

# **COMP 7005**

## **Project**

## **Design**

Sami Roudgarian, A01294122

Harmanbir Dhillon, A00994245

December 1, 2023

Client Main Table.....	30
Client Receive Table.....	34
Proxy Main Table.....	36
Proxy Server Losiness Table.....	40
Proxy Keyboard Table.....	42
Server Main Table.....	43
Functions For Client.....	47
main.....	47
Purpose.....	47
Parameters.....	47
Return.....	47
Pseudocode.....	47
parse_arguments_handler.....	48
Purpose.....	48
Parameters.....	48
Return.....	48
Pseudocode.....	48
handle_arguments_handler.....	49
Purpose.....	49
Parameters.....	49
Return.....	49
Pseudocode.....	49
convert_address_handler.....	50
Purpose.....	50
Parameters.....	50
Return.....	50
Pseudocode.....	50
create_socket_handler.....	51
Purpose.....	51
Parameters.....	51
Return.....	51
Pseudocode.....	51
bind_socket_handler.....	52
Purpose.....	52
Parameters.....	52
Return.....	52
Pseudocode.....	52
create_window_handler.....	53
Purpose.....	53

Parameters.....	53
Return.....	53
Pseudocode.....	53
start_handshake_handler.....	54
Purpose.....	54
Parameters.....	54
Return.....	54
Pseudocode.....	54
create_handshake_timer_handler.....	55
Purpose.....	55
Parameters.....	55
Return.....	55
Pseudocode.....	55
wait_for_syn_ack_handler.....	56
Purpose.....	56
Parameters.....	56
Return.....	56
Pseudocode.....	56
send_handshake_ack_handler.....	57
Purpose.....	57
Parameters.....	57
Return.....	57
Pseudocode.....	57
create_rcv_thread_handler.....	58
Purpose.....	58
Parameters.....	58
Return.....	58
Pseudocode.....	58
read_from_keyboard_handler.....	59
Purpose.....	59
Parameters.....	59
Return.....	59
Pseudocode.....	59
check_window_handler.....	60
Purpose.....	60
Parameters.....	60
Return.....	60
Pseudocode.....	60
add_packet_to_buffer_handler.....	61
Purpose.....	61

Parameters.....	61
Return.....	61
Pseudocode.....	61
add_packet_to_window_handler.....	62
Purpose.....	62
Parameters.....	62
Return.....	62
Pseudocode.....	62
check_window_thread_handler.....	63
Purpose.....	63
Parameters.....	63
Return.....	63
Pseudocode.....	63
send_message_handler.....	64
Purpose.....	64
Parameters.....	64
Return.....	64
Pseudocode.....	64
create_new_timer_thread_handler.....	65
Purpose.....	65
Parameters.....	65
Return.....	65
Pseudocode.....	65
cleanup_handler.....	66
Purpose.....	66
Parameters.....	66
Return.....	66
Pseudocode.....	66
error_handler.....	67
Purpose.....	67
Parameters.....	67
Return.....	67
Pseudocode.....	67
listen_handler.....	68
Purpose.....	68
Parameters.....	68
Return.....	68
Pseudocode.....	68
check_ack_number_handler.....	69
Purpose.....	69

Parameters.....	69
Return.....	69
Pseudocode.....	69
remove_packet_from_window_handler.....	70
Purpose.....	70
Parameters.....	70
Return.....	70
Pseudocode.....	70
send_packet_handler.....	71
Purpose.....	71
Parameters.....	71
Return.....	71
Pseudocode.....	71
init_rcv_function(thread).....	72
Purpose.....	72
Parameters.....	72
Return.....	72
Pseudocode.....	72
init_timer_function (thread).....	73
Purpose.....	73
Parameters.....	73
Return.....	73
Pseudocode.....	73
init_window_checker_function (thread).....	74
Purpose.....	74
Parameters.....	74
Return.....	74
Pseudocode.....	74
init_gui_function(thread).....	75
Purpose.....	75
Parameters.....	75
Return.....	75
Pseudocode.....	75
create_file.....	76
Purpose.....	76
Parameters.....	76
Return.....	76
Pseudocode.....	76
parse_arguments.....	77
Purpose.....	77

