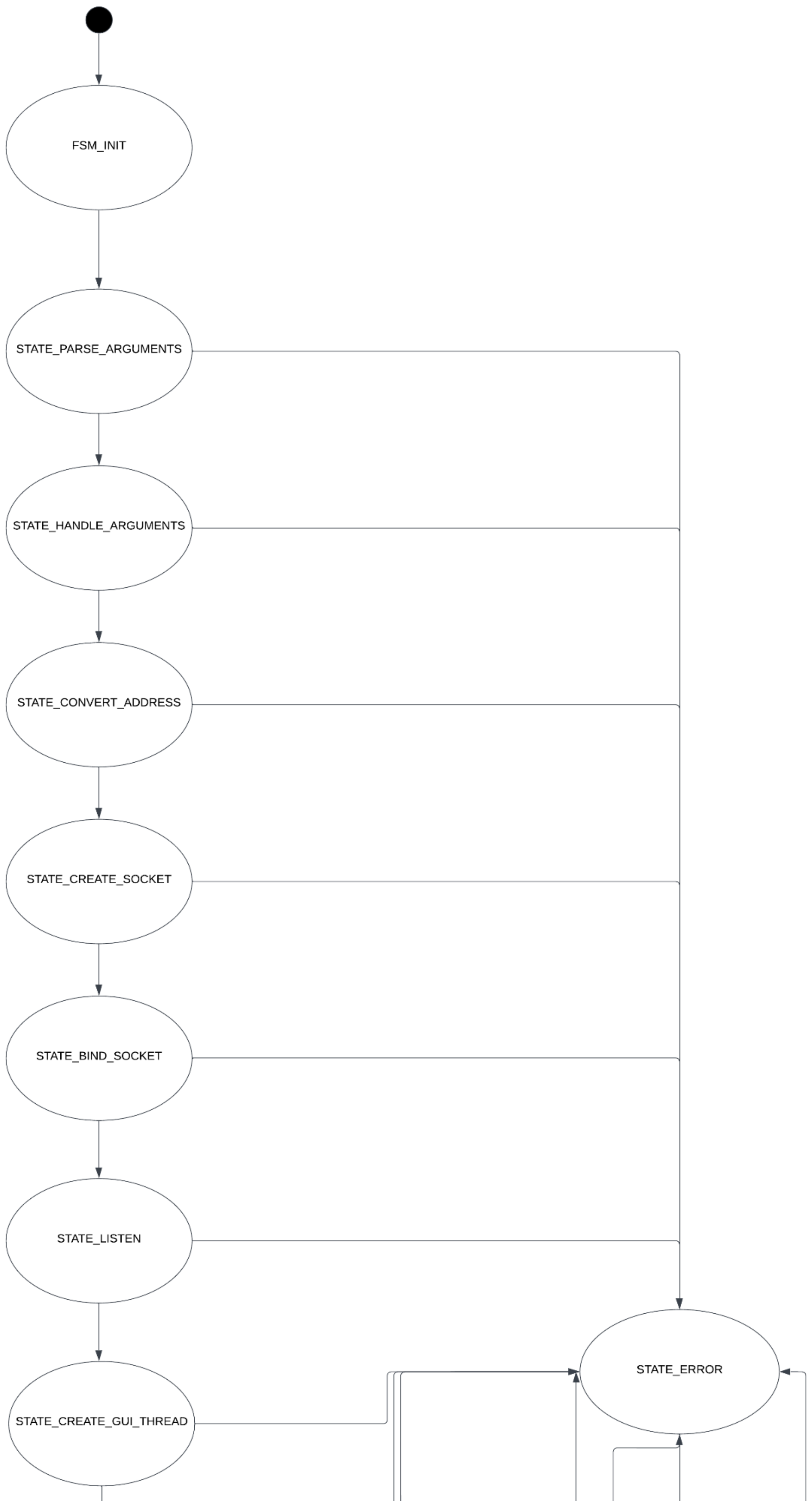
From	To	Handler
FSM_INIT	STATE_WAIT	wait_handler
STATE_WAIT	STATE_CHECK_ACK_NUMBER,	check_ack_number_handler
STATE_CHECK_ACK_NUMBER	STATE_REMOVE_FROM_WINDOW,	remove_packet_from_window_handler
STATE_CHECK_ACK_NUMBER	STATE_SEND_PACKET	send_packet_handler
STATE_CHECK_ACK_NUMBER	STATE_WAIT	wait_handler
STATE_REMOVE_FROM_WINDOW	STATE_WAIT	wait_handler
STATE_SEND_PACKET	STATE_WAIT	wait_handler
STATE_WAIT	STATE_ERROR	error_handler
STATE_WAIT	FSM_EXIT	-
STATE_ERROR	FSM_EXIT	-

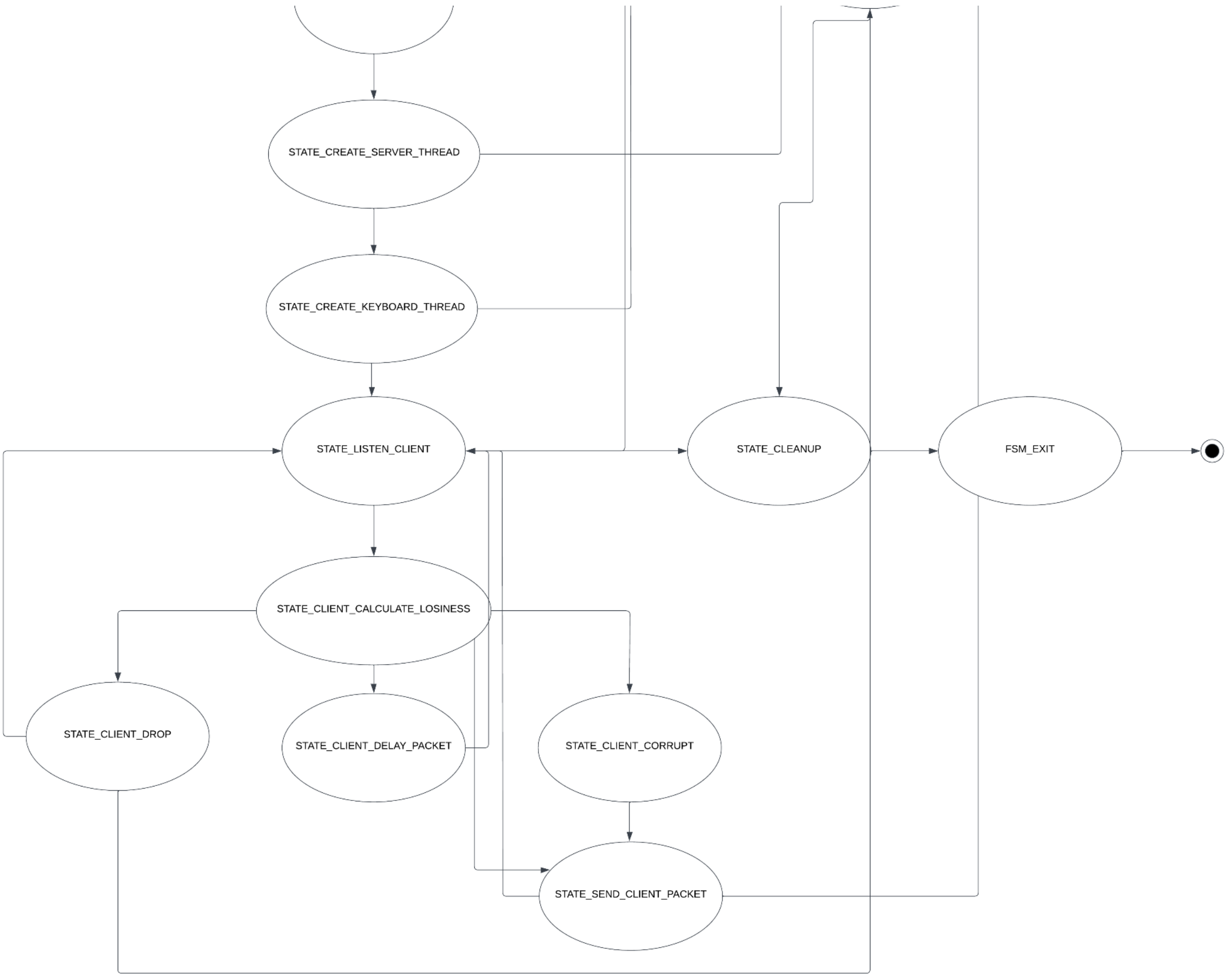

FSM_INIT
STATE_WAIT
STATE_CHECK_ACK_NUMBER
STATE_WAIT
STATE_SEND_PACKET
STATE_REMOVE_FROM_WINDOW
FSM_EXIT
STATE_ERROR

Proxy Main Table

From	To	Handler
FSM_INIT	STATE_PARSE_ARGUMENTS	parse_arguments_handler
STATE_PARSE_ARGUMENTS	STATE_HANDLE_ARGUMENTS	handle_arguments_handler
STATE_HANDLE_ARGUMENTS	STATE_CONVERT_ADDRESS	convert_address_handler
STATE_CONVERT_ADDRESS	STATE_CREATE_SOCKET	create_socket_handler
STATE_CREATE_SOCKET	STATE_BIND_SOCKET	bind_socket_handler
STATE_BIND_SOCKET	STATE_LISTEN	listen_handler
STATE_LISTEN	STATE_CREATE_GUI_THREAD	create_gui_thread_handler
STATE_CREATE_GUI_THREAD	STATE_CREATE_WINDOW	create_window_handler
STATE_CREATE_SERVER_THREAD	STATE_CREATE_KEYBOARD_THREAD	create_keyboard_thread_handler
STATE_CREATE_KEYBOARD_THREAD	STATE_LISTEN_CLIENT,	listen_client_handler
STATE_LISTEN_CLIENT,	STATE_CLIENT_CALCULATE_LOSSINESS	calculate_client_lossiness_handler
STATE_LISTEN_CLIENT	STATE_CLEANUP	cleanup_handler
STATE_CLIENT_CALCULATE_LOSSINESS	STATE_CLIENT_DROP,	client_drop_packet_handler
STATE_CLIENT_CALCULATE_LOSSINESS	STATE_CLIENT_DELAY_PACKET	client_delay_packet_handler
STATE_CLIENT_CALCULATE_LOSSINESS	STATE_CLIENT_CORRUPT	client_corrupt_packet_handler
STATE_CLIENT_CALCULATE_LOSSINESS	STATE_SEND_CLIENT_PACKET	send_client_packet_handler
STATE_CLIENT_DROP	STATE_LISTEN_CLIENT	listen_client_handler
STATE_CLIENT_DELAY_PACKET,	STATE_LISTEN_CLIENT,	listen_client_handler
STATE_CLIENT_CORRUPT	STATE_SEND_CLIENT_PACKET,	send_client_packet_handler
STATE_SEND_CLIENT_PACKET	STATE_LISTEN_CLIENT,	listen_client_handler
STATE_ERROR	STATE_CLEANUP	cleanup_handler
STATE_PARSE_ARGUMENTS,	STATE_ERROR	error_handler

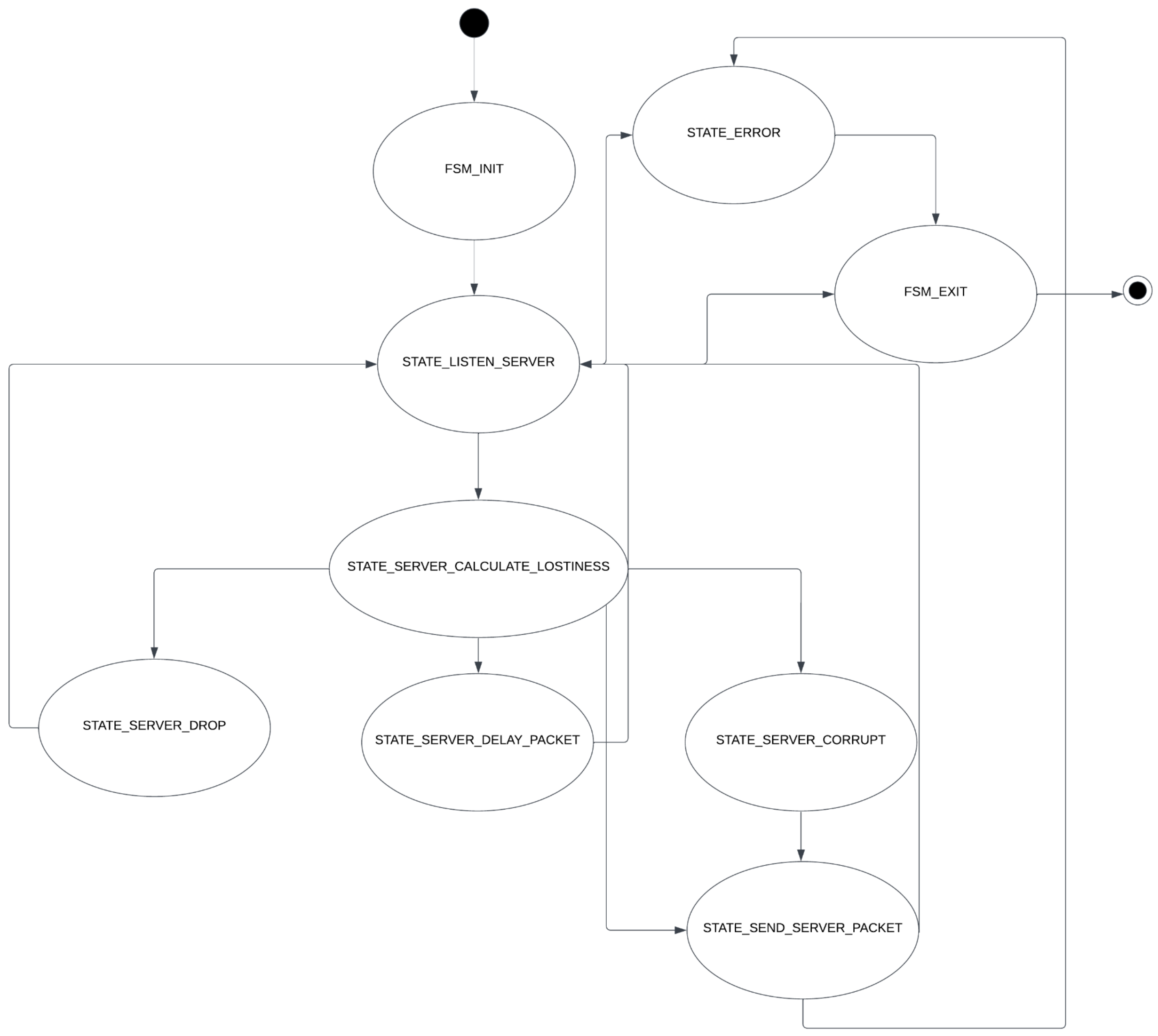
From	To	Handler
STATE_HANDLE_ARGUMENTS	STATE_ERROR	error_handler
STATE_CONVERT_ADDRESS	STATE_ERROR	error_handler
STATE_CREATE_SOCKET	STATE_ERROR	error_handler
STATE_BIND_SOCKET	STATE_ERROR	error_handler
STATE_CREATE_WINDOW	STATE_ERROR	error_handler
STATE_CREATE_SERVER_THREAD	STATE_ERROR	error_handler
STATE_CREATE_KEYBOARD_THREAD	STATE_ERROR	error_handler
STATE_LISTEN_CLIENT	STATE_ERROR	error_handler
STATE_CLIENT_DROP	STATE_ERROR	error_handler
STATE_SEND_CLIENT_PACKET	STATE_ERROR	error_handler
STATE_CLEANUP	STATE_ERROR	error_handler
STATE_CLEANUP	FSM_EXIT	-


FSM_INIT
STATE_PARSE_ARGUMENTS
STATE_HANDLE_ARGUMENTS
STATE_CONVERT_ADDRESS
STATE_CREATE_SOCKET
STATE_BIND_SOCKET
STATE_LISTEN
STATE_CREATE_GUI_THREAD
STATE_ERROR


STATE_CREATE_SERVER_THREAD
STATE_CREATE_KEYBOARD_THREAD
STATE_LISTEN_CLIENT
STATE_CLEANUP
FSM_EXIT
STATE_CLIENT_CALCULATE_LOSINESS
STATE_CLIENT_DROP
STATE_CLIENT_DELAY_PACKET
STATE_CLIENT_CORRUPT
STATE_SEND_CLIENT_PACKET

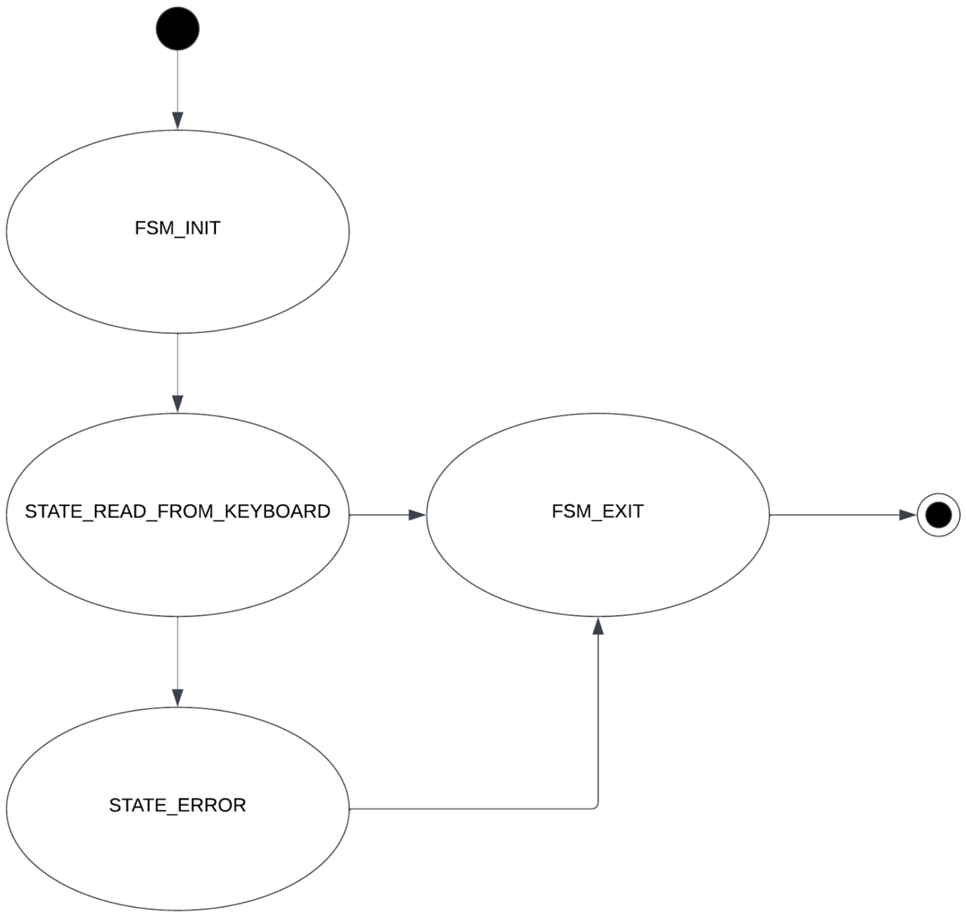
Proxy Server Losiness Table

From	To	Handler
FSM_INIT	STATE_LISTEN_SERVER	listen_server_handler
STATE_LISTEN_SERVER	STATE_SERVER_CALCULATE_LOSSINESS	calculate_server_lossiness_handler
STATE_LISTEN_SERVER	STATE_ERROR	error_handler
STATE_SERVER_CALCULATE_LOSSINESS	STATE_SERVER_DROP	server_drop_packet_handler
STATE_SERVER_CALCULATE_LOSSINESS	STATE_SERVER_DELAY_PACKET	server_delay_packet_handler
STATE_SERVER_CALCULATE_LOSSINESS	STATE_SERVER_CORRUPT	server_corrupt_packet_handler
STATE_SERVER_CALCULATE_LOSSINESS	STATE_SEND_SERVER_PACKET	send_server_packet_handler
STATE_SERVER_DROP	STATE_LISTEN_SERVER	listen_server_handler
STATE_SERVER_DELAY_PACKET	STATE_LISTEN_SERVER	listen_server_handler
STATE_CREATE_SERVER_THREAD	STATE_CREATE_KEYBOARD_THREAD	create_keyboard_thread_handler
STATE_CREATE_KEYBOARD_THREAD	STATE_LISTEN_CLIENT,	listen_client_handler
STATE_SERVER_CORRUPT	STATE_SEND_SERVER_PACKET	send_server_packet_handler
STATE_SEND_SERVER_PACKET	STATE_LISTEN_SERVER	listen_server_handler
STATE_SEND_SERVER_PACKET	STATE_ERROR	error_handler
STATE_LISTEN_SERVER	FSM_EXIT	-
STATE_ERROR	FSM_EXIT	-


FSM_INIT
STATE_ERROR
FSM_EXIT
STATE_LISTEN_SERVER
STATE_SERVER_CALCULATE_LOSTINESS
STATE_SERVER_DROP
STATE_SERVER_DELAY_PACKET
STATE_SERVER_CORRUPT
STATE_SEND_SERVER_PACKET

Proxy Keyboard Table

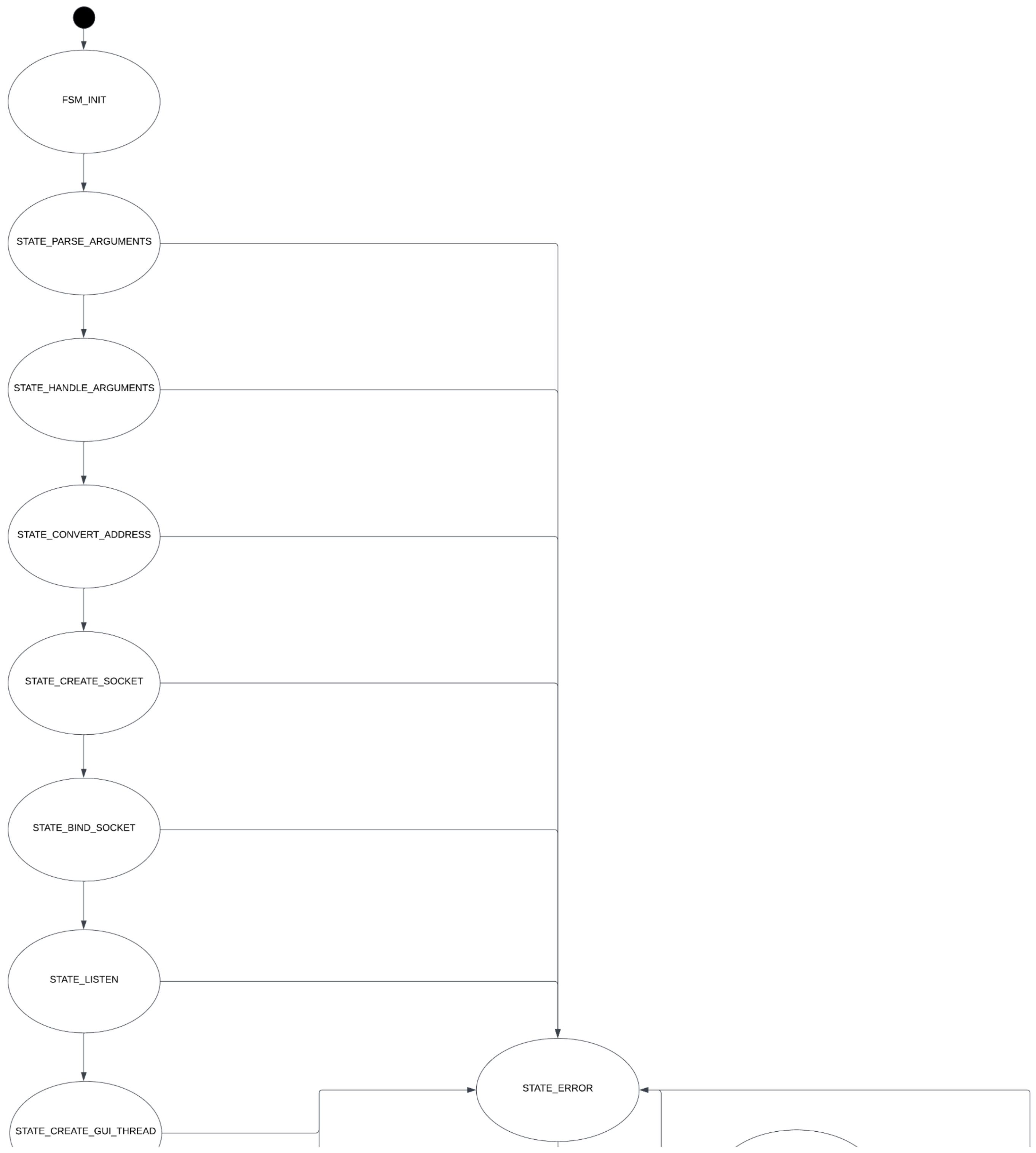
From	**To**	**Handler**
FSM_INIT	STATE_READ_FROM_KEYBOARD	read_from_keyboard_handler
STATE_READ_FROM_KEYBOARD	FSM_EXIT	-
STATE_READ_FROM_KEYBOARD	STATE_ERROR	error_handler
STATE_ERROR	FSM_EXIT	-



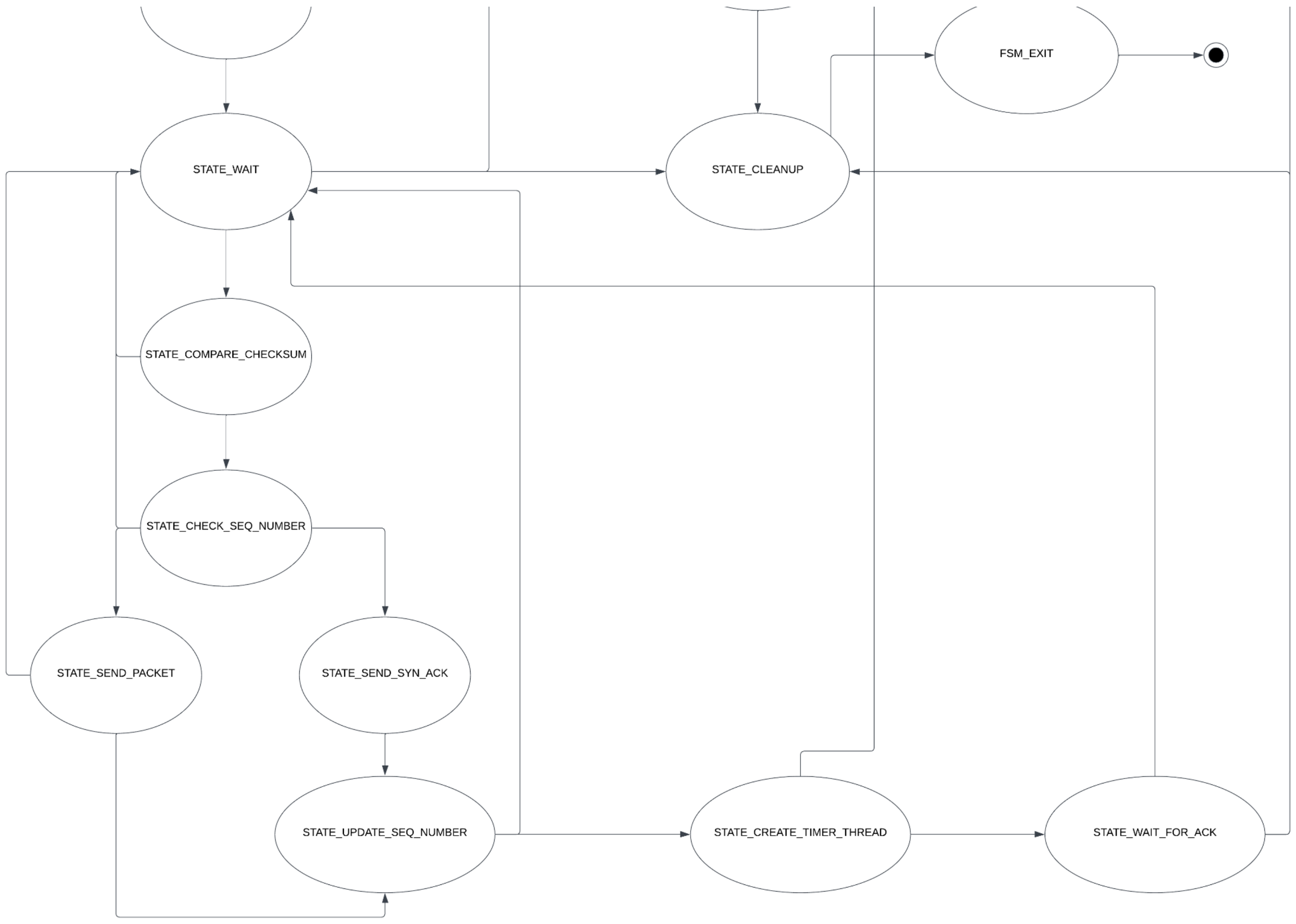
Server Main Table

From	To	Handler
FSM_INIT	STATE_PARSE_ARGUMENTS	parse_arguments_handler
STATE_PARSE_ARGUMENTS	STATE_HANDLE_ARGUMENTS	handle_arguments_handler
STATE_HANDLE_ARGUMENTS	STATE_CONVERT_ADDRESS	convert_address_handler
STATE_CONVERT_ADDRESS	STATE_CREATE_SOCKET	create_socket_handler
STATE_CREATE_SOCKET	STATE_BIND_SOCKET	bind_socket_handler
STATE_BIND_SOCKET	STATE_LISTEN	listen_handler
STATE_LISTEN	STATE_CREATE_GUI_THREAD	create_gui_thread_handler
STATE_CREATE_GUI_THREAD	STATE_CREATE_WINDOW	create_window_handler
STATE_WAIT	STATE_COMPARE_CHECKSUM	compare_checksum_handler
STATE_COMPARE_CHECKSUM	STATE_CHECK_SEQ_NUMBER	check_seq_number_handler
STATE_COMPARE_CHECKSUM	STATE_WAIT	wait_handler
STATE_WAIT	STATE_CLEANUP	cleanup_handler
STATE_CHECK_SEQ_NUMBER	STATE_SEND_PACKET	send_packet_handler
STATE_CHECK_SEQ_NUMBER	STATE_SEND_SYN_ACK	send_syn_ack_handler
STATE_SEND_SYN_ACK	STATE_UPDATE_SEQ_NUMBER	update_seq_num_handler
STATE_CHECK_SEQ_NUMBER	STATE_WAIT	wait_handler
STATE_SEND_PACKET	STATE_UPDATE_SEQ_NUMBER	update_seq_num_handler
STATE_SEND_PACKET	STATE_WAIT	wait_handler
STATE_UPDATE_SEQ_NUMBER	STATE_WAIT	wait_handler
STATE_UPDATE_SEQ_NUMBER	STATE_CREATE_TIMER_THREAD	create_timer_handler
STATE_CREATE_TIMER_THREAD	STATE_WAIT_FOR_ACK	wait_for_ack_handler
STATE_WAIT_FOR_ACK	STATE_WAIT	wait_handler
STATE_WAIT_FOR_ACK	STATE_CLEANUP	cleanup_handler
STATE_ERROR	STATE_CLEANUP	cleanup_handler
STATE_PARSE_ARGUMENTS,	STATE_ERROR	error_handler

From	**To**	**Handler**
STATE_HANDLE_ARGUMENTS	STATE_ERROR	error_handler
STATE_CONVERT_ADDRESS	STATE_ERROR	error_handler
STATE_CREATE_SOCKET	STATE_ERROR	error_handler
STATE_BIND_SOCKET	STATE_ERROR	error_handler
STATE_LISTEN	STATE_ERROR	error_handler
STATE_CREATE_GUI_THREAD	STATE_ERROR	error_handler
STATE_WAIT	STATE_ERROR	error_handler
STATE_CREATE_TIMER_THREAD	STATE_ERROR	error_handler
STATE_WAIT_FOR_ACK	STATE_ERROR	error_handler
STATE_CLEANUP	STATE_ERROR	error_handler
STATE_CLEANUP	FSM_EXIT	-



FSM_INIT
STATE_PARSE_ARGUMENTS
STATE_HANDLE_ARGUMENTS
STATE_CONVERT_ADDRESS
STATE_CREATE_SOCKET
STATE_BIND_SOCKET
STATE_LISTEN
STATE_CREATE_GUI_THREAD
STATE_ERROR



FSM_EXIT
STATE_WAIT
STATE_CLEANUP
STATE_COMPARE_CHECKSUM
STATE_CHECK_SEQ_NUMBER
STATE_SEND_PACKET
STATE_SEND_SYN_ACK
STATE_UPDATE_SEQ_NUMBER
STATE_CREATE_TIMER_THREAD
STATE_WAIT_FOR_ACK

Functions For Client

main

Purpose

Initializes structures and starts the finite state machine (FSM) for network communication.

Parameters

int argc: Count of command-line arguments.

char **argv: Array of command-line argument strings.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE err as fsm_error
DECLARE args as arguments with initial values NULL and 0 for head and is_buffered
DECLARE context as fsm_context with argc, argv, and address of args
DECLARE transitions as array of fsm_transition with predefined states and handlers
CALL fsm_run with address of context, address of err, and additional parameters
RETURN 0
```

parse_arguments_handler

Purpose

Parses command-line arguments in the FSM context.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_HANDLE_ARGUMENTS

Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, descriptive message, and current state
IF CALL parse_arguments with ctx's argc, argv, and args returns non-zero THEN
    RETURN STATE_ERROR
ELSE
    RETURN STATE_HANDLE_ARGUMENTS
ENDIF
```

handle_arguments_handler

Purpose

Processes parsed arguments to set up the application's configuration.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CONVERT_ADDRESS
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, "in handle arguments", "STATE_HANDLE_ARGUMENTS"
IF CALL handle_arguments with argv[0], server_addr, client_addr, server_port_str,
client_port_str,
   address of server_port, address of client_port, window_size from ctx's args returns non-zero
THEN
   RETURN STATE_ERROR
ENDIF
IF CALL create_file with "../client_received_data.csv", address of received_data from ctx's args,
err returns -1 THEN
   RETURN STATE_ERROR
ENDIF
IF CALL create_file with "../client_sent_data.csv", address of sent_data from ctx's args, err
returns -1 THEN
   RETURN STATE_ERROR
ENDIF
RETURN STATE_CONVERT_ADDRESS
```

convert_address_handler

Purpose

Processes parsed arguments to set up the application's configuration.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CREATE_SOCKET
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, descriptive message, and current state
IF CALL convert_address for server address with server_addr, server_addr_struct, server_port from ctx returns non-zero THEN
    RETURN STATE_ERROR
ENDIF
IF CALL convert_address for client address with client_addr, client_addr_struct, client_port from ctx returns non-zero THEN
    RETURN STATE_ERROR
ENDIF
RETURN STATE_CREATE_SOCKET
```

create_socket_handler

Purpose

Processes parsed arguments to set up the application's configuration.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_BIND_SOCKET
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, descriptive message, and current state
SET ctx's sockfd with the result of CALL socket_create with family, type, and protocol from ctx
IF ctx's sockfd is -1 THEN
    RETURN STATE_ERROR
ELSE
    RETURN STATE_BIND_SOCKET
ENDIF
```

bind_socket_handler

Purpose

Binds the created socket to a client address.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CREATE_WINDOW
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, "in bind socket", "STATE_BIND_SOCKET"
IF CALL socket_bind with ctx's sockfd and client_addr_struct returns non-zero THEN
   RETURN STATE_ERROR
ELSE
   RETURN STATE_CREATE_WINDOW
ENDIF
```

create_window_handler

Purpose

Initializes a window for managing packets in network communication.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_START_HANDSHAKE
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, "in create window", "STATE_CREATE_WINDOW"
IF CALL create_window with address of ctx's args window and window_size returns non-zero
THEN
   RETURN STATE_ERROR
ELSE
   RETURN STATE_START_HANDSHAKE
ENDIF
```

start_handshake_handler

Purpose

Begins the handshake process for establishing a connection.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_START_HANDSHAKE
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, "in connect socket", "STATE_START_HANDSHAKE"
IF CALL send_syn_packet with ctx's sockfd, server_addr_struct, and window returns non-zero
THEN
   RETURN STATE_ERROR
ELSE
   RETURN STATE_CREATE_HANDSHAKE_TIMER
ENDIF
```

create_handshake_timer_handler

Purpose

Creates a timer thread for managing handshake timing.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_START_HANDSHAKE
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, "", "STATE_CREATE_HANDSHAKE_TIMER"
INCREMENT ctx's args num_of_threads
REALLOCATE memory for thread_pool in ctx's args based on num_of_threads
IF reallocated thread_pool is NULL THEN
    RETURN STATE_ERROR
ENDIF
CREATE new thread in thread_pool with init_timer_function and ctx
RETURN STATE_WAIT_FOR_SYN_ACK
```

wait_for_syn_ack_handler

Purpose

Waits for SYN-ACK packet during handshake process.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CLEANUP
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
DECLARE result as ssize_t
CALL SET_TRACE with context, "in connect socket", "STATE_WAIT_FOR_SYN_ACK"
WHILE exit_flag is not true DO
   ASSIGN result with CALL receive_packet with ctx's sockfd, window, and temp_packet
   IF result is -1 THEN
      RETURN STATE_ERROR
   ENDIF
   PRINT "Server packet with seq number: ", ctx's temp_packet.hd.seq_number
   IF ctx's temp_packet.hd.flags is SYNACK THEN
      RETURN STATE_SEND_HANDSHAKE_ACK
   ENDIF
END WHILE
RETURN STATE_CLEANUP
```

send_handshake_ack_handler

Purpose

Sends ACK for the handshake process.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CLEANUP
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, "in connect socket", "STATE_SEND_HANDSHAKE_ACK"
CALL read_received_packet with ctx's sockfd, server_addr_struct, window, and temp_packet
RETURN STATE_CREATE_RECV_THREAD
```

create_recv_thread_handler

Purpose

Creates a thread for receiving data.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_READ_FROM_KEYBOARD
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
DECLARE result as int
CALL SET_TRACE with context, "in create receive thread",
"STATE_CREATE_RECV_THREAD"
ASSIGN result with CALL pthread_create for recv_thread with init_recv_function and ctx
IF result is less than 0 THEN
   RETURN STATE_ERROR
ELSE
   RETURN STATE_READ_FROM_KEYBOARD
ENDIF
```

read_from_keyboard_handler

Purpose

Reads input from the keyboard and determines the next state based on the input.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CHECK_WINDOW, STATE_CLEANUP
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, empty message, "STATE_READ_FROM_KEYBOARD"
WHILE exit_flag is not true DO
   IF CALL read_keyboard with address of ctx's args temp_buffer is -1 THEN
      INCREMENT exit_flag
      RETURN STATE_CLEANUP
   ENDIF
   RETURN STATE_CHECK_WINDOW
END WHILE
RETURN STATE_CLEANUP
```

check_window_handler

Purpose

Checks the window's availability for packet transmission and decides the next state.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_ADD_PACKET_TO_WINDOW
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, empty message, "STATE_CHECK_WINDOW"
IF is_window_available is false OR ctx's args is_buffered is true THEN
    RETURN STATE_ADD_PACKET_TO_BUFFER
ELSE
    RETURN STATE_ADD_PACKET_TO_WINDOW
ENDIF
```

add_packet_to_buffer_handler

Purpose

Adds a packet to the buffer if the window is not available.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