Parameters.....	77
Return.....	77
Pseudocode.....	77
usage.....	79
Purpose.....	79
Parameters.....	79
Return.....	79
Pseudocode.....	79
handle_arguments.....	80
Purpose.....	80
Parameters.....	80
Return.....	80
Pseudocode.....	80
parse_in_port_t.....	81
Purpose.....	81
Parameters.....	81
Return.....	81
Pseudocode.....	81
convert_to_int.....	82
Purpose.....	82
Parameters.....	82
Return.....	82
Pseudocode.....	82
fsm_run.....	83
Purpose.....	83
Parameters.....	83
Return.....	83
Pseudocode.....	83
fsm_transition.....	84
Purpose.....	84
Parameters.....	84
Return.....	84
Pseudocode.....	84
init_list.....	85
Purpose.....	85
Parameters.....	85
Return.....	85
Pseudocode.....	85
push.....	86
Purpose.....	86

Parameters.....	86
Return.....	86
Pseudocode.....	86
pop.....	87
Purpose.....	87
Parameters.....	87
Return.....	87
Pseudocode.....	87
create_window.....	88
Purpose.....	88
Parameters.....	88
Return.....	88
Pseudocode.....	88
window_empty.....	89
Purpose.....	89
Parameters.....	89
Return.....	89
Pseudocode.....	89
first_packet_ring_buffer.....	90
Purpose.....	90
Parameters.....	90
Return.....	90
Pseudocode.....	90
first_unacked_ring_buffer.....	91
Purpose.....	91
Parameters.....	91
Return.....	91
Pseudocode.....	91
send_packet.....	92
Purpose.....	92
Parameters.....	92
Return.....	92
Pseudocode.....	92
add_packet_to_window.....	93
Purpose.....	93
Parameters.....	93
Return.....	93
Pseudocode.....	93
receive_packet.....	94
Purpose.....	94

Parameters.....	94
Return.....	94
Pseudocode.....	94
remove_single_packet.....	95
Purpose.....	95
Parameters.....	95
Return.....	95
Pseudocode.....	95
remove_cumulative_packets.....	96
Purpose.....	96
Parameters.....	96
Return.....	96
Pseudocode.....	96
remove_lesser_index.....	97
Purpose.....	97
Parameters.....	97
Return.....	97
Pseudocode.....	97
remove_greater_index.....	98
Purpose.....	98
Parameters.....	98
Return.....	98
Pseudocode.....	98
remove_packet_from_window.....	99
Purpose.....	99
Parameters.....	99
Return.....	99
Pseudocode.....	99
is_window_empty.....	100
Purpose.....	100
Parameters.....	100
Return.....	100
Pseudocode.....	100
get_expected_ack_number.....	101
Purpose.....	101
Parameters.....	101
Return.....	101
Pseudocode.....	101
create_second_handshake_seq_number.....	102
Purpose.....	102



Parameters.....	102
Return.....	102
Pseudocode.....	102
create_sequence_number.....	103
Purpose.....	103
Parameters.....	103
Return.....	103
Pseudocode.....	103
create_ack_number.....	104
Purpose.....	104
Parameters.....	104
Return.....	104
Pseudocode.....	104
previous_seq_number.....	105
Purpose.....	105
Parameters.....	105
Return.....	105
Pseudocode.....	105
previous_date_size.....	106
Purpose.....	106
Parameters.....	106
Return.....	106
Pseudocode.....	106
previous_ack_number.....	107
Purpose.....	107
Parameters.....	107
Return.....	107
Pseudocode.....	107
check_ack_number.....	108
Purpose.....	108
Parameters.....	108
Return.....	108
Pseudocode.....	108
check_ack_number_equal.....	109
Purpose.....	109
Parameters.....	109
Return.....	109
Pseudocode.....	109
check_ack_number_greater.....	110
Purpose.....	110

Parameters.....	110
Return.....	110
Pseudocode.....	110
get_ack_number_index.....	111
Purpose.....	111
Parameters.....	111
Return.....	111
Pseudocode.....	111
previous_index.....	112
Purpose.....	112
Parameters.....	112
Return.....	112
Pseudocode.....	112
write_stats_to_file.....	113
Purpose.....	113
Parameters.....	113
Return.....	113
Pseudocode.....	113
read_received_packet.....	114
Purpose.....	114
Parameters.....	114
Return.....	114
Pseudocode.....	114
read_flags.....	115
Purpose.....	115
Parameters.....	115
Return.....	115
Pseudocode.....	115
send_syn_packet.....	116
Purpose.....	116
Parameters.....	116
Return.....	116
Pseudocode.....	116
send_syn_ack_packet.....	117
Purpose.....	117
Parameters.....	117
Return.....	117
Pseudocode.....	117
send_handshake_ack_packet.....	118
Purpose.....	118

Parameters.....	118
Return.....	118
Pseudocode.....	118
send_data_packet.....	119
Purpose.....	119
Parameters.....	119
Return.....	119
Pseudocode.....	119
send_data_ack_packet.....	120
Purpose.....	120
Parameters.....	120
Return.....	120
Pseudocode.....	120
create_flags.....	121
Purpose.....	121
Parameters.....	121
Return.....	121
Pseudocode.....	121
create_data_packet.....	122
Purpose.....	122
Parameters.....	122
Return.....	122
Pseudocode.....	122
create_handshake_ack_packet.....	123
Purpose.....	123
Parameters.....	123
Return.....	123
Pseudocode.....	123
calculate_checksum.....	124
Purpose.....	124
Parameters.....	124
Return.....	124
Pseudocode.....	124
checksum_one.....	125
Purpose.....	125
Parameters.....	125
Return.....	125
Pseudocode.....	125
checksum_two.....	126
Purpose.....	126

Parameters.....	126
Return.....	126
Pseudocode.....	126
socket_create.....	127
Purpose.....	127
Parameters.....	127
Return.....	127
Pseudocode.....	127
read_keyboard.....	128
Purpose.....	128
Parameters.....	128
Return.....	128
Pseudocode.....	128
start_listening.....	129
Purpose.....	129
Parameters.....	129
Return.....	129
Pseudocode.....	129
socket_accept_connection.....	130
Purpose.....	130
Parameters.....	130
Return.....	130
Pseudocode.....	130
socket_close.....	131
Purpose.....	131
Parameters.....	131
Return.....	131
Pseudocode.....	131
convert_address.....	132
Purpose.....	132
Parameters.....	132
Return.....	132
Pseudocode.....	132
socket_bind.....	133
Purpose.....	133
Parameters.....	133
Return.....	133
Pseudocode.....	133
size_of_address.....	134
Purpose.....	134

Parameters.....	134
Return.....	134
Pseudocode.....	134
get_sockaddr_info.....	135
Purpose.....	135
Parameters.....	135
Return.....	135
Pseudocode.....	135
safe_malloc.....	136
Purpose.....	136
Parameters.....	136
Return.....	136
Pseudocode.....	136
send_stats_gui.....	137
Purpose.....	137
Parameters.....	137
Return.....	137
Pseudocode.....	137
<b>Functions For Proxy.....</b>	<b>138</b>
parse_arguments.....	138
Purpose.....	138
Parameters.....	138
Return.....	138
Pseudocode.....	138
usage.....	140
Purpose.....	140
Parameters.....	140
Return.....	140
Pseudocode.....	140
handle_arguments.....	141
Purpose.....	141
Parameters.....	141
Return.....	141
Pseudocode.....	141
parse_in_port_t.....	142
Purpose.....	142
Parameters.....	142
Return.....	142
Pseudocode.....	142
convert_to_int.....	143

Purpose.....	143
Parameters.....	143
Return.....	143
Pseudocode.....	143
random_number.....	144
Purpose.....	144
Parameters.....	144
Return.....	144
Pseudocode.....	144
calculate_lossiness.....	145
Purpose:.....	145
Parameters.....	145
Return.....	145
Pseudocode.....	145
calculate_drop.....	146
Purpose:.....	146
Parameters.....	146
uint8_t percentage: Percentage rate for drop.....	146
Return.....	146
Pseudocode.....	146
calculate_delay.....	147
Purpose:.....	147
Parameters.....	147
Return.....	147
Pseudocode.....	147
calculate_corruption.....	148
Purpose:.....	148
Parameters.....	148
Return.....	148
Pseudocode.....	148
send_packet.....	149
Purpose:.....	149
Sends a packet over a socket.....	149
Parameters.....	149
Return.....	149
Pseudocode.....	149
receive_packet.....	150
Purpose:.....	150
Parameters.....	150