STATE_READ_FROM_KEYBOARD,
STATE_CHECK_WINDOW_THREAD

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, empty message, "STATE_ADD_PACKET_TO_BUFFER"
IF ctx's args head is NULL THEN
   CALL init_list with address of ctx's args head and ctx's args temp_buffer
   INCREMENT ctx's args is_buffered
   RETURN STATE_CHECK_WINDOW_THREAD
ELSE
   CALL push with ctx's args head and ctx's args temp_buffer
   RETURN STATE_READ_FROM_KEYBOARD
ENDIF
```

add_packet_to_window_handler

Purpose

Adds a packet to the window for transmission.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_SEND_MESSAGE

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, empty message, "STATE_ADD_PACKET_TO_WINDOW"
CALL create_data_packet with address of ctx's args temp_message, ctx's args window, and ctx's args temp_buffer
RETURN STATE_SEND_MESSAGE
```

check_window_thread_handler

Purpose

Initializes the window checker function in a separate thread.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_READ_FROM_KEYBOARD

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, empty message, "STATE_CHECK_WINDOW_THREAD"
CALL init_window_checker_function with ctx as argument
RETURN STATE_READ_FROM_KEYBOARD
```

send_message_handler

Purpose

Sends a message packet over the network.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CREATE_TIMER_THREAD
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, empty message, "STATE_SEND_PACKET"
IF CALL send_packet with ctx's sockfd, server_addr_struct, window, and temp_message returns
-1 THEN
   RETURN STATE_ERROR
ELSE
   RETURN STATE_CREATE_TIMER_THREAD
ENDIF
```

create_new_timer_thread_handler

Purpose

Creates a new timer thread and adds it to the thread pool.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_READ_FROM_KEYBOARD
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
DECLARE temp_thread_pool as pointer to pthread_t
ASSIGN temp_thread_pool with ctx's args thread_pool
CALL SET_TRACE with context, empty message, "STATE_CREATE_TIMER_THREAD"
INCREMENT ctx's args num_of_threads
REALLOCATE memory for temp_thread_pool based on num_of_threads
IF temp_thread_pool is NULL THEN
    RETURN STATE_ERROR
ENDIF
ASSIGN ctx's args thread_pool with temp_thread_pool
CREATE a new thread in thread_pool with init_timer_function and ctx
RETURN STATE_READ_FROM_KEYBOARD
```

cleanup_handler

Purpose

Performs cleanup operations, closing sockets, and joining threads.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

FSM_EXIT

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, "in cleanup handler", "STATE_CLEANUP"
CALL pthread_join with ctx's args recv_thread
IF CALL socket_close with ctx's args sockfd returns non-zero THEN
    PRINT "close socket error"
ENDIF
FOR EACH thread in ctx's args thread_pool DO
    CALL pthread_join with thread
END FOR
IF closing client_gui_fd in ctx's args fails THEN
    PRINT "close socket error" for proxy GUI socket
IF closing connected_gui_fd in ctx's args fails THEN
    PRINT "close socket error" for connected GUI socket
FREE ctx's args thread_pool
FREE ctx's args window
CLOSE sent_data file in ctx's args
CLOSE received_data file in ctx's args
RETURN FSM_EXIT
```

error_handler

Purpose

Handles errors by logging the error information.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

STATE_CLEANUP

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
PRINT error message using err's err_msg, file_name, function_name, and error_line
RETURN STATE_CLEANUP
```

listen_handler

Purpose

Listens for incoming packets and processes them

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_READ_FROM_KEYBOARD
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
DECLARE result as ssize_t
CALL SET_TRACE with context, empty message, "STATE_LISTEN_SERVER"
WHILE exit_flag is not true DO
    ASSIGN result with CALL receive_packet with ctx's sockfd, window, and temp_packet
    IF result is -1 THEN
        RETURN STATE_ERROR
    ENDIF
    PRINT "Server packet with seq number: ", ctx's temp_packet.hd.seq_number
    RETURN STATE_CHECK_ACK_NUMBER
END WHILE
RETURN FSM_EXIT
```

check_ack_number_handler

Purpose

Checks the ACK number in the received packet and decides the next state based on the result.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_LISTEN, STATE_REMOVE_FROM_WINDOW
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
DECLARE result as int
CALL SET_TRACE with context, empty message, "STATE_CHECK_ACK_NUMBER"
ASSIGN result with CALL read_flags with ctx's args temp_packet's flags
IF result is RECV_ACK THEN
   PRINT "received ack"
   IF CALL check_ack_number with expected_ack_number and ack_number from ctx's args
window and temp_packet THEN
      PRINT "removing from window"
      RETURN STATE_REMOVE_FROM_WINDOW
   ENDIF
ELSE IF result is END_CONNECTION THEN
   RETURN STATE_TERMINATION
ELSE IF result is SEND_HANDSHAKE_ACK THEN
   PRINT "received syn ack again"
   DECLARE pt as packet, assign with ctx's args temp_packet
   CALL create_handshake_ack_packet with sockfd, server_addr_struct, window, and
temp_packet from ctx's args
   RETURN STATE_LISTEN
ENDIF
RETURN STATE_SEND_PACKET
```

remove_packet_from_window_handler

Purpose

Removes a packet from the window after successful acknowledgment.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_LISTEN
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, empty message, "STATE_CHECK_ACK_NUMBER"
CALL remove_packet_from_window with window and temp_packet from ctx's args
RETURN STATE_LISTEN
```

send_packet_handler

Purpose

Handles the sending of packets in the FSM.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_LISTEN
Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, empty message, "STATE_SEND_PACKET"
CALL read_received_packet with sockfd, server_addr_struct, window, and temp_packet from
ctx's args
RETURN STATE_LISTEN
```

init_recv_function(thread)

Purpose

Initializes the receive thread with a set of FSM transitions.

Parameters

void *ptr: Pointer to FSM context.

Return

void: NULL upon completion

Pseudocode

```
DECLARE ctx as pointer to fsm_context from ptr
DECLARE err as fsm_error
DEFINE transitions as array of fsm_transition
CALL fsm_run with ctx, err, and transitions
RETURN NULL
```

init_timer_function (thread)

Purpose

Initializes the receive thread with a set of FSM transitions.

Parameters

void *ptr: Pointer to FSM context.

Return

void: exits the thread upon completion.

Pseudocode

```
DECLARE ctx as pointer to fsm_context from ptr
DECLARE err as fsm_error
DECLARE index as int, ASSIGN with CALL previous_index with ctx's args window
DECLARE counter as int, INITIALIZE to 0
WHILE ctx's args window at index is_packet_full is true DO
   CALL sleep with TIMER_TIME
   IF ctx's args window at index is_packet_full is still true THEN
      CALL send_packet with sockfd, server_addr_struct, window, and packet at index from ctx's
args
      INCREMENT counter
   ENDIF
END WHILE
CALL pthread_exit with NULL
```

init_window_checker_function (thread)

Purpose

Initializes the receive thread with a set of FSM transitions.

Parameters

void *ptr: Pointer to FSM context.

Return

void: NULL upon completion

Pseudocode

```
DECLARE ctx as pointer to fsm_context from ptr
DECLARE err as pointer to fsm_error, INITIALIZE to NULL
WHILE ctx's args head is not NULL DO
   IF is_window_available is true THEN
      DECLARE pt as packet
      CALL create_data_packet with address of pt, window, and head's data from ctx's args
      CALL send_packet with sockfd, server_addr_struct, window, and address of pt from ctx's
args
      CALL create_timer_thread_handler with ctx and err
      PRINT "sent packet with seq number", pt.hd.seq_number
      CALL pop with address of ctx's args head
   ENDIF
END WHILE
ASSIGN ctx's args is_buffered to 0
RETURN NULL
```

init_gui_function(thread)

Purpose

Continuously listens for GUI connections until an exit condition is met.

Parameters

void *ptr: Pointer to FSM context.

Return

void: NULL upon completion

Pseudocode

```
DECLARE ctx as pointer to fsm_context from ptr
DECLARE err as fsm_error
WHILE exit_flag is not true DO
    ASSIGN ctx's args connected_gui_fd with CALL socket_accept_connection with
client_gui_fd from ctx's args and address of err
    INCREMENT ctx's args is_connected_gui
END WHILE
RETURN NULL
```

create_file

Purpose

Creates and opens a file for writing, handling file opening errors.

Parameters

const char *filepath: Path to the file to be created.
struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: 0
Failure: -1

Pseudocode

```
DECLARE fp as pointer to FILE, ASSIGN with CALL fopen with filepath and "w" mode
IF fp is NULL THEN
   CALL SET_ERROR with err and "Error in opening file."
   RETURN -1
ENDIF
ASSIGN value at fp to *fp
RETURN 0
```

parse_arguments

Purpose

Parses command-line arguments for server and client addresses, ports, and window size. It handles errors and ensures each argument is passed only once.

Parameters

int argc: Number of command-line arguments.
char *argv[]: Array of command-line argument strings.
char **server_addr: Pointer to store the server address.
char **client_addr: Pointer to store the client address.
char **server_port_str: Pointer to store the server port string.
char **client_port_str: Pointer to store the client port string.
uint8_t *window_size: Pointer to store the window size.
struct fsm_error *err: Pointer to error structure for error handling.

Return

Success: 0
Failure: -1

Pseudocode

```
DECLARE opt as int
DECLARE C_flag, c_flag, S_flag, s_flag, as bool, INITIALIZE to 0
DISABLE getopt error messages
WHILE parsing command-line arguments using getopt DO
  SWITCH opt
    CASE 'C': // Client address
      IF C_flag is true THEN
        RETURN -1
      ENDIF
      INCREMENT C_flag
      ASSIGN client address with optarg
    CASE 'c': // Client port
      IF c_flag is true THEN
        RETURN -1
      ENDIF
      INCREMENT c_flag
      ASSIGN client port string with optarg
    CASE 'S': // Server address
      IF S_flag is true THEN
```

```
            RETURN -1
         ENDIF
         INCREMENT S_flag
         ASSIGN server address with optarg
      CASE 's': // Server port
         IF s_flag is true THEN
            RETURN -1
         ENDIF
         INCREMENT s_flag
         ASSIGN server port string with optarg
      CASE 'w': // Window size
         IF w_flag is true THEN
            RETURN -1
         ENDIF
         INCREMENT w_flag
         CALL convert_to_int to convert window size, RETURN -1 if error occurs
      CASE 'h': // Help
         CALL usage and SET_ERROR
         RETURN -1
      CASE '?':
         CALL usage and SET_ERROR
         RETURN -1
      DEFAULT:
         CALL usage
   END SWITCH
END WHILE
RETURN 0
```

usage

Purpose

Displays the usage information for the program, detailing the expected command-line arguments.

Parameters

const char *program_name: The name of the program.

Return

Success: STATE_LISTEN
Failure: STATE_ERROR

Pseudocode

```
PRINT "Usage: <program_name> [-C] <value> [-c] <value> [-S] <value> [-s] <value> [-w]
<value> [-h]" to stderr
PRINT detailed options and their descriptions to stderr
```

handle_arguments

Purpose

Validates the required command-line arguments for server and client addresses and ports, and window size. Sets errors if arguments are missing or invalid.

Parameters

const char *program_name: The name of the program.
const char *server_addr: Pointer to a string for storing the server address.
const char *client_addr: Pointer to a string for storing the client address.
const char *server_port_str: Pointer to a string for storing the server port as a string.
const char *client_port_str: Pointer to a string for storing the client port as a string.
in_port_t *server_port: Pointer to store the parsed server port.
in_port_t *client_port: Pointer to store the parsed client port.
uint8_t window_size: The size of the window, as a uint8_t value.
struct fsm_error *err: Pointer to an error structure for handling and recording any errors that occur.

Return

Success: 0
Failure: -1

Pseudocode

```
IF server_addr is NULL THEN
   RETURN -1
ENDIF
IF client_addr is NULL THEN
   RETURN -1
ENDIF
IF server_port_str is NULL THEN
   RETURN -1
ENDIF
IF client_port_str is NULL THEN
   RETURN -1
IF window_size is less than 3 THEN
   RETURN -1
CALL parse_in_port_t for server_port_str
IF error THEN RETURN -1
CALL parse_in_port_t for client_port_str
IF error THEN RETURN -1
RETURN 0
```

parse_in_port_t

Purpose

Parses a string to an in_port_t type, validating the input.

Parameters

const char *binary_name: The name of the program.
const char *str: The string to parse.
in_port_t *port: Pointer to store the parsed value.
struct fsm_error *err: Error handling structure.

Return

Success: 0
Failure: -1

Pseudocode

```
PARSE str to a uintmax_t value
IF error occurred during parsing THEN
   RETURN -1
ENDIF
IF parsed_value is greater than UINT16_MAX THEN
   CALL usage
   RETURN -1
ENDIF
ASSIGN parsed_value to *port
RETURN 0
```

convert_to_int

Purpose

Converts a string to an integer (uint8_t), checking for errors and range.

Parameters

const char *binary_name: The name of the program.

char *string: The string to convert.
uint8_t *value: Pointer to store the converted value.
struct fsm_error *err: Error handling structure.

Return

Success: 0
Failure: -1

Pseudocode

```
PARSE string to a uintmax_t value
IF error occurred during parsing, THEN
   RETURN -1
ENDIF
IF parsed_value is greater than 100 THEN
   CALL usage
   RETURN -1
ENDIF
ASSIGN parsed_value to *value
RETURN 0
```

fsm_run

Purpose

Executes the finite state machine (FSM) by transitioning between states based on a set of defined transitions until reaching the exit state.

Parameters

struct fsm_context *context: Pointer to the FSM context, containing state and data for the FSM.
struct fsm_error *err: Pointer to a structure for error handling.
const struct fsm_transition transitions[]: Array of FSM transitions

Return

Success: 0

Pseudocode

```
DECLARE from_id as int, INITIALIZE to FSM_INIT
DECLARE to_id as int, INITIALIZE to FSM_USER_START
WHILE to_id is not FSM_EXIT DO
    DECLARE perform as fsm_state_func
    DECLARE next_id as int
    ASSIGN perform to the result of fsm_transition with context, from_id, to_id, and transitions
    IF perform is NULL THEN
    ENDIF
    ASSIGN from_id to to_id
    ASSIGN next_id to the result of calling perform with context and err
    ASSIGN to_id to next_id
END WHILE
RETURN 0
```

fsm_transition

Purpose

Finds and returns the function to be executed for a specific state transition in the FSM.

Parameters

struct fsm_context *context: Pointer to the FSM context, containing state and data for the FSM.
int from_id: ID of the current state.
int to_id: ID of the next state.
const struct fsm_transition transitions[]: Array of FSM transitions

Return

Success: performs transition
Failure: NULL

Pseudocode

```
DECLARE transition as pointer to fsm_transition, ASSIGN to the first element of transitions array
WHILE transition's from_id is not FSM_IGNORE DO
    IF transition's from_id is from_id AND transition's to_id is to_id THEN
        RETURN transition's perform function
    ENDIF
    INCREMENT transition to point to the next element in the transitions array
END WHILE
RETURN NULL
```

init_list

Purpose

Initializes a linked list by creating the head node with provided data.

Parameters

struct node **head: Pointer to the pointer to the head of the list.
char *data: Data to store in the head node.

Return

void

Pseudocode

```
DECLARE next_node as pointer to node, INITIALIZE to NULL
ALLOCATE memory for next_node as node
COPY data to next_node's data
ASSIGN next_node's next to NULL
ASSIGN head to next_node
```

push

Purpose

Adds a new node with the specified data at the end of the list.

Parameters

struct node **head: Pointer to the pointer to the head of the list.
char *data: Data to store in the head node.

Return

void

Pseudocode

```
DECLARE current as pointer to node, ASSIGN to head
WHILE current's next is not NULL DO
   ASSIGN current to current's next
END WHILE
ALLOCATE memory for current's next as node
COPY data to current's next's data
ASSIGN current's next's next to NULL
```

pop

Purpose

Adds a new node with the specified data at the end of the list.

Parameters

struct node **head: Pointer to the pointer to the head of the list.

Return

void

Pseudocode

```
DECLARE next_node as pointer to node, INITIALIZE to NULL
IF head is NULL THEN
    RETURN
ENDIF
ASSIGN next_node to head's next
FREE the node pointed by head
ASSIGN head to next_node
```

create_window

Purpose

Initializes a linked list by creating the head node with provided data.

Parameters

struct sent_packet **window: Double pointer to the window buffer.
uint8_t cmd_line_window_size: The size of the window specified on the command line.
struct fsm_error *err: Error structure for error handling.

Return

Success: 0
Failure: -1

Pseudocode

```
ASSIGN window_size to cmd_line_window_size
ALLOCATE memory for window of size 'sent_packet * window_size + 1'
IF window is NULL THEN
   SET error using strerror(errno)
   RETURN -1
ENDIF
FOR i from 0 to window_size DO
   SET window[i].is_packet_full to 0
END FOR
SET first_empty_packet to 0
SET first_unacked_packet to 0
SET is_window_available to TRUE
RETURN 0
```

window_empty

Purpose

Checks if the first empty packet slot in the window is available.

Parameters

struct sent_packet *window: pointer to the window buffer.

Return

Success: 1
Failure: 0

Pseudocode

```
IF window[first_empty_packet].is_packet_full is FALSE THEN
   SET is_window_available to TRUE
   RETURN 1
ELSE
   SET is_window_available to FALSE
   RETURN 0
ENDIF
```

first_packet_ring_buffer

Purpose

Finds the first available packet slot in the ring buffer.

Parameters

struct sent_packet *window: pointer to the window buffer.

Return

Success: 1
Failure: 0

Pseudocode

```
IF window[first_empty_packet].is_packet_full is FALSE THEN
   RETURN 1
ENDIF
IF first_empty_packet + 1 is greater than or equal to window_size THEN
   IF window[0].is_packet_full is FALSE THEN
      SET first_empty_packet to 0
      RETURN 1
   ENDIF
   SET first_empty_packet to first_unacked_packet
   RETURN 0
ELSE
   IF window[first_empty_packet + 1].is_packet_full is FALSE THEN
      INCREMENT first_empty_packet
      RETURN 1
   ELSE
      SET first_empty_packet to first_unacked_packet
      RETURN 0
   ENDIF
ENDIF
```

first_unacked_ring_buffer

Purpose

Updates the index of the first unacknowledged packet in the ring buffer.

Parameters

struct sent_packet *window: pointer to the window buffer.

Return

Success: 1
Failure: 0

Pseudocode

```
IF window[first_unacked_packet].is_packet_full is TRUE THEN
    RETURN 1
ENDIF
IF first_unacked_packet + 1 is greater than or equal to window_size THEN
    IF window[0].is_packet_full is TRUE THEN
        SET first_unacked_packet to 0
        RETURN 1
    ENDIF
    SET first_unacked_packet to first_empty_packet
    RETURN 0
ELSE
    IF window[first_unacked_packet + 1].is_packet_full is TRUE THEN
        INCREMENT first_unacked_packet
        RETURN 1
    ELSE
        SET first_unacked_packet to first_empty_packet
        RETURN 0
    ENDIF
ENDIF
```

send_packet

Purpose

Sends a packet using a socket and updates the window buffer.

Parameters

int sockfd: The socket file descriptor.
struct sockaddr_storage *addr: Pointer to the address structure.
struct sent_packet *window: Pointer to the window buffer.
struct packet *pt: Pointer to the packet to be sent.
FILE *fp: File pointer for logging.
struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE result as ssize_t
ASSIGN result to sendto call with sockfd, pt, and address information
IF result is less than 0 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
CALL write_stats_to_file with fp and pt
RETURN 0
```

add_packet_to_window

Purpose

Adds a packet to the window buffer and manages the state of the window.