Return.....	150
Pseudocode.....	150
delay_packet.....	151
Purpose:.....	151
Parameters.....	151
Return.....	151
Pseudocode.....	151
read_keyboard.....	152
Purpose:.....	152
Parameters.....	152
Return.....	152
Pseudocode.....	152
read_menu.....	154
Purpose:.....	154
Parameters.....	154
Return.....	154
Pseudocode.....	154
corrupt_data.....	155
Purpose:.....	155
Parameters.....	155
Return.....	155
Pseudocode.....	155
write_stats_to_file.....	156
Purpose:.....	156
Parameters.....	156
Return.....	156
Pseudocode.....	156
parse_argument_handler.....	157
Purpose:.....	157
Parameters.....	157
Return.....	157
Pseudocode.....	157
handle_argument_handler.....	158
Purpose:.....	158
Parameters.....	158
Return.....	158
Pseudocode.....	158
convert_address_handler.....	159

Purpose:.....	159
Parameters.....	159
Return.....	159
Pseudocode.....	159
create_socket_handler.....	160
Purpose:.....	160
Parameters.....	160
Return.....	160
Pseudocode.....	160
bind_socket_handler.....	161
Purpose:.....	161
Parameters.....	161
Return.....	161
Pseudocode.....	161
listen_handler.....	162
Purpose:.....	162
Parameters.....	162
Return.....	162
Pseudocode.....	162
create_gui_thread_handler.....	163
Purpose:.....	163
Parameters.....	163
Return.....	163
Pseudocode.....	163
create_server_thread_handler.....	164
Purpose:.....	164
Parameters.....	164
Return.....	164
Pseudocode.....	164
create_keyboard_thread_handler.....	165
Purpose:.....	165
Parameters.....	165
Return.....	165
Pseudocode.....	165
listen_client_handler.....	166
Purpose:.....	166
Parameters.....	166
Return.....	166



Pseudocode.....	166
calculate_client_lossiness_handler.....	167
Purpose:.....	167
Parameters.....	167
Return.....	167
Pseudocode.....	167
client_drop_packet_handler.....	168
Purpose:.....	168
Parameters.....	168
Return.....	168
Pseudocode.....	168
client_delay_packet_handler.....	169
Purpose:.....	169
Parameters.....	169
Return.....	169
Pseudocode.....	169
client_corrupt_packet_handler.....	170
Purpose:.....	170
Parameters.....	170
Return.....	170
Pseudocode.....	170
send_client_packet_handler.....	171
Purpose:.....	171
Parameters.....	171
Return.....	171
Pseudocode.....	171
cleanup_handler.....	172
Purpose:.....	172
Parameters.....	172
Return.....	172
Pseudocode.....	172
server_drop_handler.....	173
Purpose:.....	173
Parameters.....	173
Return.....	173
Pseudocode.....	173
server_delay_handler.....	174
Purpose:.....	174

Parameters.....	174
Return.....	174
Pseudocode.....	174
server_drop_packet_handler.....	175
Purpose:.....	175
Parameters.....	175
Return.....	175
Pseudocode.....	175
server_delay_packet_handler.....	176
Purpose:.....	176
Parameters.....	176
Return.....	176
Pseudocode.....	176
server_corrupt_packet_handler.....	177
Purpose:.....	177
Parameters.....	177
Return.....	177
Pseudocode.....	177
send_server_packet_handler.....	178
Purpose:.....	178
Parameters.....	178
Return.....	178
Pseudocode.....	178
error_handler.....	179
Purpose:.....	179
Parameters.....	179
Return.....	179
Pseudocode.....	179
read_from_keyboard_handler.....	180
Purpose:.....	180
Parameters.....	180
Return.....	180
Pseudocode.....	180
init_server_thread_handler.....	181
Purpose:.....	181
Parameters.....	181
Return.....	181
Pseudocode.....	181

init_keyboard_thread_handler.....	182
Purpose:.....	182
Parameters.....	182
Return.....	182
Pseudocode.....	182
init_client_delay_thread_handler.....	183
Purpose:.....	183
Parameters.....	183
Return.....	183
Pseudocode.....	183
init_server_delay_thread_handler.....	184
Purpose:.....	184
Parameters.....	184
Return.....	184
Pseudocode.....	184
init_gui_function.....	185
Purpose:.....	185
Parameters.....	185
Return.....	185
Pseudocode.....	185
fsm_run.....	186
Purpose:.....	186
Parameters.....	186
Return.....	186
Pseudocode.....	186
fsm_transition.....	187
Purpose:.....	187
Parameters.....	187
Return.....	187
Pseudocode.....	187
socket_create.....	188
Purpose:.....	188
Parameters.....	188
Return.....	188
Pseudocode.....	188
start_listening.....	189
Purpose:.....	189
Parameters.....	189