Parameters

struct sent_packet *window: Pointer to the window buffer.

struct packet *pt: Pointer to the packet to be added.

Return

Success: 0

Pseudocode

```
GET current time and store in pt's header tv
ASSIGN *pt to window at first_empty_packet
IF pt's header flags is ACK THEN
   SET is_packet_full of window at first_empty_packet to FALSE
   INCREMENT first_empty_packet
   CALL first_unacked_ring_buffer with window
ELSE
   SET is_packet_full of window at first_empty_packet to TRUE
IF pt's header flags is SYN THEN
   SET expected_ack_number of window at first_empty_packet to pt's header seq_number + 1
ELSE IF pt's header flags is SYNACK THEN
   SET expected_ack_number of window at first_empty_packet to pt's header seq_number + 1
   SET seq_number of window at first_empty_packet's pt header to pt's header seq_number + 1
ELSE
   SET expected_ack_number of window at first_empty_packet to pt's header seq_number +
strlen(pt's data)
CALL first_packet_ring_buffer with window
CALL window_empty with window
RETURN 0
```

receive_packet

Purpose

Receives a packet from a socket and updates the window state.

Parameters

int sockfd: Socket file descriptor.
struct sent_packet *window: Pointer to the window buffer.
struct packet *pt: Pointer to the packet to be populated with received data.
FILE *fp: File pointer for statistics logging.
struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE client_addr as sockaddr_storage
DECLARE client_addr_len as socklen_t, INITIALIZE to size of client_addr
DECLARE temp_pt as packet
DECLARE result as ssize_t
ASSIGN result to recvfrom call with sockfd, temp_pt, and client address
IF result is -1 THEN
   SET error using strerror(errno)
   RETURN -1
ENDIF
ASSIGN temp_pt to *pt
CALL write_stats_to_file with fp and pt
CALL window_empty with window
RETURN 0
```

remove_single_packet

Purpose

Removes a single packet from the window

Parameters

struct sent_packet *window: Pointer to the window buffer.

struct packet *pt: Pointer to the packet to be added.

Return

Success: 1

Failure: 0

Pseudocode

```
IF window at first_unacked_packet's expected_ack_number equals pt's header ack_number
THEN
    SET is_packet_full of window at first_unacked_packet to FALSE
    CALL first_unacked_ring_buffer with window
    RETURN 1
ENDIF
RETURN 0
```

remove_cumulative_packets

Purpose

Removes cumulative packets from the window

Parameters

struct sent_packet *window: Pointer to the window buffer.

struct packet *pt: Pointer to the packet to be added.

Return

Success: 1

Failure: 0

Pseudocode

```
get the index of the highest ACK packet that was received
IF index > index of the first unacked packet
        CALL remove_lesser_index
ENDIF
IF index < index of the first unacked packet
        CALL remove_greater_index
ENDIF
SET first_unacked_packet to index
CALL first_unacked_ring_buffer
RETURN 0
```

remove_lesser_index

Purpose

remove packets from the window from 0 to the index

Parameters

struct sent_packet *window: Pointer to the window buffer.

uint8_t index: the index to remove from the window till

Return

Success: 1

Failure: 0

Pseudocode

```
FOR 0 till index
        remove packet from the window
ENDIF
CALL remove_greater with the index as the window_size - 1
RETURN 0
```

remove_greater_index

Purpose

remove packets from the window from first unacked packet to the index

Parameters

struct sent_packet *window: Pointer to the window buffer.

uint8_t index: the index to remove from the window till

Return

Success: 1

Failure: 0

Pseudocode

```
FOR first_unacked_packet till index
        remove packet from the window
ENDIF
RETURN 0
```

remove_packet_from_window

Purpose

removes packets from the window depending on the number

Parameters

struct sent_packet *window: Pointer to the window buffer.

struct packet *pt: Pointer to the packet to be added.

Return

Success: 1

Failure: 0

Pseudocode

```
IF expected ack number equals the ack number in packet
        CALL remove_single_packet
        RETURN 0
ENDIF
if expected ack number greater than the ack number in the packet
        CALL remove_cumulative_packet
        RETURN 0
ENDIF

RETURN -1
```

is_window_empty

Purpose

returns if the window is empty

Parameters

void

Return

1 is window is empty

0 if window is not empty

Pseudocode

```
RETURN is_window_available
```

get_expected_ack_number

Purpose

return the expected ack number for the current tail of the ring buffer

Parameters

struct sent_packet *window

Return

1 is window is empty

0 if window is not empty

Pseudocode

```
RETURN the value of the expected ack number with the index of first unacked packet
```

create_second_handshake_seq_number

Purpose

Generates a second sequence number for the second handshake.

Parameters

void

Return

The new sequence number

Pseudocode

```
RETURN 100
```

create_sequence_number

Purpose

Generates a new sequence number based on the previous sequence number and data size.

Parameters

uint32_t prev_seq_number: The previous sequence number.

uint32_t data_size: The size of the data.

Return

The new sequence number

Pseudocode

```
RETURN prev_seq_number + data_size
```

create_ack_number

Purpose

Generates an acknowledgment number based on the received sequence number and data size.

Parameters

uint32_t recv_seq_number: The received sequence number.

uint32_t data_size: The size of the data.

Return

The acknowledgment number.

Pseudocode

```
RETURN recv_seq_number + data_size
```

previous_seq_number

Purpose

Finds the previous sequence number in the window.

Parameters

struct sent_packet *window: Pointer to the window buffer.

Return

The previous sequence number.

Pseudocode

```
IF first_empty_packet is 0 THEN
    RETURN seq_number from header of packet in window at window_size - 1
ENDIF
RETURN seq_number from header of packet in window at first_empty_packet - 1
```

previous_date_size

Purpose

Determines the size of the data in the previous packet in the window.

Parameters

struct sent_packet *window: Pointer to the window buffer.

Return

Size of data in previous packet

Pseudocode

```
IF first_empty_packet is 0 THEN
   RETURN length of data in window at window_size - 1
ELSE
   RETURN length of data in window at first_empty_packet - 1
ENDIF
```

previous_ack_number

Purpose

Retrieves the acknowledgment number of the previous packet in the window.

Parameters

struct sent_packet *window: Pointer to the window buffer.

Return

Acknowledgment number of previous packet.

Pseudocode

```
IF first_empty_packet is 0 THEN
    RETURN ack_number from header of packet in window at window_size - 1
ELSE
    RETURN ack_number from header of packet in window at first_empty_packet - 1
ENDIF
```

check_ack_number

Purpose

Compares an expected acknowledgment number with an actual acknowledgment number.

Parameters

uint32_t expected_ack_number: The expected acknowledgment number.

uint32_t ack_number: The actual acknowledgment number.

Return

Success: True

Failure: False

Pseudocode

```
IF window at the index of the first_unacked_packet is empty
    RETURN -1
RETURN the or value of CALL to check_ack_number_equal and check_ack_number_greater
```

check_ack_number_equal

Purpose

Compares an expected acknowledgment number with an actual acknowledgment number.

Parameters

uint32_t expected_ack_number: The expected acknowledgment number.

uint32_t ack_number: The actual acknowledgment number.

Return

Success: True

Failure: False

Pseudocode

```
RETURN expected_ack_number is equal to the ack number
```

check_ack_number_greater

Purpose

Checks the if ack number is greater than the expected ack number

Parameters

uint32_t expected_ack_number: The expected acknowledgment number.

uint32_t ack_number: The actual acknowledgment number.

Return

Success: True

Failure: False

Pseudocode

```
RETURN ack number is greater than the expected ack number
```

get_ack_number_index

Purpose

returns the index of the ack number passed in, in relation to the window

Parameters

struct sent_packet *window: Pointer to the window buffer.

uint32_t ack_number: The actual acknowledgment number.

Return

Success: True

Failure: False

Pseudocode

```
ITERATES through the window
    IF expected number of the window at that index is equal to the ack number
        RETURN index
    ENDIF
RETURN -1
```

previous_index

Purpose

Determines the index of the previous packet in the window buffer.

Parameters

struct sent_packet *window: Pointer to the window buffer.

Return

Index of previous packet in the window

Pseudocode

```
IF first_empty_packet is 0 THEN
   RETURN window_size - 1
ELSE
   RETURN first_empty_packet - 1
ENDIF
```

write_stats_to_file

Purpose

Writes packet statistics to a file.

Parameters

FILE *fp: File pointer where the statistics will be written.

const struct packet *pt: Pointer to the packet whose statistics are to be written.

Return

Success : 0

Pseudocode

```
WRITE pt's header seq_number, ack_number, flags, window_size, checksum, and data to fp
FLUSH the file stream fp
RETURN 0
```

read_received_packet

Purpose

Processes a received packet based on its flags and performs appropriate actions.

Parameters

int sockfd: Socket file descriptor.

struct sockaddr_storage *addr: Pointer to the address structure.

struct sent_packet *window: Pointer to the window buffer.

struct packet *pt: Pointer to the received packet.

FILE *fp: File pointer for logging.

Return

Success: 0
Failure: -1

Pseudocode

```
SWITCH on result
   CASE ESTABLISH_HANDSHAKE:
      CALL send_syn_ack_packet with sockfd, addr, window, pt, fp, err
      BREAK
   CASE SEND_HANDSHAKE_ACK:
      CALL send_handshake_ack_packet with sockfd, addr, window, pt, fp, err
      BREAK
   CASE SEND_ACK:
      CALL send_data_ack_packet with sockfd, addr, window, pt, fp, err
      BREAK
   CASE RECV_ACK:
      CALL recv_ack_packet with sockfd, addr, window, pt, fp, err
      BREAK
   CASE END_CONNECTION:
      CALL recv_termination_request with sockfd, addr, window, pt, fp, err
      BREAK
   CASE RECV_RST, UNKNOWN_FLAG, default:
      RETURN -1
RETURN 0
```

read_flags

Purpose

Interprets the flags of a packet and returns the corresponding action.

Parameters

uint8_t flags: Flags of the packet.

Return

The action should be taken in integer format.

Pseudocode

```
IF flags is SYN THEN
   RETURN ESTABLISH_HANDSHAKE
ENDIF
IF flags is SYNACK THEN
   RETURN SEND_HANDSHAKE_ACK
ENDIF
IF flags is PSHACK THEN
   RETURN SEND_ACK
ENDIF
IF flags is ACK THEN
   RETURN RECV_ACK
ENDIF
IF flags is FINACK THEN
   RETURN END_CONNECTION
ENDIF
IF flags is RSTACK THEN
   RETURN RECV_RST
ENDIF
RETURN UNKNOWN_FLAG
```

send_syn_packet

Purpose

Sends a SYN packet to initiate a 3 way handshake.

Parameters

int sockfd: Socket file descriptor for sending the packet.
struct sockaddr_storage *addr: Address structure for the destination.
struct sent_packet *window: Window buffer for managing sent packets.
FILE *fp: File pointer for logging.
struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to CALL create_sequence_number with 0, 0
SET packet_to_send's header ack_number to CALL create_ack_number with 0, 0
SET packet_to_send's header flags to SYN
SET packet_to_send's header window_size to global window_size
CLEAR packet_to_send's data
CALL calculate_checksum with &packet_to_send.hd.checksum, packet_to_send.data
CALL send_packet with sockfd, addr, window, &packet_to_send, fp, err
CALL add_packet_to_window with window, &packet_to_send
RETURN 0
```

send_syn_ack_packet

Purpose

Sends a SYN-ACK packet as a response in the handshake process.

Parameters

int sockfd: The socket file descriptor used for network communication.
struct sockaddr_storage *addr: Pointer to the address structure of the sender or receiver.
struct sent_packet *window: Pointer to the window buffer used for managing sent packets.
struct packet *pt: Pointer to the packet structure that has been received.
FILE *fp: File pointer used for logging packet information.
struct fsm_error *err: Pointer to an error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to CALL create_second_handshake_seq_number
SET packet_to_send's header ack_number to CALL create_ack_number with pt.hd.seq_number
SET packet_to_send's header flags to CALL create_flags with pt.hd.flags
SET packet_to_send's header window_size to global window_size
CLEAR packet_to_send's data
CALL calculate_checksum with &packet_to_send.hd.checksum, packet_to_send.data
CALL send_packet with sockfd, addr, window, &packet_to_send, fp, err
CALL add_packet_to_window with window, &packet_to_send
RETURN 0
```

send_handshake_ack_packet

Purpose

Sends a handshake acknowledgment packet.

Parameters

int sockfd: The socket file descriptor used for network communication.
struct sockaddr_storage *addr: Pointer to the address structure of the sender or receiver.
struct sent_packet *window: Pointer to the window buffer used for managing sent packets.
struct packet *pt: Pointer to the packet structure that has been received.
FILE *fp: File pointer used for logging packet information.
struct fsm_error *err: Pointer to an error structure for error handling.

Return

Success: 0

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to sequence number from pt's ack_number
SET packet_to_send's header ack_number to ack number from pt's seq_number plus 1
SET packet_to_send's header flags to flags from pt's flags
SET packet_to_send's header window_size to global window_size
CLEAR packet_to_send's data
CALL calculate_checksum with packet_to_send's header checksum and data
CALL send_packet with sockfd, addr, window, &packet_to_send, fp, err
CALL add_packet_to_window with window and &packet_to_send
RETURN 0
```

send_data_packet

Purpose

Sends a data packet with the specified payload.

Parameters

int sockfd: The socket file descriptor used for network communication.
struct sockaddr_storage *addr: Pointer to the address structure of the sender or receiver.
struct sent_packet *window: Pointer to the window buffer used for managing sent packets.
FILE *fp: File pointer used for logging packet information.
struct fsm_error *err: Pointer to an error structure for error handling.
char *data: Payload to be sent in the packet.

Return

Success: 0

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to sequence number from previous seq_number and data_size
SET packet_to_send's header ack_number to ack number from previous ack_number
SET packet_to_send's header flags to PSHACK
SET packet_to_send's header window_size to global window_size
COPY data to packet_to_send's data
CALL calculate_checksum with packet_to_send's header checksum and data
CALL send_packet with sockfd, addr, window, &packet_to_send, fp, err
CALL add_packet_to_window with window and &packet_to_send
RETURN 0
```

send_data_ack_packet

Purpose

Sends an acknowledgment packet for received data.

Parameters

int sockfd: The socket file descriptor used for network communication.
struct sockaddr_storage *addr: Pointer to the address structure of the sender or receiver.
struct sent_packet *window: Pointer to the window buffer used for managing sent packets.
struct packet *pt: Pointer to the packet structure that has been received.
FILE *fp: File pointer used for logging packet information.
struct fsm_error *err: Pointer to an error structure for error handling.

Return

Success: 0

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to sequence number from previous seq_number and data_size
SET packet_to_send's header ack_number to ack number from pt's seq_number and length of pt's data
SET packet_to_send's header flags to flags from pt's flags
SET packet_to_send's header window_size to global window_size
CLEAR packet_to_send's data
CALL calculate_checksum with packet_to_send's header checksum and data
CALL send_packet with sockfd, addr, window, &packet_to_send, fp, err
CALL add_packet_to_window with window and &packet_to_send
RETURN 0
```

create_flags

Purpose

Determines the appropriate flags for a response packet based on the received packet's flags.

Parameters

uint8_t flags: Flags from the received packet.

Return

New flag for the response packet

Pseudocode

```
IF flags is SYN THEN
   RETURN SYNACK
ELSE IF flags is SYNACK THEN
   RETURN ACK
ELSE IF flags is PSHACK THEN
   RETURN ACK
ELSE IF flags is FINACK THEN
   RETURN ACK
ELSE
   RETURN UNKNOWN_FLAG
ENDIF
```

create_data_packet

Purpose

Creates a data packet for transmission.

Parameters

struct packet *pt: Pointer to the packet to be created.

struct sent_packet *window: Pointer to the window buffer.

char *data: Data to be included in the packet

Return

success: 0

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to sequence number from previous seq_number and data_size
SET packet_to_send's header ack_number to ack number from previous ack_number
SET packet_to_send's header flags to PSHACK
SET packet_to_send's header window_size to global window_size
COPY data to packet_to_send's data
CALL calculate_checksum with packet_to_send's header checksum and data
ASSIGN packet_to_send to *pt
CALL add_packet_to_window with window and &packet_to_send
RETURN 0
```

create_handshake_ack_packet

Purpose

Creates and sends a handshake acknowledgment packet based on the received packet.

Parameters

int sockfd: Socket file descriptor used for sending the packet.
struct sockaddr_storage *addr: Pointer to the address structure for the destination.
struct sent_packet *window: Pointer to the window buffer for managing packets.
struct packet *pt: Pointer to the received packet that initiated this response.
FILE *fp: File pointer for logging.
struct fsm_error *err: Error structure for error handling.

Return

success: 0

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to result of create_sequence_number with pt's
ack_number and 0
SET packet_to_send's header ack_number to result of create_ack_number with pt's seq_number
and 1
SET packet_to_send's header flags to result of create_flags with pt's flags
SET packet_to_send's header window_size to global window_size
CLEAR packet_to_send's data
CALL calculate_checksum with address of packet_to_send's header checksum, packet_to_send's
data, and length of data
CALL send_packet with sockfd, addr, window, address of packet_to_send, fp, err
ASSIGN packet_to_send to *pt
RETURN 0
```

calculate_checksum

Purpose

Calculates a checksum for given data.

Parameters

uint16_t *checksum: Pointer to store the calculated checksum.

const char *data: Data for which the checksum is calculated.

size_t length: Length of the data.

Return

success: 0

Pseudocode

```
ASSIGN to *checksum the product of checksum_one and checksum_two results with data and
length
RETURN 0
```

checksum_one

Purpose

Calculates the first part of the checksum.

Parameters

const char *data: Data for which the checksum is calculated.

size_t length: Length of the data.

Return

The calculated checksum value

Pseudocode

```
DECLARE result as unsigned char, INITIALIZE to 0
FOR EACH byte in data up to length DO
   INCREMENT result by data[i] multiplied by 34
END FOR
RETURN result
```

checksum_two

Purpose

Calculates the second part of the checksum.

Parameters

const char *data: Data for which the checksum is calculated.

size_t length: Length of the data.

Return

The calculated checksum value

Pseudocode

```
DECLARE result as unsigned char, INITIALIZE to 0
FOR EACH byte in data up to length DO
   XOR result with data[i]
END FOR

RETURN result
```

socket_create

Purpose

Creates a socket with specified parameters.

Parameters

int domain: The domain of the socket (e.g., AF_INET).

int type: The type of the socket (e.g., SOCK_STREAM).

int protocol: The protocol to be used with the socket (usually 0 for default).

struct fsm_error *err: Error structure for error handling.

Return

Success: socket file descriptor

Failure: -1

Pseudocode

```
DECLARE sockfd as int
ASSIGN sockfd to socket call with domain, type, and protocol
IF sockfd is -1 THEN
   SET error using strerror(errno)
   RETURN -1
ENDIF
RETURN sockfd
```

read_keyboard

Purpose

Read from stdin

Parameters

char **buffer: Pointer to a char pointer where the input string will be stored.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE line as char array of size DATA_SIZE
CLEAR line
PRINT prompt message to enter string
IF fgets with line, size of line, and stdin is NULL THEN
    RETURN -1
ENDIF
ALLOCATE memory for buffer with size of line plus 1
COPY line to buffer
RETURN 0
```

start_listening

Purpose

Puts the socket in listening mode to listen for incoming connections.

Parameters

int sockfd: The socket file descriptor.

int backlog: The maximum length for the queue of pending connections.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
IF listen call with sockfd and backlog is -1 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN 0
```

socket_accept_connection

Purpose

Accepts a new connection on a socket

Parameters

int sockfd: The socket file descriptor.

struct fsm_error *err: Error structure for error handling.

Return

Success: file descriptor of the new socket

Failure: -1

Pseudocode

```
DECLARE client_addr as sockaddr
DECLARE client_addr_len as socklen_t, INITIALIZE to size of client_addr
DECLARE client_fd as int
SET errno to 0
ASSIGN client_fd to accept call with sockfd, client_addr, and client_addr_len
IF client_fd is -1 THEN
    IF errno is not EINTR THEN
        PRINT error message
    ENDIF
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN client_fd
```

socket_close

Purpose

Closes a socket

Parameters

int sockfd: The socket file descriptor.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
IF close call with sockfd is -1 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN 0
```

convert_address

Purpose

Creates an IPv4 or IPv6 sockaddr based off the ip address and port passed in.

Parameters

const char *address: The IP address in string format.

struct sockaddr_storage *addr: Pointer to the address structure to store the result.

in_port_t port: The port number.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE addr_str as char array of size INET6_ADDRSTRLEN
DECLARE addr_len as socklen_t
DECLARE vaddr as void pointer
DECLARE net_port as in_port_t, ASSIGN to htons of port
IF inet_pton with AF_INET, address, and ipv4_addr's sin_addr is 1 THEN
    DECLARE ipv4_addr as pointer to sockaddr_in, ASSIGN to addr
    SET addr_len to size of ipv4_addr
    SET ipv4_addr's sin_port to net_port
    SET vaddr to ipv4_addr's sin_addr
    SET addr's ss_family to AF_INET
ELSE IF inet_pton with AF_INET6, address, and ipv6_addr's sin6_addr is 1 THEN
    DECLARE ipv6_addr as pointer to sockaddr_in6, ASSIGN to addr
    SET addr_len to size of ipv6_addr
    SET ipv6_addr's sin6_port to net_port
    SET vaddr to ipv6_addr's sin6_addr
    SET addr's ss_family to AF_INET6
ELSE
    SET error "Address family not supported"
    RETURN -1
RETURN 0
```

socket_bind

Purpose

Binds a socket to an IP address and port.

Parameters

int sockfd: Socket file descriptor.

struct sockaddr_storage *addr: Pointer to the address structure to bind the socket to.

struct fsm_error *err: Error structure for error handling

Return

Success: 0

Failure: -1

Pseudocode

```
ALLOCATE ip_address using safe_malloc for NI_MAXHOST
ALLOCATE port using safe_malloc for NI_MAXSERV
IF get_sockaddr_info with addr, &ip_address, &port, err is not 0 THEN
   RETURN -1
ENDIF
PRINT "binding to: ", ip_address, ":", port
IF bind call with sockfd, addr, and size_of_address(addr) is -1 THEN
   SET error using strerror(errno)
   RETURN -1
ENDIF
PRINT "Bound to socket: ", ip_address, ":", port
FREE ip_address
FREE port
RETURN 0
```

size_of_address

Purpose

Determines the size of an address structure based on its sa_family.

Parameters

struct sockaddr_storage *addr: Pointer to the address structure.

Return

Success: Size of the address structure

Pseudocode

```
IF addr's ss_family is AF_INET
    RETURNsizeof(struct sockaddr_in) :
ELSE
    sizeof(struct sockaddr_in6)
```

get_sockaddr_info

Purpose

Retrieves IP address and port information from a sockaddr_storage structure.

Parameters

struct sockaddr_storage *addr: Pointer to the address structure.

char **ip_address: Pointer to store the IP address string.

char **port: Pointer to store the port string.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

failure: -1

Pseudocode

```
DECLARE temp_ip as char array of NI_MAXHOST
DECLARE temp_port as char array of NI_MAXSERV
DECLARE ip_size as socklen_t, ASSIGN to size of *addr
DECLARE result as int
ASSIGN result to getnameinfo with addr, ip_size, temp_ip, temp_port, and flags
NI_NUMERICHOST | NI_NUMERICSERV
IF result is not 0 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
COPY temp_ip to *ip_address
COPY temp_port to *port
RETURN 0
```

safe_malloc

Purpose

Safely allocates memory and checks for allocation failure.

Parameters

uint32_t size: Size of memory to allocate.

struct fsm_error *err: Error structure for error handling.

Return

void*: Pointer to the allocated memory.

Pseudocode

```
DECLARE ptr as void pointer
ALLOCATE memory to ptr with size
IF ptr is NULL and size is greater than 0 THEN
    PRINT error message
    EXIT program with EXIT_FAILURE
ENDIF
RETURN ptr
```

send_stats_gui

Purpose

Sends statistical data to a GUI over a socket.

Parameters

int sockfd: Socket file descriptor.

int stat: The statistical data to send.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE result as ssize_t
ASSIGN result to write call with sockfd, address of stat, and size of stat
IF result is 0 or less THEN
    RETURN -1
ENDIF
RETURN 0
```

Functions For Proxy

parse_arguments

Purpose

Parses command-line arguments to configure network settings including addresses, ports, and rates.

Parameters

int argc: Number of command-line arguments.
char *argv[]: Array of command-line argument strings.
char **server_addr: Pointer to store the server IP address.
char **client_addr: Pointer to store the client IP address.
char **proxy_addr: Pointer to store the proxy IP address.
char **server_port_str: Pointer to store the server port string.
char **client_port_str: Pointer to store the client port string.
uint8_t *client_delay_rate: Pointer to store the client's delay rate.
uint8_t *client_drop_rate: Pointer to store the client's packet drop rate.
uint8_t *server_delay_rate: Pointer to store the server's delay rate.
uint8_t *server_drop_rate: Pointer to store the server's packet drop rate.
uint8_t *corruption_rate: Pointer to store the packet corruption rate.
struct fsm_error *err: Error structure for error handling.

Return

Success: 0
Failure: -1

Pseudocode

```
DISABLE getopt error messages
INITIALIZE all flags (C_flag, S_flag, etc.) to 0
WHILE parsing command-line arguments using getopt DO
    SWITCH on opt
        CASE for each option ('C', 'c', 'S', 's', 'P', 'D', 'd', 'L', 'l', 'E'):
            CHECK if corresponding flag is set (e.g., C_flag for 'C')
                IF set, SET error "option can only be passed in once" and RETURN -1
            INCREMENT flag (e.g., C_flag++)
            ASSIGN corresponding parameter (e.g., *client_addr) to optarg
            IF option requires conversion to int (e.g., 'D', 'd', 'L', 'l', 'E'):
                CALL convert_to_int with optarg, corresponding rate variable, err
                IF conversion fails, RETURN -1
            BREAK
        CASE 'h':
            CALL usage and SET error "user called for help", RETURN -1
        CASE '?':
            SET error "Unknown option" and RETURN -1
        DEFAULT:
            CALL usage
    END SWITCH
END WHILE

RETURN 0
```

usage

Purpose

Displays usage instructions for the program.

Parameters

const char *program_name: The name of the program.

Return

void

Pseudocode

```
PRINT "Usage: <program_name> [-C] <value> [-c] <value> [-S] <value> [-s] <value> [-P]
<value>"
PRINT "[-w] <value> [-D] <value> [-d] <value> [-L] <value> [-l] <value> [-E] <value> [-h]"
PRINT "Options:"
PRINT "  -h                Display this help message"
PRINT "  -C <value>        Option 'C' (required) with value, Sets the IP client_addr"
PRINT "  -c <value>        Option 'c' (required) with value, Sets the client port"
PRINT "  -S <value>        Option 'S' (required) with value, Sets the IP server_addr"
PRINT "  -s <value>        Option 's' (required) with value, Sets the server port"
PRINT "  -P <value>        Option 'P' (required) with value, Sets the IP proxy_addr"
PRINT "  -D <value>        Option 'D' (required) with value, Sets the client drop rate"
PRINT "  -d <value>        Option 'd' (required) with value, Sets the server drop rate"
PRINT "  -L <value>        Option 'L' (required) with value, Sets the client delay rate"
PRINT "  -l <value>        Option 'l' (required) with value, Sets the server delay rate"
PRINT "  -E <value>        Option 'E' (required) with value, Sets the corruption rate"
```

handle_arguments

Purpose

Validates the required command-line arguments for network settings.

Parameters

const char *binary_name: The name of the binary or program.
const char *server_addr: The server IP address.
const char *client_addr: The client IP address.
const char *server_port_str: The server port string.
const char *proxy_addr: The proxy IP address.
const char *client_port_str: The client port string.
in_port_t *server_port: Pointer to store the parsed server port.
in_port_t *client_port: Pointer to store the parsed client port.
uint8_t client_delay_rate: The client's delay rate.
uint8_t client_drop_rate: The client's packet drop rate.
uint8_t server_delay_rate: The server's delay rate.
uint8_t server_drop_rate: The server's packet drop rate.
uint8_t corruption_rate: The packet corruption rate.
struct fsm_error *err: Error structure for error handling.

Return

void

Pseudocode

```
CHECK if client_addr, server_addr, server_port_str, client_port_str, proxy_addr are NULL
CHECK if client_drop_rate, client_delay_rate, server_drop_rate, server_delay_rate,
corruption_rate are greater than 100
IF any checks fail, SET appropriate error, CALL usage and RETURN -1
IF parse_in_port_t with binary_name, server_port_str, server_port, err is -1 THEN
   RETURN -1
ENDIF

IF parse_in_port_t with binary_name, client_port_str, client_port, err is -1 THEN
   RETURN -1
ENDIF
RETURN 0
```

parse_in_port_t

Purpose

Converts a string representation of a port number to its in_port_t type, validating the input.

Parameters

const char *binary_name: Name of the binary or program, used for error messages.

const char *str: String to be parsed into a port number.

in_port_t *port: Pointer to store the parsed port number.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0
Failure: -1

Pseudocode

```
DECLARE endptr as char pointer
DECLARE parsed_value as uintmax_t
PARSE str to uintmax_t value stored in parsed_value
IF errno is not 0 THEN
    SET error using strerror(errno)
    RETURN -1
IF endptr is not pointing to null character THEN
    SET error "Invalid characters in input."
    CALL usage with binary_name
    RETURN -1
IF parsed_value is greater than UINT16_MAX THEN
    SET error "in_port_t value out of range."
    CALL usage with binary_name
    RETURN -1
ASSIGN parsed_value to *port
RETURN 0
```

convert_to_int

Purpose

Converts a string to an integer value, ensuring it is within a valid range.

Parameters

const char *binary_name: Name of the binary or program.
char *string: String to be converted.
uint8_t *value: Pointer to store the converted value.
struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE endptr as char pointer
DECLARE parsed_value as uintmax_t
RESET errno to 0
PARSE string to uintmax_t value stored in parsed_value
IF errno is not 0 THEN
    SET error using strerror(errno)
    RETURN -1
IF endptr is not pointing to null character THEN
    SET error "Invalid characters in input."
    CALL usage with binary_name
    RETURN -1
IF parsed_value is greater than 100 THEN
    DECLARE error_message as char array
    FORMAT error_message with string "value out of range"
    SET error with error_message
    CALL usage with binary_name
    RETURN -1
ASSIGN parsed_value to *value
RETURN 0
```

random_number

Purpose

Generates a random number within a specified upper bound.

Parameters

size_t upperbound: The upper limit for the random number.

Return

Random number withing the range

Pseudocode

```
IF upperbound is 0 THEN
   RETURN 0
ELSE
   RETURN rand() modulo upperbound
ENDIF
```

calculate_lossiness

Purpose:

Calculates the likelihood of packet loss, delay, or corruption based on given rates.

Parameters

uint8_t drop_rate: Packet drop rate percentage.

uint8_t delay_rate: Packet delay rate percentage.

uint8_t corruption_rate: Packet corruption rate percentage.

Return

DROP, DELAY, CORRUPT, or SEND

Pseudocode

```
IF drop_rate is greater than 0 AND calculate_drop with drop_rate is TRUE THEN
   RETURN DROP
ENDIF
IF delay_rate is greater than 0 AND calculate_delay with delay_rate is TRUE THEN
   RETURN DELAY
ENDIF
IF corruption_rate is greater than 0 AND calculate_corruption with corruption_rate is TRUE
THEN
   RETURN CORRUPT
ENDIF
RETURN SEND
```

calculate_drop

Purpose:

Calculates the probability of a packet drop.

Parameters

uint8_t percentage: Percentage rate for drop.

Return

Success: TRUE
Failure: FALSE

Pseudocode

```
DECLARE rand as int
ASSIGN rand to random_number with 101

IF rand is greater than percentage
        RETURN FALSE
ELSE
        TRUE
```

calculate_delay

Purpose:

Calculates the probability of a packet delay.

Parameters

uint8_t percentage: Percentage rate for delay.

Return

Success: TRUE
Failure: FALSE

Pseudocode

```
DECLARE rand as int
ASSIGN rand to random_number with 101

IF rand is greater than percentage
        RETURN FALSE
ELSE
        TRUE
```

calculate_corruption

Purpose:

Calculates the probability of a packet corruption.

Parameters

uint8_t percentage: Percentage rate for corruption.

Return

Success: TRUE
Failure: FALSE

Pseudocode

```
DECLARE rand as int
ASSIGN rand to random_number with 101

IF rand is greater than percentage
    RETURN FALSE
ELSE
    TRUE
```

send_packet

Purpose:

Sends a packet over a socket.

Parameters

int sockfd: Socket file descriptor.

packet *pt: Pointer to the packet to be sent.

struct sockaddr_storage *addr: Pointer to the address structure.

Return

Success: 0
Failure: -1

Pseudocode

```
DECLARE result as ssize_t
ASSIGN result to sendto call with sockfd, pt, size of *pt, addr, and size_of_address of addr
IF result is less than 0 THEN
   RETURN -1
ENDIF
RETURN 0
```

receive_packet

Purpose:

receive a packet over a socket.

Parameters

int sockfd: Socket file descriptor.

packet *pt: Pointer to the packet structure to store the received packet.

Return

Success: 0
Failure: -1

Pseudocode

```
DECLARE client_addr as sockaddr_storage
DECLARE client_addr_len as socklen_t
DECLARE temp_pt as packet
DECLARE result as ssize_t
SET client_addr_len to size of client_addr
ASSIGN result to recvfrom call with sockfd, &temp_pt, size of temp_pt, client_addr,
&client_addr_len
IF result is -1 THEN
   PRINT "Error: ", strerror(errno)
   RETURN -1
ENDIF
ASSIGN temp_pt to *pt
RETURN 0
```

delay_packet

Purpose:

Delays the execution for a specified time to simulate packet delay.

Parameters

uint8_t delay_time: Delay time in seconds.

Return

void

Pseudocode

```
CALL sleep with delay_time
```

read_keyboard

Purpose:

Calculates the size of the address structure based on its family.

Parameters

uint8_t *client_drop: Pointer to store the client's drop rate.

uint8_t *client_delay: Pointer to store the client's delay rate.

uint8_t *server_drop: Pointer to store the server's drop rate.

uint8_t *server_delay: Pointer to store the server's delay rate.

uint8_t *corruption_rate: Pointer to store the data corruption rate.

Return

void

Pseudocode

```
DECLARE menu as string
DECLARE client_menu as string
INITIALIZE first_menu, second_menu, third_menu as integers
FORMAT menu string with options for lossiness values
FORMAT client_menu string with options for client lossiness
REPEAT
  DISPLAY menu string
  ASSIGN first_menu to result of read_menu with 4
  SWITCH on first_menu
    CASE 1 (Client Lossiness):
      REPEAT
        DISPLAY client_menu
        ASSIGN second_menu to result of read_menu with 3
        SWITCH on second_menu
          CASE 1 (Client Drop Rate):
            READ client drop rate, VALIDATE, and ASSIGN to *client_drop
            CASE 2 (Client Delay Rate):
               READ client delay rate, VALIDATE, and ASSIGN to *client_delay
            CASE 3 (Back):
               BREAK the loop
            CASE -1 (Invalid input):
               DISPLAY error message
            DEFAULT:
               BREAK
         END SWITCH
      UNTIL second_menu is 3
      BREAK
   CASE 2 (Server Lossiness):
      REPEAT
         DISPLAY server lossiness options
         READ and PROCESS server drop and delay rates
      UNTIL third_menu is 3
      BREAK
   CASE 3 (Data Corruption):
      READ data corruption rate, VALIDATE, and ASSIGN to *corruption_rate
      BREAK
   CASE 4 (Exit):
      RETURN from the function
   CASE -1 (Invalid input):
      DISPLAY error message
   DEFAULT:
      BREAK
  END SWITCH
UNTIL the loop is exited
```

read_menu

Purpose:

Reads an integer input from the user and ensures it's within the specified upper bound.

Parameters

int upperbound: The maximum valid value for the input.

Return

Success: User input
Failure: -1

Pseudocode

```
DECLARE buf as char array of size 128
READ a line from stdin into buf
DECLARE endptr as char pointer
PARSE buf to an integer using strtol, store the result in temp
IF errno is not 0 THEN
    RETURN -1
ENDIF
IF endptr is not pointing to a newline character THEN
    RETURN -1
ENDIF
IF temp is greater than upperbound THEN
    RETURN -1
ENDIF
RETURN temp
```

corrupt_data

Purpose:

Corrupts a given string by randomly toggling bits.

Parameters

char **data: Pointer to the string to be corrupted.

size_t length: The length of the string to be corrupted.

Return

Success: 0

Pseudocode

```
DUPLICATE *data into temp
FOR i from 0 to length DO
   DECLARE rbyte as int, ASSIGN a random number within the string length
   DECLARE rbit as int, ASSIGN a random number within 0 to 7 (for a byte)
   XOR temp at index rbyte with 1 shifted left by rbit
END FOR
ASSIGN temp to *data
RETURN 0
```

write_stats_to_file

Purpose:

Writes packet statistics to a file.