Return.....	189
Pseudocode.....	189
socket_accept_connection.....	190
Purpose:.....	190
Parameters.....	190
Return.....	190
Pseudocode.....	190
socket_bind.....	191
Purpose:.....	191
Parameters.....	191
Return.....	191
Pseudocode.....	191
convert_address.....	192
Purpose:.....	192
Parameters.....	192
Return.....	192
Pseudocode.....	192
socket_close.....	193
Purpose:.....	193
Parameters.....	193
Return.....	193
Pseudocode.....	193
send_stats_gui.....	194
Purpose:.....	194
Parameters.....	194
Return.....	194
Pseudocode.....	194
<b>Functions For Server.....</b>	<b>195</b>
parse_arguments.....	195
Purpose:.....	195
Parameters.....	195
Return.....	195
Pseudocode.....	195
usage.....	197
Purpose:.....	197
Parameters.....	197
Return.....	197
Pseudocode.....	197

handle_arguments.....	198
Purpose:.....	198
Parameters.....	198
Return.....	198
Pseudocode.....	198
parse_in_port_t.....	199
Purpose:.....	199
Parameters.....	199
Return.....	199
Pseudocode.....	199
fsm_run.....	200
Purpose:.....	200
Parameters.....	200
Return.....	200
Pseudocode.....	200
fsm_transition.....	201
Purpose:.....	201
Parameters.....	201
Return.....	201
Pseudocode.....	201
main.....	202
Purpose:.....	202
Parameters.....	202
Return.....	202
Pseudocode.....	202
parse_arguments_handler.....	203
Purpose:.....	203
Parameters.....	203
Return.....	203
Pseudocode.....	203
handle_arguments_handler.....	204
Purpose:.....	204
Parameters.....	204
Return.....	204
Pseudocode.....	204
convert_address_handler.....	205
Purpose:.....	205
Parameters.....	205

Return.....	205
Pseudocode.....	205
create_socket_handler.....	206
Purpose:.....	206
Parameters.....	206
Return.....	206
Pseudocode.....	206
bind_socket_handler.....	207
Purpose:.....	207
Parameters.....	207
Return.....	207
Pseudocode.....	207
listen_handler.....	208
Purpose:.....	208
Parameters.....	208
Return.....	208
Pseudocode.....	208
create_gui_thread_handler.....	209
Purpose:.....	209
Parameters.....	209
Return.....	209
Pseudocode.....	209
wait_handler.....	210
Purpose:.....	210
Parameters.....	210
Return.....	210
Pseudocode.....	210
compare_checksum.....	211
Purpose:.....	211
Parameters.....	211
Return.....	211
Pseudocode.....	211
check_seq_number_handler.....	212
Purpose:.....	212
Parameters.....	212
Return.....	212
Pseudocode.....	212
send_syn_ack_handler.....	213

Purpose:.....	213
Parameters.....	213
Return.....	213
Pseudocode.....	213
create_timer_handler.....	214
Purpose:.....	214
Parameters.....	214
Return.....	214
Pseudocode.....	214
wait_for_ack_handler.....	215
Purpose:.....	215
Parameters.....	215
Return.....	215
Pseudocode.....	215
send_packet_handler.....	216
Purpose:.....	216
Parameters.....	216
Return.....	216
Pseudocode.....	216
update_seq_num_handler.....	217
Purpose:.....	217
Parameters.....	217
Return.....	217
Pseudocode.....	217
cleanup_handler.....	218
Purpose:.....	218
Parameters.....	218
Return.....	218
Pseudocode.....	218
error_handler.....	219
Purpose:.....	219
Parameters.....	219
Return.....	219
Pseudocode.....	219
init_timer_function.....	220
Purpose:.....	220
Parameters.....	220
Return.....	220