Parameters

FILE *fp: File pointer where the statistics will be written.

const struct packet *pt: Pointer to the packet whose statistics are to be written.

Return

Success : 0

Pseudocode

```
WRITE pt's header seq_number, ack_number, flags, window_size, checksum, and data to fp
FLUSH the file stream fp
RETURN 0
```

parse_argument_handler

Purpose:

Handles the parsing of command-line arguments in the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_HANDLE_ARGUMENTS
Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "in parse arguments handler" at "STATE_PARSE_ARGUMENTS"
IF parse_arguments with context's argc, argv, and relevant pointers returns -1 THEN
   RETURN STATE_ERROR
ENDIF
RETURN STATE_HANDLE_ARGUMENTS
```

handle_argument_handler

Purpose:

Handles the validated arguments in the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CONVERT_ADDRESS
Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "in handle arguments" at "STATE_HANDLE_ARGUMENTS"
IF handle_arguments with context's argv[0], server_addr, client_addr, etc., returns non-zero
THEN
   RETURN STATE_ERROR
ENDIF
IF CALL create_file with "../client_received_data.csv", address of received_data from ctx's args,
err returns -1 THEN
   RETURN STATE_ERROR
ENDIF
IF CALL create_file with "../client_sent_data.csv", address of sent_data from ctx's args, err
returns -1 THEN
   RETURN STATE_ERROR
ENDIF
RETURN STATE_CONVERT_ADDRESS
```

convert_address_handler

Purpose:

Converts string addresses to their binary form in the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CREATE_SOCKET
Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "in convert server_addr" at "STATE_CONVERT_ADDRESS"
IF convert_address for proxy_addr, server_addr, client_addr, and gui_addr fails THEN
    RETURN STATE_ERROR
ENDIF
RETURN STATE_CREATE_SOCKET
```

create_socket_handler

Purpose:

Converts string addresses to their binary form in the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CREATE_SOCKET
Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "in create socket" at "STATE_CREATE_SOCKET"
IF socket_create for client_sockfd, server_sockfd, and proxy_gui_fd fails THEN
    RETURN STATE_ERROR
ENDIF

RETURN STATE_BIND_SOCKET
```

bind_socket_handler

Purpose:

Binds the created sockets to specified ports in the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_LISTEN
Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "in bind socket" at "STATE_BIND_SOCKET"
IF socket_bind for client_sockfd, server_sockfd, and proxy_gui_fd fails THEN
    RETURN STATE_ERROR
ENDIF


RETURN STATE_LISTEN
```

listen_handler

Purpose:

Puts the proxy GUI socket in listening mode in the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_LISTEN
Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "in start listening" at "STATE_START_LISTENING"
IF start_listening for proxy_gui_fd with SOMAXCONN fails THEN
   RETURN STATE_ERROR
ENDIF

RETURN STATE_CREATE_GUI_THREAD
```

create_gui_thread_handler

Purpose:

Creates a thread for handling the proxy GUI in the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CREATE_SERVER_THREAD
Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "STATE_CREATE_GUI_THREAD"
IF pthread_create for accept_gui_thread with init_gui_function fails THEN
   RETURN STATE_ERROR
ENDIF

RETURN STATE_CREATE_SERVER_THREAD
```

create_server_thread_handler

Purpose:

Creates a thread for handling server operations in the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CREATE_SERVER_THREAD
Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "in create receive thread" at "STATE_CREATE_RECV_THREAD"
IF pthread_create for server_thread with init_server_thread fails THEN
   RETURN STATE_ERROR
ENDIF

RETURN STATE_CREATE_KEYBOARD_THREAD
```

create_keyboard_thread_handler

Purpose:

Creates a thread for handling keyboard input within the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_LISTEN_CLIENT
Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "in create keyboard thread" at "STATE_CREATE_KEYBOARD_THREAD"
IF pthread_create for keyboard_thread with init_keyboard_thread fails THEN
    RETURN STATE_ERROR
ENDIF

RETURN STATE_LISTEN_CLIENT
```

listen_client_handler

Purpose:

Listens for packets from the client and handles them based on FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CLIENT_CALCULATE_LOSSINESS, STATE_CLEANUP

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "in connect socket" at "STATE_LISTEN_CLIENT"
WHILE exit_flag is not set DO
    IF receive_packet with client_sockfd and client_packet fails THEN
        RETURN STATE_ERROR
    ENDIF
    PRINT "Client packet received" with packet details
    IF is_connected_gui is true THEN
        CALL send_stats_gui with connected_gui_fd and RECEIVED_PACKET
    ENDIF
    RETURN STATE_CLIENT_CALCULATE_LOSSINESS
END WHILE
RETURN STATE_CLEANUP
```

calculate_client_losiness_handler

Purpose:

Calculates lossiness for a client packet and determines the next state.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CLIENT_DROP, STATE_CLIENT_DELAY_PACKET,

STATE_CLIENT_CORRUPT, STATE_SEND_CLIENT_PACKET

Pseudocode

```
SET_TRACE to "STATE_CLIENT_CALCULATE_LOSSINESS"
ASSIGN result to calculate_lossiness with client_drop_rate, client_delay_rate, and
corruption_rate
SWITCH on result
   CASE DROP: RETURN STATE_CLIENT_DROP
   CASE DELAY: RETURN STATE_CLIENT_DELAY_PACKET
   CASE CORRUPT: RETURN STATE_CLIENT_CORRUPT
   DEFAULT: RETURN STATE_SEND_CLIENT_PACKET
END SWITCH
```

client_drop_packet_handler

Purpose:

Handles dropping clients' packets.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_LISTEN_CLIENT

Pseudocode

```
SET_TRACE to "STATE_CLIENT_DROP"
PRINT "Client packet dropped" with packet details
IF is_connected_gui is true THEN
    CALL send_stats_gui with connected_gui_fd and DROPPED_CLIENT_PACKET
ENDIF

RETURN STATE_LISTEN_CLIENT
```

client_delay_packet_handler

Purpose:

Handles delaying of a client packet by creating a delay thread.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_LISTEN_CLIENT

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "STATE_CLIENT_DELAY_PACKET"
LOCK num_of_threads_mutex
INCREMENT num_of_threads
REALLOCATE thread_pool for new size
IF thread_pool allocation fails THEN
    UNLOCK mutex and RETURN STATE_ERROR
ENDIF
CREATE delay thread with init_client_delay_thread
UNLOCK num_of_threads_mutex
IF is_connected_gui is true THEN
    CALL send_stats_gui with connected_gui_fd and DELAYED_CLIENT_PACKET
ENDIF
RETURN STATE_LISTEN_CLIENT
```

client_corrupt_packet_handler

Purpose:

Handles corruption of a client packet's data.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_SEND_CLIENT_PACKET

Pseudocode

```
SET_TRACE to "STATE_CLIENT_CORRUPT"
IF client_packet's data length is 0 THEN
    RETURN STATE_SEND_CLIENT_PACKET
ENDIF
IF is_connected_gui is true THEN
    CALL send_stats_gui with connected_gui_fd and CORRUPTED_DATA
ENDIF
DUPLICATE client_packet's data to temp
CALL corrupt_data with temp and length of client_packet's data
COPY temp back to client_packet's data
PRINT "Client packet corrupted" with packet details
RETURN STATE_SEND_CLIENT_PACKET
```

send_client_packet_handler

Purpose:

Sends a packet from the client to the server within the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_LISTEN_CLIENT

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "STATE_SEND_CLIENT_PACKET"
IF send_packet with server_sockfd, client_packet, and server_addr_struct fails THEN
   RETURN STATE_ERROR
ENDIF
PRINT "Client packet sent" with packet details
IF is_connected_gui is true THEN
   CALL send_stats_gui with connected_gui_fd and SENT_PACKET
ENDIF

RETURN STATE_LISTEN_CLIENT
```

cleanup_handler

Purpose:

Cleans up resources when exiting the FSM.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: FSM_EXIT

Pseudocode

```
SET_TRACE to "in cleanup handler" at "STATE_CLEANUP"
JOIN server_thread
IF closing client_sockfd in ctx's args fails THEN
    PRINT "close socket error" for client socket
IF closing server_sockfd in ctx's args fails THEN
    PRINT "close socket error" for server socket
IF closing proxy_gui_fd in ctx's args fails THEN
    PRINT "close socket error" for proxy GUI socket
IF closing connected_gui_fd in ctx's args fails THEN
    PRINT "close socket error" for connected GUI socket
CLOSE sent_data file in ctx's args
CLOSE received_data file in ctx's args
RETURN FSM_EXIT
```

server_drop_handler

Purpose:

Listens for packets from the server and handles them within the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_SERVER_CALCULATE_LOSSINESS, FSM_EXIT

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "STATE_LISTEN_SERVER"
WHILE exit_flag is not set DO
  IF receive_packet with server_sockfd and server_packet fails THEN
    RETURN STATE_ERROR
  ENDIF
  PRINT "Server packet received" with packet details
  IF is_connected_gui is true THEN
    CALL send_stats_gui with connected_gui_fd and RECEIVED_PACKET
  ENDIF
  RETURN STATE_SERVER_CALCULATE_LOSSINESS
END WHILE
RETURN FSM_EXIT
```

server_delay_handler

Purpose:

Calculates lossiness for a server packet and determines the next FSM state.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_SERVER_DROP, STATE_SERVER_DELAY_PACKET, STATE_SERVER_CORRUPT, STATE_SEND_SERVER_PACKET

Pseudocode

```
SET_TRACE to "STATE_SERVER_CALCULATE_LOSSINESS"
ASSIGN result to calculate_lossiness with server_drop_rate, server_delay_rate, and
corruption_rate
SWITCH on result
   CASE DROP: RETURN STATE_SERVER_DROP
   CASE DELAY: RETURN STATE_SERVER_DELAY_PACKET
   CASE CORRUPT: RETURN STATE_SERVER_CORRUPT
   DEFAULT: RETURN STATE_SEND_SERVER_PACKET
END SWITCH
```

server_ drop_packet_handler

Purpose:

Handles dropping server’s packets.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_LISTEN_CLIENT

Pseudocode

```
SET_TRACE to "STATE_CLIENT_DROP"
PRINT "Client packet dropped" with packet details
IF is_connected_gui is true THEN
    CALL send_stats_gui with connected_gui_fd and DROPPED_SERVER_PACKET
ENDIF

RETURN STATE_LISTEN_CLIENT
```

server_delay_packet_handler

Purpose:

Handles delaying of a server packet by creating a delay thread.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_LISTEN_CLIENT

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "STATE_SERVER_DELAY_PACKET"
LOCK num_of_threads_mutex
INCREMENT num_of_threads
REALLOCATE thread_pool for new size
IF thread_pool allocation fails THEN
    UNLOCK mutex and RETURN STATE_ERROR
ENDIF
CREATE delay thread with init_client_delay_thread
UNLOCK num_of_threads_mutex
IF is_connected_gui is true THEN
    CALL send_stats_gui with connected_gui_fd and DELAYED_SERVER_PACKET
ENDIF
RETURN STATE_LISTEN_CLIENT
```

server_corrupt_packet_handler

Purpose:

Handles corruption of a server packet's data within the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_SEND_SERVER_PACKET

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE with no specific message
IF server_packet's data length is 0 THEN
   RETURN STATE_SEND_SERVER_PACKET
ENDIF
IF is_connected_gui is true THEN
   CALL send_stats_gui with connected_gui_fd and CORRUPTED_DATA
ENDIF
DUPLICATE server_packet's data to temp
CALL corrupt_data with temp and length of server_packet's data
COPY temp back to server_packet's data
PRINT "Server packet corrupted" with packet details
RETURN STATE_SEND_SERVER_PACKET
```

send_server_packet_handler

Purpose:

Sends a packet from the server to the client within the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_LISTEN_SERVER

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE to "STATE_SEND_SERVER_PACKET"
IF send_packet with client_sockfd, server_packet, and client_addr_struct fails THEN
    RETURN STATE_ERROR
ENDIF
IF is_connected_gui is true THEN
    CALL send_stats_gui with connected_gui_fd and SENT_PACKET
ENDIF
PRINT "Server packet sent" with packet details
RETURN STATE_LISTEN_SERVER
```

error_handler

Purpose:

Handles errors within the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CLEANUP

Pseudocode

```
PRINT error details using err structure
RETURN STATE_CLEANUP
```

read_from_keyboard_handler

Purpose:

Reads keyboard input to adjust network settings within the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: FSM_EXIT

Pseudocode

```
SET_TRACE to "STATE_READ_FROM_KEYBOARD"
WHILE exit_flag is not set DO
    CALL read_keyboard with pointers to client_drop_rate, client_delay_rate, etc.
END WHILE
RETURN FSM_EXIT
```

init_server_thread_handler

Purpose:

Initializes the FSM for server.

Parameters

void *ptr: Pointer to the FSM context.

Return

void: returns null after completion

Pseudocode

```
ASSIGN ctx to ptr casted as fsm_context
DECLARE err as fsm_error
DECLARE transitions array specific to each thread's functionality
CALL fsm_run with ctx, err, and transitions
RETURN NULL
```

init_keyboard_thread_handler

Purpose:

Initializes the FSM for keyboard input handling.

Parameters

void *ptr: Pointer to the FSM context.

Return

void: returns null after completion

Pseudocode

```
ASSIGN ctx to ptr casted as fsm_context
DECLARE err as fsm_error
DECLARE transitions array specific to each thread's functionality
CALL fsm_run with ctx, err, and transitions
RETURN NULL
```

init_client_delay_thread_handler

Purpose:

Initializes the FSM for keyboard input handling.

Parameters

void *ptr: Pointer to the FSM context.

Return

void: returns null after completion

Pseudocode

```
ASSIGN ctx to ptr casted as fsm_context
ASSIGN temp_packet to address of client_packet in ctx's args
PRINT "Client packet delayed" with packet details and DELAY_TIME
CALL delay_packet with DELAY_TIME
CALL send_packet with server_sockfd, temp_packet, and server_addr_struct in ctx's args
IF is_connected_gui is true in ctx's args THEN
    CALL send_stats_gui with connected_gui_fd in ctx's args and SENT_PACKET
ENDIF
PRINT "Client packet sent" with packet details
RETURN NULL
```

init_server_delay_thread_handler

Purpose:

Delays the sending of a server packet by a specified duration

Parameters

void *ptr: Pointer to the FSM context.

Return

void: returns null after completion

Pseudocode

```
ASSIGN ctx to ptr casted as fsm_context
ASSIGN temp_packet to address of server_packet in ctx's args
PRINT "Server packet delayed" with packet details and DELAY_TIME
CALL delay_packet with DELAY_TIME
CALL send_packet with client_sockfd, temp_packet, and client_addr_struct in ctx's args
IF is_connected_gui is true in ctx's args THEN
    CALL send_stats_gui with connected_gui_fd in ctx's args and SENT_PACKET
ENDIF
PRINT "Server packet sent" with packet details
RETURN NULL
```

init_gui_function

Purpose:

Handles the GUI connections for the proxy within the FSM context.

Parameters

void *ptr: Pointer to the FSM context.

Return

void: returns null after completion

Pseudocode

```
ASSIGN ctx to ptr casted as fsm_context
WHILE exit_flag is not set DO
   ASSIGN connected_gui_fd in ctx's args to result of socket_accept_connection with
proxy_gui_fd in ctx's args and &err
   INCREMENT is_connected_gui in ctx's args
END WHILE
RETURN NULL
```

fsm_run

Purpose:

Executes the finite state machine (FSM) by transitioning between states based on a set of defined transitions until reaching the exit state.

Parameters

struct fsm_context *context: Pointer to the FSM context, containing state and data for the FSM.
struct fsm_error *err: Pointer to a structure for error handling.
const struct fsm_transition transitions[]: Array of FSM transitions

Return

Success: 0

Pseudocode

```
DECLARE from_id as int, INITIALIZE to FSM_INIT
DECLARE to_id as int, INITIALIZE to FSM_USER_START
WHILE to_id is not FSM_EXIT DO
    DECLARE perform as fsm_state_func
    DECLARE next_id as int
    ASSIGN perform to the result of fsm_transition with context, from_id, to_id, and transitions
    IF perform is NULL THEN
    ENDIF
    ASSIGN from_id to to_id
    ASSIGN next_id to the result of calling perform with context and err
    ASSIGN to_id to next_id
END WHILE
RETURN 0
```

fsm_transition

Purpose:

Finds and returns the function to be executed for a specific state transition in the FSM.

Parameters

struct fsm_context *context: Pointer to the FSM context, containing state and data for the FSM.

int from_id: ID of the current state.

int to_id: ID of the next state.

const struct fsm_transition transitions[]: Array of FSM transitions

Return

Success: performs transition

Failure: NULL

Pseudocode

```
DECLARE transition as pointer to fsm_transition, ASSIGN to the first element of transitions
array
WHILE transition's from_id is not FSM_IGNORE DO
  IF transition's from_id is from_id AND transition's to_id is to_id THEN
    RETURN transition's perform function
  ENDIF
  INCREMENT transition to point to the next element in the transitions array
END WHILE
RETURN NULL
```

socket_create

Purpose:

Creates a socket with specified parameters.

Parameters

int domain: The domain of the socket (e.g., AF_INET).

int type: The type of the socket (e.g., SOCK_STREAM).

int protocol: The protocol to be used with the socket (usually 0 for default).

struct fsm_error *err: Error structure for error handling.

Return

Success: socket file descriptor

Failure: -1

Pseudocode

```
DECLARE sockfd as int
ASSIGN sockfd to socket call with domain, type, and protocol
IF sockfd is -1 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN sockfd
```

start_listening

Purpose:

Puts the socket in listening mode to listen for incoming connections.