Pseudocode.....	220
init_gui_function(thread).....	221
Purpose:.....	221
Parameters.....	221
Return.....	221
Pseudocode.....	221
create_file.....	222
Purpose:.....	222
Parameters.....	222
Return.....	222
Pseudocode.....	222
send_packet.....	223
Purpose:.....	223
Parameters.....	223
Return.....	223
Pseudocode.....	223
receive_packet.....	224
Purpose:.....	224
Parameters.....	224
Return.....	224
Pseudocode.....	224
create_second_handshake_seq_number.....	225
Purpose:.....	225
Parameters.....	225
Return.....	225
Pseudocode.....	225
create_sequence_number.....	226
Purpose:.....	226
Parameters.....	226
Return.....	226
Pseudocode.....	226
create_ack_number.....	227
Purpose:.....	227
Parameters.....	227
Return.....	227
Pseudocode.....	227
check_seq_number.....	228
Purpose:.....	228



Parameters.....	228
Return.....	228
Pseudocode.....	228
check_if_equal.....	229
Purpose:.....	229
Parameters.....	229
Return.....	229
Pseudocode.....	229
check_if_less.....	230
Purpose:.....	230
Parameters.....	230
Return.....	230
Pseudocode.....	230
update_expected_seq_number.....	231
Purpose:.....	231
Parameters.....	231
Return.....	231
Pseudocode.....	231
write_stats_to_file.....	232
Purpose:.....	232
Parameters.....	232
Return.....	232
Pseudocode.....	232
read_received_packet.....	233
Purpose:.....	233
Parameters.....	233
Return.....	233
Pseudocode.....	233
read_flags.....	234
Purpose:.....	234
Parameters.....	234
Return.....	234
Pseudocode.....	234
send_syn_ack_packet.....	235
Purpose:.....	235
Parameters.....	235
Return.....	235
Pseudocode.....	235

create_syn_ack_packet.....	236
Purpose:.....	236
Parameters.....	236
Return.....	236
Pseudocode.....	236
send_handshake_ack_packet.....	237
Purpose:.....	237
Parameters.....	237
Return.....	237
Pseudocode.....	237
send_data_ack_packet.....	238
Purpose:.....	238
Parameters.....	238
Return.....	238
Pseudocode.....	238
create_flags.....	239
Purpose:.....	239
Parameters.....	239
Return.....	239
Pseudocode.....	239
calculate_checksum.....	240
Purpose:.....	240
Parameters.....	240
Return.....	240
Pseudocode.....	240
checksum_one.....	241
Purpose:.....	241
Parameters.....	241
Return.....	241
Pseudocode.....	241
checksum_two.....	242
Purpose:.....	242
Parameters.....	242
Return.....	242
Pseudocode.....	242
compare_checksum.....	243
Purpose:.....	243
Parameters.....	243

Return.....	243
Pseudocode.....	243
socket_create.....	244
Purpose:.....	244
Parameters.....	244
Return.....	244
Pseudocode.....	244
start_listening.....	245
Purpose:.....	245
Parameters.....	245
Return.....	245
Pseudocode.....	245
socket_accept_connection.....	246
Purpose:.....	246
Parameters.....	246
Return.....	246
Pseudocode.....	246
socket_bind.....	247
Purpose:.....	247
Parameters.....	247
Return.....	247
Pseudocode.....	247
convert_address.....	248
Purpose:.....	248
Parameters.....	248
Return.....	248
Pseudocode.....	248
socket_close.....	249
Purpose:.....	249
Parameters.....	249
Return.....	249
Pseudocode.....	249
size_of_address.....	250
Purpose:.....	250
Parameters.....	250
Return.....	250
Pseudocode.....	250
get_sockaddr_info.....	251

Purpose:.....	251
Parameters.....	251
Return.....	251
Pseudocode.....	251
safe_malloc.....	252
Purpose:.....	252
Parameters.....	252
Return.....	252
Pseudocode.....	252
send_stats_gui.....	253
Purpose:.....	253
Parameters.....	253
Return.....	253
Pseudocode.....	253
<b>Functions For GUI.....</b>	<b>254</b>
connect_to_server.....	254
Purpose:.....	254
Parameters.....	254
Return.....	254
Pseudocode.....	254
update_plot.....	255
Purpose:.....	255
Parameters.....	255
Return.....	255
Pseudocode.....	255
start_plot.....	256
Purpose:.....	256
Parameters.....	256
Return.....	256
Pseudocode.....	256
start.....	257
Purpose:.....	257
Parameters.....	257
Return.....	257
Pseudocode.....	257
main.....	258
Purpose:.....	258
Parameters.....	258