Parameters

int sockfd: The socket file descriptor.

int backlog: The maximum length for the queue of pending connections.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
IF listen call with sockfd and backlog is -1 THEN
   SET error using strerror(errno)
   RETURN -1
ENDIF
RETURN 0
```

socket_accept_connection

Purpose:

Accepts a new connection on a socket

Parameters

int sockfd: The socket file descriptor.

struct fsm_error *err: Error structure for error handling.

Return

Success: file descriptor of the new socket

Failure: -1

Pseudocode

```
DECLARE client_addr as sockaddr
DECLARE client_addr_len as socklen_t, INITIALIZE to size of client_addr
DECLARE client_fd as int
SET errno to 0
ASSIGN client_fd to accept call with sockfd, client_addr, and client_addr_len
IF client_fd is -1 THEN
    IF errno is not EINTR THEN
        PRINT error message
    ENDIF
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN client_fd
```

socket_bind

Purpose:

Binds a socket to an IP address and port.

Parameters

int sockfd: Socket file descriptor.

struct sockaddr_storage *addr: Pointer to the address structure to bind the socket to.

struct fsm_error *err: Error structure for error handling

Return

Success: 0

Failure: -1

Pseudocode

```
ALLOCATE ip_address using safe_malloc for NI_MAXHOST
ALLOCATE port using safe_malloc for NI_MAXSERV
IF get_sockaddr_info with addr, &ip_address, &port, err is not 0 THEN
   RETURN -1
ENDIF
PRINT "binding to: ", ip_address, ":", port
IF bind call with sockfd, addr, and size_of_address(addr) is -1 THEN
   SET error using strerror(errno)
   RETURN -1
ENDIF
PRINT "Bound to socket: ", ip_address, ":", port
FREE ip_address
FREE port
RETURN 0
```

convert_address

Purpose:

Creates an IPv4 or IPv6 sockaddr based off the ip address and port passed in.

Parameters

const char *address: The IP address in string format.

struct sockaddr_storage *addr: Pointer to the address structure to store the result.

in_port_t port: The port number.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE addr_str as char array of size INET6_ADDRSTRLEN
DECLARE addr_len as socklen_t
DECLARE vaddr as void pointer
DECLARE net_port as in_port_t, ASSIGN to htons of port
IF inet_pton with AF_INET, address, and ipv4_addr's sin_addr is 1 THEN
    DECLARE ipv4_addr as pointer to sockaddr_in, ASSIGN to addr
    SET addr_len to size of ipv4_addr
    SET ipv4_addr's sin_port to net_port
    SET vaddr to ipv4_addr's sin_addr
    SET addr's ss_family to AF_INET
ELSE IF inet_pton with AF_INET6, address, and ipv6_addr's sin6_addr is 1 THEN
    DECLARE ipv6_addr as pointer to sockaddr_in6, ASSIGN to addr
    SET addr_len to size of ipv6_addr
    SET ipv6_addr's sin6_port to net_port
    SET vaddr to ipv6_addr's sin6_addr
    SET addr's ss_family to AF_INET6
ELSE
    SET error "Address family not supported"
    RETURN -1
RETURN 0
```

socket_close

Purpose:

Closes a socket

Parameters

int sockfd: The socket file descriptor.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
IF close call with sockfd is -1 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN 0
```

send_stats_gui

Purpose:

Sends statistical data to a GUI over a socket.

Parameters

int sockfd: Socket file descriptor.

int stat: The statistical data to send.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE result as ssize_t
ASSIGN result to write call with sockfd, address of stat, and size of stat
IF result is 0 or less THEN
    RETURN -1
ENDIF
RETURN 0
```

Functions For Server

parse_arguments

Purpose:

Parses command-line arguments for server and client addresses, ports, and window size. It handles errors and ensures each argument is passed only once.

Parameters

int argc: Number of command-line arguments.
char *argv[]: Array of command-line argument strings.
char **server_addr: Pointer to store the server address.
char **client_addr: Pointer to store the client address.
char **server_port_str: Pointer to store the server port string.
char **client_port_str: Pointer to store the client port string.
struct fsm_error *err: Pointer to error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE opt as int
DECLARE C_flag, c_flag, S_flag, s_flag, as bool, INITIALIZE to 0
DISABLE getopt error messages
WHILE parsing command-line arguments using getopt DO
  SWITCH opt
    CASE 'C': // Client address
      IF C_flag is true THEN
        RETURN -1
      ENDIF
      INCREMENT C_flag
      ASSIGN client address with optarg
    CASE 'c': // Client port
      IF c_flag is true THEN
```

```
            RETURN -1
        ENDIF
        INCREMENT c_flag
        ASSIGN client port string with optarg
    CASE 'S': // Server address
        IF S_flag is true THEN
            RETURN -1
        ENDIF
        INCREMENT S_flag
        ASSIGN server address with optarg
    CASE 's': // Server port
        IF s_flag is true THEN
            RETURN -1
        ENDIF
        INCREMENT s_flag
        ASSIGN server port string with optarg
    CASE 'h': // Help
        CALL usage and SET_ERROR
        RETURN -1
    CASE '?':
        CALL usage and SET_ERROR
        RETURN -1
    DEFAULT:
        CALL usage
  END SWITCH
END WHILE
RETURN 0
```

usage

Purpose:

Displays the usage information for the program, detailing the expected command-line arguments.

Parameters

const char *program_name: The name of the program.

Return

Success: STATE_LISTEN
Failure: STATE_ERROR

Pseudocode

```
PRINT "Usage: <program_name> [-C] <value> [-c] <value> [-S] <value> [-s] <value> [-h]" to
stderr
PRINT detailed options and their descriptions to stderr
```

handle_arguments

Purpose:

Validates the required command-line arguments for server and client addresses and ports, and window size. Sets errors if arguments are missing or invalid.

Parameters

const char *program_name: The name of the program.
const char *server_addr: Pointer to a string for storing the server address.
const char *client_addr: Pointer to a string for storing the client address.
const char *server_port_str: Pointer to a string for storing the server port as a string.
const char *client_port_str: Pointer to a string for storing the client port as a string.
in_port_t *server_port: Pointer to store the parsed server port.
in_port_t *client_port: Pointer to store the parsed client port.
struct fsm_error *err: Pointer to an error structure for handling and recording any errors that occur.

Return

Success: 0

Failure: -1

Pseudocode

```
IF server_addr is NULL THEN
    RETURN -1
IF client_addr is NULL THEN
    RETURN -1
IF server_port_str is NULL THEN
    RETURN -1
IF client_port_str is NULL THEN
    RETURN -1
CALL parse_in_port_t for server_port_str
IF error THEN RETURN -1
CALL parse_in_port_t for client_port_str
IF error THEN RETURN -1
RETURN 0
```

parse_in_port_t

Purpose:

Parses a string to an in_port_t type, validating the input.

Parameters

const char *binary_name: The name of the program.

const char *str: The string to parse.

in_port_t *port: Pointer to store the parsed value.

struct fsm_error *err: Error handling structure.

Return

Success: 0

Failure: -1

Pseudocode

```
PARSE str to a uintmax_t value
IF error occurred during parsing THEN
   RETURN -1
ENDIF
IF parsed_value is greater than UINT16_MAX THEN
   CALL usage
   RETURN -1
ENDIF
ASSIGN parsed_value to *port
RETURN 0
```

fsm_run

Purpose:

Executes the finite state machine (FSM) by transitioning between states based on a set of defined transitions until reaching the exit state.

Parameters

struct fsm_context *context: Pointer to the FSM context, containing state and data for the FSM.
struct fsm_error *err: Pointer to a structure for error handling.
const struct fsm_transition transitions[]: Array of FSM transitions

Return

Success: 0

Pseudocode

```
DECLARE from_id as int, INITIALIZE to FSM_INIT
DECLARE to_id as int, INITIALIZE to FSM_USER_START
WHILE to_id is not FSM_EXIT DO
    DECLARE perform as fsm_state_func
    DECLARE next_id as int
    ASSIGN perform to the result of fsm_transition with context, from_id, to_id, and transitions
    IF perform is NULL THEN
    ENDIF
    ASSIGN from_id to to_id
    ASSIGN next_id to the result of calling perform with context and err
    ASSIGN to_id to next_id
END WHILE
RETURN 0
```

fsm_transition

Purpose:

Finds and returns the function to be executed for a specific state transition in the FSM.

Parameters

struct fsm_context *context: Pointer to the FSM context, containing state and data for the FSM.

int from_id: ID of the current state.

int to_id: ID of the next state.

const struct fsm_transition transitions[]: Array of FSM transitions

Return

Success: performs transition

Failure: NULL

Pseudocode

```
DECLARE transition as pointer to fsm_transition, ASSIGN to the first element of transitions
array
WHILE transition's from_id is not FSM_IGNORE DO
   IF transition's from_id is from_id AND transition's to_id is to_id THEN
      RETURN transition's perform function
   ENDIF
   INCREMENT transition to point to the next element in the transitions array
END WHILE
RETURN NULL
```

main

Purpose:

Initializes structures and starts the finite state machine (FSM) for network communication.

Parameters

int argc: Count of command-line arguments.

char **argv: Array of command-line argument strings.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE err as fsm_error
DECLARE args as arguments with initial values NULL and 0 for head and is_buffered
DECLARE context as fsm_context with argc, argv, and address of args
DECLARE transitions as array of fsm_transition with predefined states and handlers
CALL fsm_run with address of context, address of err, and additional parameters
RETURN 0
```

parse_arguments_handler

Purpose:

Parses command-line arguments in the FSM context.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_HANDLE_ARGUMENTS

Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, descriptive message, and current state
IF CALL parse_arguments with ctx's argc, argv, and args returns non-zero THEN
    RETURN STATE_ERROR
ELSE
    RETURN STATE_HANDLE_ARGUMENTS
ENDIF
```

handle_arguments_handler

Purpose:

Processes parsed arguments to set up the application's configuration.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CONVERT_ADDRESS

Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, "in handle arguments", "STATE_HANDLE_ARGUMENTS"
IF CALL handle_arguments with argv[0], server_addr, client_addr, server_port_str,
client_port_str,
   address of server_port, address of client_port, window_size from ctx's args returns non-zero
THEN
   RETURN STATE_ERROR
ENDIF
IF CALL create_file with "../client_received_data.csv", address of received_data from ctx's args,
err returns -1 THEN
   RETURN STATE_ERROR
ENDIF
IF CALL create_file with "../client_sent_data.csv", address of sent_data from ctx's args, err
returns -1 THEN
   RETURN STATE_ERROR
ENDIF
RETURN STATE_CONVERT_ADDRESS
```

convert_address_handler

Purpose:

Processes parsed arguments to set up the application's configuration.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CREATE_SOCKET

Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, descriptive message, and current state
IF convert_address with server_addr, server_addr_struct, and server_port in ctx's args fails
THEN
   RETURN STATE_ERROR
ENDIF
IF convert_address with server_addr, gui_addr_struct, and fixed port 61000 in ctx's args fails
THEN
   RETURN STATE_ERROR
ENDIF
IF convert_address with client_addr, client_addr_struct, and client_port in ctx's args fails THEN
   RETURN STATE_ERROR
ENDIF
RETURN STATE_CREATE_SOCKET
```

create_socket_handler

Purpose:

Processes parsed arguments to set up the application's configuration.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_BIND_SOCKET

Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, descriptive message, and current state
SET ctx's sockfd with the result of CALL socket_create with family, type, and protocol from ctx
IF ctx's sockfd is -1 THEN
    RETURN STATE_ERROR
ELSE
    RETURN STATE_BIND_SOCKET
ENDIF
```

bind_socket_handler

Purpose:

Binds the created socket to a client address.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_CREATE_WINDOW

Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
CALL SET_TRACE with context, "in bind socket", "STATE_BIND_SOCKET"
IF binding sockfd with server_addr_struct in ctx's args fails THEN
   RETURN STATE_ERROR
ENDIF
IF binding server_gui_fd with gui_addr_struct in ctx's args fails THEN
   RETURN STATE_ERROR
ENDIF

RETURN STATE_LISTEN
```

listen_handler

Purpose:

Listens for incoming packets and processes them

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: STATE_READ_FROM_KEYBOARD

Failure: STATE_ERROR

Pseudocode

```
DECLARE ctx as pointer to fsm_context from context
DECLARE result as ssize_t
CALL SET_TRACE with context, empty message, "STATE_LISTEN_SERVER"
IF start_listening with server_gui_fd in ctx's args and SOMAXCONN fails THEN
    RETURN STATE_ERROR
ENDIF
RETURN STATE_CREATE_GUI_THREAD
```

create_gui_thread_handler

Purpose:

Initializes the new thread to handle GUI interaction within FSM context.

Parameters

struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

void: NULL upon completion

Pseudocode

```
SET_TRACE
CREATE a new thread accept_gui_thread in ctx's args to run init_gui_function with ctx
IF thread creation result is negative THEN
    RETURN STATE_ERROR
ENDIF
RETURN STATE_WAIT
```

wait_handler

Purpose:

Listens for incoming packets and processes them in the FSM context.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CLEANUP, STATE_COMPARE_CHECKSUM

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE with "STATE_LISTEN_SERVER"
WHILE exit_flag is not set DO
    CALL receive_packet with sockfd, temp_packet, received_data, and err from ctx's args
    IF result is -1 THEN
        RETURN STATE_ERROR
    ENDIF
    IF is_connected_gui in ctx's args THEN
        CALL send_stats_gui with connected_gui_fd and RECEIVED_PACKET
    ENDIF
    RETURN STATE_COMPARE_CHECKSUM
END WHILE
RETURN STATE_CLEANUP
```

compare_checksum

Purpose:

Compares the checksum of the received packet to verify its integrity.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CHECK_SEQ_NUMBER

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE with "STATE_COMPARE_CHECKSUM"
IF compare_checksum with temp_packet's checksum and data is TRUE THEN
    RETURN STATE_CHECK_SEQ_NUMBER
ELSE
    IF is_connected_gui in ctx's args THEN
        CALL send_stats_gui with connected_gui_fd and DROPPED_CLIENT_PACKET
    ENDIF
    RETURN STATE_WAIT
ENDIF
```

check_seq_number_handler

Purpose:

Checks the sequence number of the packet to determine its processing order.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CHECK_SEQ_NUMBER

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE with "STATE_CHECK_SEQ_NUMBER"
IF check_seq_number with temp_packet's seq_number and expected_seq_number is TRUE
THEN
   IF temp_packet's flags is SYN THEN
      RETURN STATE_SEND_SYN_ACK
   ELSE
      RETURN STATE_SEND_PACKET
   ENDIF
ELSE
   IF is_connected_gui in ctx's args THEN
      CALL send_stats_gui with connected_gui_fd and DROPPED_CLIENT_PACKET
   ENDIF
   RETURN STATE_WAIT
ENDIF
```

send_syn_ack_handler

Purpose:

Sends a SYN-ACK packet in response to a received SYN packet.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CHECK_SEQ_NUMBER

Pseudocode

```
SET_TRACE with "STATE_START_HANDSHAKE"
INCREMENT is_handshake_ack in ctx's args
PRINT handshake ack count
CALL create_syn_ack_packet with sockfd, client_addr_struct, temp_packet, sent_data, and err
CALL send_packet with sockfd, client_addr_struct, temp_packet, sent_data, and err
IF is_connected_gui in ctx's args THEN
    CALL send_stats_gui with connected_gui_fd and SENT_PACKET
ENDIF
RETURN STATE_UPDATE_SEQ_NUMBER
```

create_timer_handler

Purpose:

Creates a timer thread for packet timeout handling.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_WAIT_FOR_ACK

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE with "STATE_CREATE_TIMER_THREAD"
INCREMENT num_of_threads in ctx's args
REALLOCATE thread_pool in ctx's args for new thread count
IF temp_thread_pool is NULL THEN
    RETURN STATE_ERROR
ENDIF
CREATE a new thread in thread_pool with init_timer_function and ctx
RETURN STATE_WAIT_FOR_ACK
```

wait_for_ack_handler

Purpose:

Waits for an ACK packet, verifying handshake completion or processing received packets.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_WAIT, STATE_CLEANUP

Failure: STATE_ERROR

Pseudocode

```
ASSIGN ctx to context
SET_TRACE with "STATE_WAIT_FOR_ACK"
WHILE exit_flag is not set DO
    CALL receive_packet with sockfd, temp_packet, received_data, and err
    IF result is -1 THEN
        RETURN STATE_ERROR
    IF is_connected_gui in ctx's args THEN
        CALL send_stats_gui with connected_gui_fd and RECEIVED_PACKET
    ENDIF
    IF temp_packet's flags is ACK AND seq_number equals expected_seq_number THEN
        RESET is_handshake_ack in ctx's args
        RETURN STATE_WAIT
    IF seq_number is less than expected_seq_number THEN
        CALL read_received_packet with appropriate parameters
END WHILE
RETURN STATE_CLEANUP
```

send_packet_handler

Purpose:

Processes received packets, sending responses or updating state as needed.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_WAIT, STATE_UPDATE_SEQ_NUMBER

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE with "STATE_SEND_PACKET"
CALL read_received_packet with sockfd, client_addr_struct, temp_packet, sent_data, and err
from ctx's args
IF is_connected_gui in ctx's args THEN
   CALL send_stats_gui with connected_gui_fd and SENT_PACKET
ENDIF
IF seq_number of temp_packet is less than expected_seq_number THEN
   RETURN STATE_WAIT
ENDIF
RETURN STATE_UPDATE_SEQ_NUMBER
```

update_seq_num_handler

Purpose:

Updates the expected sequence number based on the type of packet received.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_WAIT, STATE_CREATE_TIMER_THREAD

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE with "STATE_UPDATE_SEQ_NUMBER"
IF flags of temp_packet is SYN THEN
   UPDATE expected_seq_number with seq_number and 1
   RETURN STATE_WAIT
ENDIF
IF flags of temp_packet is SYNACK THEN
   UPDATE expected_seq_number with ack_number and 0
   RETURN STATE_CREATE_TIMER_THREAD
ENDIF
UPDATE expected_seq_number with seq_number and data length of temp_packet
RETURN STATE_WAIT
```

cleanup_handler

Purpose:

Handles cleanup of resources when FSM is exiting.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_WAIT, STATE_CREATE_TIMER_THREAD

Failure: STATE_ERROR

Pseudocode

```
SET_TRACE with "in cleanup handler" at "STATE_CLEANUP"
JOIN accept_gui_thread in ctx's args
CLOSE sockfd in ctx's args and handle errors
CLOSE server_gui_fd in ctx's args and handle errors
CLOSE connected_gui_fd in ctx's args and handle errors
CLOSE sent_data and received_data files in ctx's args
RETURN FSM_EXIT
```

error_handler

Purpose:

Handles errors and transitions FSM to the cleanup state.

Parameters

struct fsm_context *context: Context of the FSM.

struct fsm_error *err: Error structure for error handling.

Return

Success: STATE_CLEANUP

Pseudocode

```
PRINT error details from err structure
RETURN STATE_CLEANUP
```

init_timer_function

Purpose:

A function for a timer thread to handle retransmissions and timeouts.

Parameters

void *ptr: Pointer to the FSM context.

Return

None

Pseudocode

```
WHILE not exit_flag or is_handshake_ack is set DO
    SLEEP for TIMER_TIME
    IF is_handshake_ack is set THEN
        CALL send_packet with sockfd, client_addr_struct, packet_to_send, and sent_data
        IF is_connected_gui THEN
            CALL send_stats_gui with connected_gui_fd and RESENT_PACKET
        ENDIF
        INCREMENT counter
    ENDIF
END WHILE
CALL pthread_exit with NULL
```

init_gui_function(thread)

Purpose:

Continuously listens for GUI connections until an exit condition is met.

Parameters

void *ptr: Pointer to FSM context.

Return

void: NULL upon completion

Pseudocode

```
DECLARE ctx as pointer to fsm_context from ptr
DECLARE err as fsm_error
WHILE exit_flag is not true DO
    ASSIGN ctx's args connected_gui_fd with CALL socket_accept_connection with client_gui_fd from ctx's args and address of err
    INCREMENT ctx's args is_connected_gui
END WHILE
RETURN NULL
```

create_file

Purpose:

Creates and opens a file for writing, handling file opening errors.

Parameters

const char *filepath: Path to the file to be created.
struct fsm_context *context: FSM context.
struct fsm_error *err: Error handling structure.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE fp as pointer to FILE, ASSIGN with CALL fopen with filepath and "w" mode
IF fp is NULL THEN
    CALL SET_ERROR with err and "Error in opening file."
    RETURN -1
ENDIF
ASSIGN value at fp to *fp
RETURN 0
```

send_packet

Purpose:

Sends a packet using a socket and updates the window buffer.

Parameters

int sockfd: The socket file descriptor.

struct sockaddr_storage *addr: Pointer to the address structure.

struct sent_packet *window: Pointer to the window buffer.

struct packet *pt: Pointer to the packet to be sent.

FILE *fp: File pointer for logging.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE result as ssize_t
ASSIGN result to sendto call with sockfd, pt, and address information
IF result is equak -1 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
CALL write_stats_to_file with fp and pt
RETURN 0
```

receive_packet

Purpose:

Receives a packet from a socket and updates the window state.

Parameters

int sockfd: Socket file descriptor.

struct sent_packet *window: Pointer to the window buffer.

struct packet *pt: Pointer to the packet to be populated with received data.

FILE *fp: File pointer for statistics logging.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE client_addr as sockaddr_storage
DECLARE client_addr_len as socklen_t, INITIALIZE to size of client_addr
DECLARE pt as packet
DECLARE result as ssize_t
ASSIGN result to recvfrom call with sockfd, pt, and client address
IF result is -1 THEN
   SET error using strerror(errno)
   RETURN -1
ENDIF
ASSIGN temp_pt to *pt
CALL write_stats_to_file with fp and pt
RETURN 0
```

create_second_handshake_seq_number

Purpose:

Generates second sequence number for the second handshake.

Parameters

void

Return

The new sequence number

Pseudocode

```
RETURN 100
```

create_sequence_number

Purpose:

Generates a new sequence number based on the previous sequence number and data size.

Parameters

uint32_t prev_seq_number: The previous sequence number.

uint32_t data_size: The size of the data.

Return

The new sequence number

Pseudocode

```
RETURN prev_seq_number + data_size
```

create_ack_number

Purpose:

Generates an acknowledgment number based on the received sequence number and data size.

Parameters

uint32_t recv_seq_number: The received sequence number.

uint32_t data_size: The size of the data.

Return

The acknowledgment number.

Pseudocode

```
RETURN recv_seq_number + data_size
```

check_seq_number

Purpose:

Check if the sequence number of a packet is equal to or less than the expected sequence number.

Parameters

uint32_t seq_number: The sequence number to check.

uint32_t expected_seq_number: The expected sequence number.

Return

Success: TRUE
Failure: FALSE

Pseudocode

```
IF seq_number and expected_seq_number  are equal OR seq_number is less than
expected_seq_number
        RETURN TRUE
ELSE
        RETURN FALSE
```

check_if_equal

Purpose:

Checks if two sequence numbers are equal.

Parameters

uint32_t seq_number: The sequence number to check.

uint32_t expected_seq_number: The expected sequence number.

Return

Success: TRUE
Failure: FALSE

Pseudocode

```
IF seq_number equals expected_seq_number
    RETURN TRUE
ELSE
    RETURN FALSE
```

check_if_less

Purpose:

Checks if the sequence number is less than the expected sequence number.

Parameters

uint32_t seq_number: The sequence number to check.

uint32_t expected_seq_number: The expected sequence number.

Return

Success: TRUE
Failure: FALSE

Pseudocode

```
IF seq_number less than expected_seq_number
    RETURN TRUE
ELSE
    RETURN FALSE
```

update_expected_seq_number

Purpose:

Updates the expected sequence number based on the current sequence number and the size of the data.

Parameters

uint32_t seq_number: The current sequence number.

uint32_t data_size: The size of the data.

Return

The updated expected sequence number

Pseudocode

```
PRINT "expected: ", seq_number plus data_size
RETURN seq_number plus data_size
```

write_stats_to_file

Purpose:

Writes packet statistics to a file.

Parameters

FILE *fp: File pointer where the statistics will be written.

const struct packet *pt: Pointer to the packet whose statistics are to be written.

Return

Success : 0

Pseudocode

```
WRITE pt's header seq_number, ack_number, flags, window_size, checksum, and data to fp
FLUSH the file stream fp
RETURN
```

read_received_packet

Purpose:

Processes a received packet based on its flags and performs appropriate actions.

Parameters

int sockfd: Socket file descriptor.

struct sockaddr_storage *addr: Pointer to the address structure.

struct sent_packet *window: Pointer to the window buffer.

struct packet *pt: Pointer to the received packet.

FILE *fp: File pointer for logging.

struct fsm_error *err: Error structure for error handling

Return

Success: 0
Failure: -1

Pseudocode

```
SWITCH on result
   CASE ESTABLISH_HANDSHAKE:
      CALL send_syn_ack_packet with sockfd, addr, window, pt, fp, err
      BREAK
   CASE SEND_HANDSHAKE_ACK:
      CALL send_handshake_ack_packet with sockfd, addr, window, pt, fp, err
      BREAK
   CASE SEND_ACK:
      CALL send_data_ack_packet with sockfd, addr, window, pt, fp, err
      BREAK
   CASE END_CONNECTION:
      CALL recv_termination_request with sockfd, addr, window, pt, fp, err
      BREAK
   CASE RECV_RST, UNKNOWN_FLAG, default:
      RETURN -1
RETURN 0
```

read_flags

Purpose:

Interprets the flags of a packet and returns the corresponding action.

Parameters

uint8_t flags: Flags of the packet.

Return

The action should be taken in integer format.

Pseudocode

```
IF flags is SYN THEN
   RETURN ESTABLISH_HANDSHAKE
ENDIF
IF flags is SYNACK THEN
   RETURN SEND_HANDSHAKE_ACK
ENDIF
IF flags is PSHACK THEN
   RETURN SEND_ACK
ENDIF
IF flags is ACK THEN
   RETURN RECV_ACK
ENDIF
IF flags is FINACK THEN
   RETURN END_CONNECTION
ENDIF
IF flags is RSTACK THEN
   RETURN RECV_RST
ENDIF
RETURN UNKNOWN_FLAG
```

send_syn_ack_packet

Purpose:

Sends a SYN-ACK packet as a response in the handshake process.

Parameters

int sockfd: The socket file descriptor used for network communication.
struct sockaddr_storage *addr: Pointer to the address structure of the sender or receiver.
struct sent_packet *window: Pointer to the window buffer used for managing sent packets.
struct packet *pt: Pointer to the packet structure that has been received.
FILE *fp: File pointer used for logging packet information.
struct fsm_error *err: Pointer to an error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to CALL create_second_handshake_seq_number
SET packet_to_send's header ack_number to CALL create_ack_number with pt.hd.seq_number
SET packet_to_send's header flags to CALL create_flags with pt.hd.flags
SET packet_to_send's header window_size to global window_size
CLEAR packet_to_send's data
CALL calculate_checksum with &packet_to_send.hd.checksum, packet_to_send.data
CALL send_packet with sockfd, addr, window, &packet_to_send, fp, err
CALL add_packet_to_window with window, &packet_to_send
RETURN 0
```

create_syn_ack_packet

Purpose:

Creates a syn packet for handshake.

Parameters

int sockfd: The socket file descriptor used for network communication.
struct sockaddr_storage *addr: Pointer to the address structure of the sender or receiver.
struct packet *pt: Pointer to the packet structure that has been received.
FILE *fp: File pointer used for logging packet information.
struct fsm_error *err: Pointer to an error structure for error handling.

Return

success: 0

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to sequence number from previous seq_number and data_size
SET packet_to_send's header ack_number to ack number from previous ack_number
SET packet_to_send's header flags to PSHACK
SET packet_to_send's header window_size to global window_size
ASSIGN packet_to_send to *pt
RETURN 0
```

send_handshake_ack_packet

Purpose:

Sends a handshake acknowledgment packet.

Parameters

int sockfd: The socket file descriptor used for network communication.
struct sockaddr_storage *addr: Pointer to the address structure of the sender or receiver.
struct sent_packet *window: Pointer to the window buffer used for managing sent packets.
struct packet *pt: Pointer to the packet structure that has been received.
FILE *fp: File pointer used for logging packet information.
struct fsm_error *err: Pointer to an error structure for error handling.

Return

Success: 0

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to sequence number from pt's ack_number
SET packet_to_send's header ack_number to ack number from pt's seq_number plus 1
SET packet_to_send's header flags to flags from pt's flags
SET packet_to_send's header window_size to global window_size
CLEAR packet_to_send's data
CALL calculate_checksum with packet_to_send's header checksum and data
CALL send_packet with sockfd, addr, window, &packet_to_send, fp, err
CALL add_packet_to_window with window and &packet_to_send
RETURN 0
```

send_data_ack_packet

Purpose:

Sends an acknowledgment packet for received data.

Parameters

int sockfd: The socket file descriptor used for network communication.
struct sockaddr_storage *addr: Pointer to the address structure of the sender or receiver.
struct sent_packet *window: Pointer to the window buffer used for managing sent packets.
struct packet *pt: Pointer to the packet structure that has been received.
FILE *fp: File pointer used for logging packet information.
struct fsm_error *err: Pointer to an error structure for error handling.

Return

Success: 0

Pseudocode

```
DECLARE packet_to_send as packet
SET packet_to_send's header seq_number to sequence number from previous seq_number and data_size
SET packet_to_send's header ack_number to ack number from pt's seq_number and length of pt's data
SET packet_to_send's header flags to flags from pt's flags
SET packet_to_send's header window_size to global window_size
CLEAR packet_to_send's data
CALL calculate_checksum with packet_to_send's header checksum and data
CALL send_packet with sockfd, addr, window, &packet_to_send, fp, err
CALL add_packet_to_window with window and &packet_to_send
RETURN 0
```

create_flags

Purpose:

Determines the appropriate flags for a response packet based on the received packet's flags.

Parameters

uint8_t flags: Flags from the received packet.

Return

New flag for the response packet

Pseudocode

```
IF flags is SYN THEN
   RETURN SYNACK
ELSE IF flags is SYNACK THEN
   RETURN ACK
ELSE IF flags is PSHACK THEN
   RETURN ACK
ELSE IF flags is FINACK THEN
   RETURN ACK
ELSE
   RETURN UNKNOWN_FLAG
ENDIF
```

calculate_checksum

Purpose:

Calculates a checksum for given data.

Parameters

uint16_t *checksum: Pointer to store the calculated checksum.

const char *data: Data for which the checksum is calculated.

size_t length: Length of the data.

Return

success: 0

Pseudocode

```
ASSIGN to *checksum the product of checksum_one and checksum_two results with data and length
RETURN 0
```

checksum_one

Purpose:

Calculates the first part of the checksum.

Parameters

const char *data: Data for which the checksum is calculated.

size_t length: Length of the data.

Return

The calculated checksum value

Pseudocode

```
DECLARE result as unsigned char, INITIALIZE to 0
FOR EACH byte in data up to length DO
    INCREMENT result by data[i] multiplied by 34
END FOR
RETURN result
```

checksum_two

Purpose:

Calculates the second part of the checksum.

Parameters

const char *data: Data for which the checksum is calculated.

size_t length: Length of the data.

Return

The calculated checksum value

Pseudocode

```
DECLARE result as unsigned char, INITIALIZE to 0
FOR EACH byte in data up to length DO
   XOR result with data[i]
END FOR

RETURN result
```

compare_checksum

Purpose:

Compares the provided checksum with a newly calculated checksum for given data to verify data integrity.

Parameters

uint16_t checksum: The original checksum to compare against.

const char *data: Pointer to the data for which the checksum is calculated.

size_t length: The length of the data.

Return

Success: TRUE
Failure: FALSE

Pseudocode

```
DECLARE new_checksum as uint16_t
CALL calculate_checksum with address of new_checksum, data, and length

RETURN new_checksum equals checksum (TRUE or FALSE)
```

socket_create

Purpose:

Creates a socket with specified parameters.

Parameters

int domain: The domain of the socket (e.g., AF_INET).

int type: The type of the socket (e.g., SOCK_STREAM).

int protocol: The protocol to be used with the socket (usually 0 for default).

struct fsm_error *err: Error structure for error handling.

Return

Success: socket file descriptor

Failure: -1

Pseudocode

```
DECLARE sockfd as int
ASSIGN sockfd to socket call with domain, type, and protocol
IF sockfd is -1 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN sockfd
```

start_listening

Purpose:

Puts the socket in listening mode to listen for incoming connections.

Parameters

int sockfd: The socket file descriptor.

int backlog: The maximum length for the queue of pending connections.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
IF listen call with sockfd and backlog is -1 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN 0
```

socket_accept_connection

Purpose:

Accepts a new connection on a socket

Parameters

int sockfd: The socket file descriptor.

struct fsm_error *err: Error structure for error handling.

Return

Success: file descriptor of the new socket

Failure: -1

Pseudocode

```
DECLARE client_addr as sockaddr
DECLARE client_addr_len as socklen_t, INITIALIZE to size of client_addr
DECLARE client_fd as int
SET errno to 0
ASSIGN client_fd to accept call with sockfd, client_addr, and client_addr_len
IF client_fd is -1 THEN
    IF errno is not EINTR THEN
        PRINT error message
    ENDIF
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN client_fd
```

socket_bind

Purpose:

Binds a socket to an IP address and port.

Parameters

int sockfd: Socket file descriptor.

struct sockaddr_storage *addr: Pointer to the address structure to bind the socket to.

struct fsm_error *err: Error structure for error handling

Return

Success: 0

Failure: -1

Pseudocode

```
ALLOCATE ip_address using safe_malloc for NI_MAXHOST
ALLOCATE port using safe_malloc for NI_MAXSERV
IF get_sockaddr_info with addr, &ip_address, &port, err is not 0 THEN
    RETURN -1
ENDIF
PRINT "binding to: ", ip_address, ":", port
IF bind call with sockfd, addr, and size_of_address(addr) is -1 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
PRINT "Bound to socket: ", ip_address, ":", port
FREE ip_address
FREE port
RETURN 0
```

convert_address

Purpose:

Creates an IPv4 or IPv6 sockaddr based off the ip address and port passed in.

Parameters

const char *address: The IP address in string format.

struct sockaddr_storage *addr: Pointer to the address structure to store the result.

in_port_t port: The port number.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE addr_str as char array of size INET6_ADDRSTRLEN
DECLARE addr_len as socklen_t
DECLARE vaddr as void pointer
DECLARE net_port as in_port_t, ASSIGN to htons of port
IF inet_pton with AF_INET, address, and ipv4_addr's sin_addr is 1 THEN
    DECLARE ipv4_addr as pointer to sockaddr_in, ASSIGN to addr
    SET addr_len to size of ipv4_addr
    SET ipv4_addr's sin_port to net_port
    SET vaddr to ipv4_addr's sin_addr
    SET addr's ss_family to AF_INET
ELSE IF inet_pton with AF_INET6, address, and ipv6_addr's sin6_addr is 1 THEN
    DECLARE ipv6_addr as pointer to sockaddr_in6, ASSIGN to addr
    SET addr_len to size of ipv6_addr
    SET ipv6_addr's sin6_port to net_port
    SET vaddr to ipv6_addr's sin6_addr
    SET addr's ss_family to AF_INET6
ELSE
    SET error "Address family not supported"
    RETURN -1
RETURN 0
```

socket_close

Purpose:

Closes a socket

Parameters

int sockfd: The socket file descriptor.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

Failure: -1

Pseudocode

```
IF close call with sockfd is -1 THEN
    SET error using strerror(errno)
    RETURN -1
ENDIF
RETURN 0
```

size_of_address

Purpose:

Determines the size of an address structure based on its sa_family.

Parameters

struct sockaddr_storage *addr: Pointer to the address structure.

Return

Success: Size of the address structure

Pseudocode

```
IF addr's ss_family is AF_INET
        RETURNsizeof(struct sockaddr_in) :
ELSE
        sizeof(struct sockaddr_in6)
```

get_sockaddr_info

Purpose:

Retrieves IP address and port information from a sockaddr_storage structure.

Parameters

struct sockaddr_storage *addr: Pointer to the address structure.

char **ip_address: Pointer to store the IP address string.

char **port: Pointer to store the port string.

struct fsm_error *err: Error structure for error handling.

Return

Success: 0

failure: -1

Pseudocode

```
DECLARE temp_ip as char array of NI_MAXHOST
DECLARE temp_port as char array of NI_MAXSERV
DECLARE ip_size as socklen_t, ASSIGN to size of *addr
DECLARE result as int
ASSIGN result to getnameinfo with addr, ip_size, temp_ip, temp_port, and flags
NI_NUMERICHOST | NI_NUMERICSERV
IF result is not 0 THEN
   SET error using strerror(errno)
   RETURN -1
ENDIF
COPY temp_ip to *ip_address
COPY temp_port to *port
RETURN 0
```

safe_malloc

Purpose:

Safely allocates memory and checks for allocation failure.

Parameters

uint32_t size: Size of memory to allocate.

struct fsm_error *err: Error structure for error handling.

Return

void*: Pointer to the allocated memory.

Pseudocode

```
DECLARE ptr as void pointer
ALLOCATE memory to ptr with size
IF ptr is NULL and size is greater than 0 THEN
    PRINT error message
    EXIT program with EXIT_FAILURE
ENDIF
RETURN ptr
```

send_stats_gui

Purpose:

Sends statistical data to a GUI over a socket.

Parameters

int sockfd: Socket file descriptor.

int stat: The statistical data to send.

Return

Success: 0

Failure: -1

Pseudocode

```
DECLARE result as ssize_t
ASSIGN result to write call with sockfd, address of stat, and size of stat
IF result is 0 or less THEN
    RETURN -1
ENDIF
RETURN 0
```

Functions For GUI

connect_to_server

Purpose:

Establishes a connection to a specified server and processes incoming data packets.

Parameters

server_id: Identifier for the server.
host: IP address of the server.
port: Port number of the server.
data: Shared data structure for storing packet information.

Return

None

Pseudocode

```
CREATE a new socket client_socket
TRY
    CONNECT client_socket to host and port
    LOOP INFINITELY
      READ a message from client_socket
      IF message is received THEN
        UNPACK the message into value
        PRINT received packet information
        APPEND value and current time to data[server_id]
HANDLE KeyboardInterrupt
    PASS
HANDLE ConnectionRefusedError
    PRINT connection error message
    CLOSE client_socket
```

update_plot

Purpose:

Updates the plot for a specific server with received packet data.

Parameters

i: Frame index for the animation (unused).

ax: Matplotlib axis object for plotting.

server_id: Identifier for the server.

data: Shared data structure with packet information.

Return

None

Pseudocode

```
IF data for server_id is not empty THEN
    CLEAR the axis ax
    EXTRACT times and values from data[server_id]
    INITIALIZE packet_count_per_type and packet_history
    FOR each packet value and time
        INCREMENT count in packet_count_per_type
        APPEND count to packet_history for each packet type
    PLOT each packet type history on ax with respective color
    SET labels and title for ax
    ADD legend to ax
ENDIF
```

start_plot

Purpose:

Initializes the plotting process for a specific server.

Parameters

server_id: Identifier for the server.

data: Shared data structure with packet information.

Return

None

Pseudocode

```
CREATE a Matplotlib figure and axis
SET ani to FuncAnimation updating using update_plot
DISPLAY the plot
```

start

Purpose:

Starts the network communication and plotting processes using multiprocessing.

Parameters

data: Shared data structure for storing packet information.

Return

None

Pseudocode

```
DEFINE server_descriptions for Server, Client, and Proxy
INITIALIZE an empty list processes
FOR each description in server_descriptions
    CREATE a process for connect_to_server and start it
    APPEND the process to processes
END FOR
FOR i in range 3
    CREATE a plot process for start_plot and start it
    APPEND the plot process to processes
END FOR
JOIN all processes in processes
```

main

Purpose:

main execution block

Parameters

None

Return

None

Pseudocode

```
SET multiprocessing start method to 'spawn'
CREATE a multiprocessing manager
INITIALIZE shared data structure data using the manager
CALL start with data
```