Return.....	258
Pseudocode.....	258

# 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

<b>From</b>	<b>To</b>	<b>Handler</b>
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	-

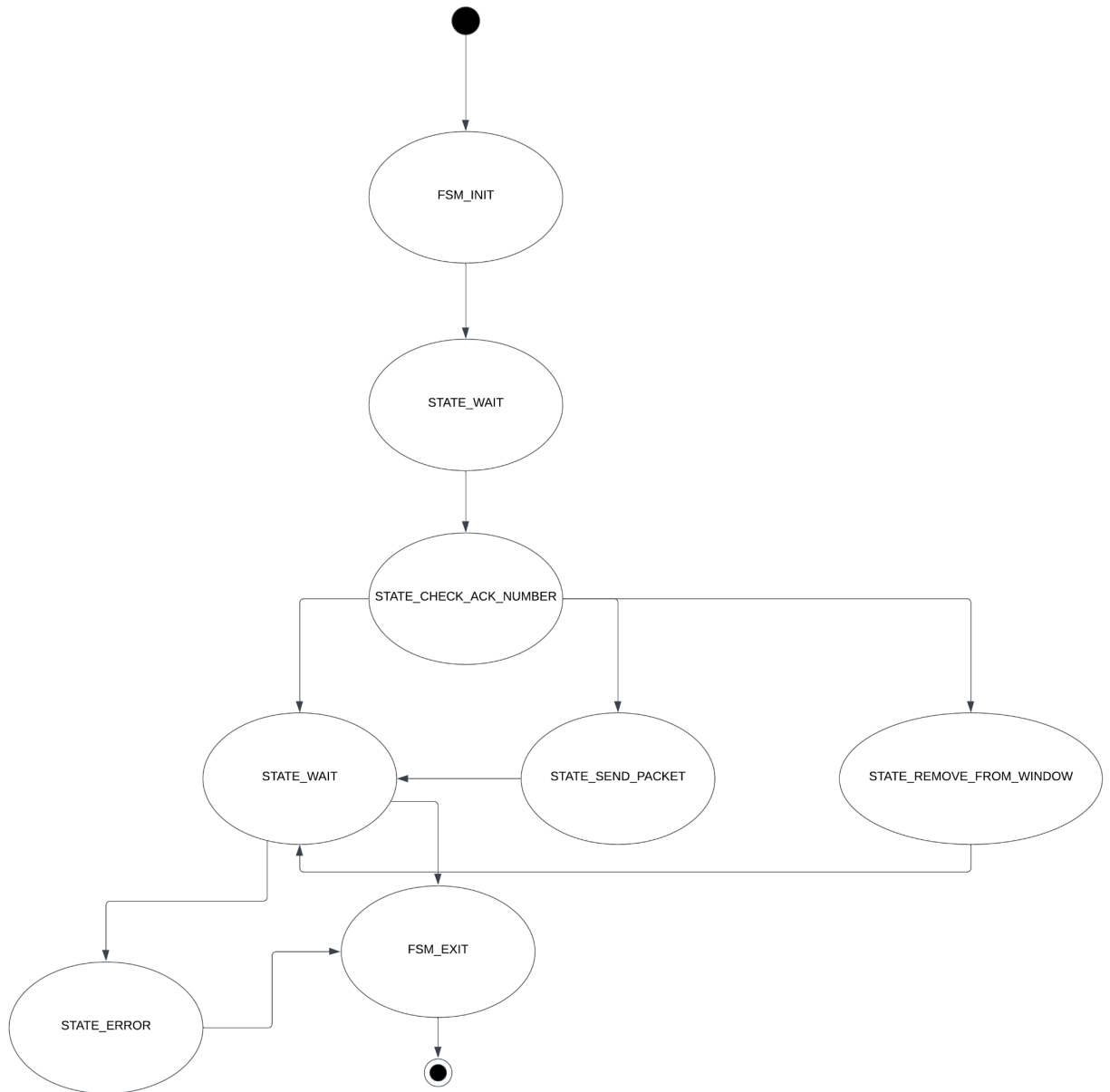






# Client Receive Table

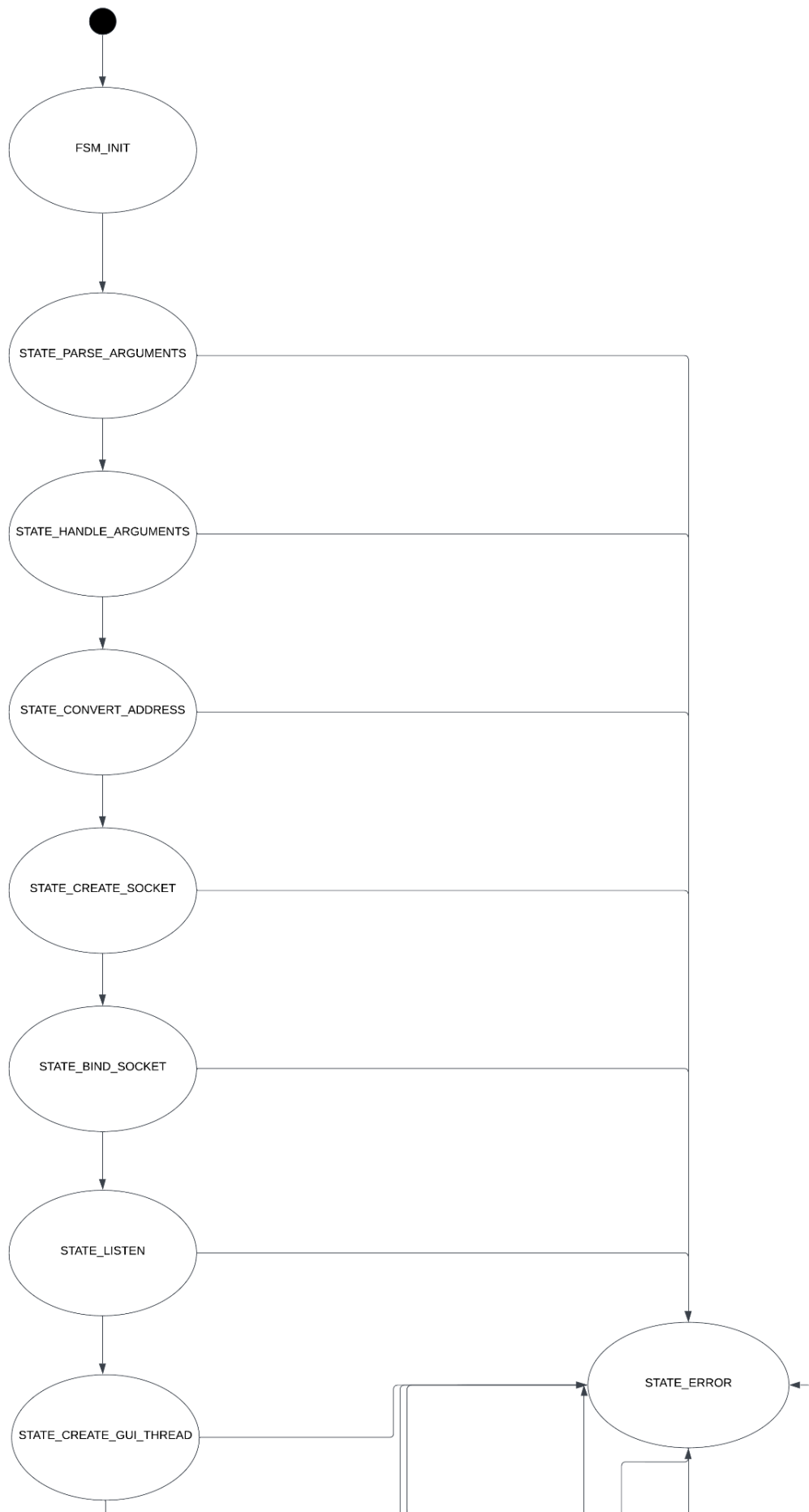
From	To	Handler
FSM_INIT	STATE_WAIT	wait_handler
STATE_WAIT	STATE_CHECK_ACK_NUMBER,	check_ack_number_handler
STATE_CHECK_ACK_NUMBER	STATE_REMOVE_FROM_WINDOW,	remove_packet_from_window_h andler
STATE_CHECK_ACK_NUMBER	STATE_SEND_PACKET	send_packet_handler
STATE_CHECK_ACK_NUMBER	STATE_WAIT	wait_handler
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	-



# 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

<b>From</b>	<b>To</b>	<b>Handler</b>
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	-

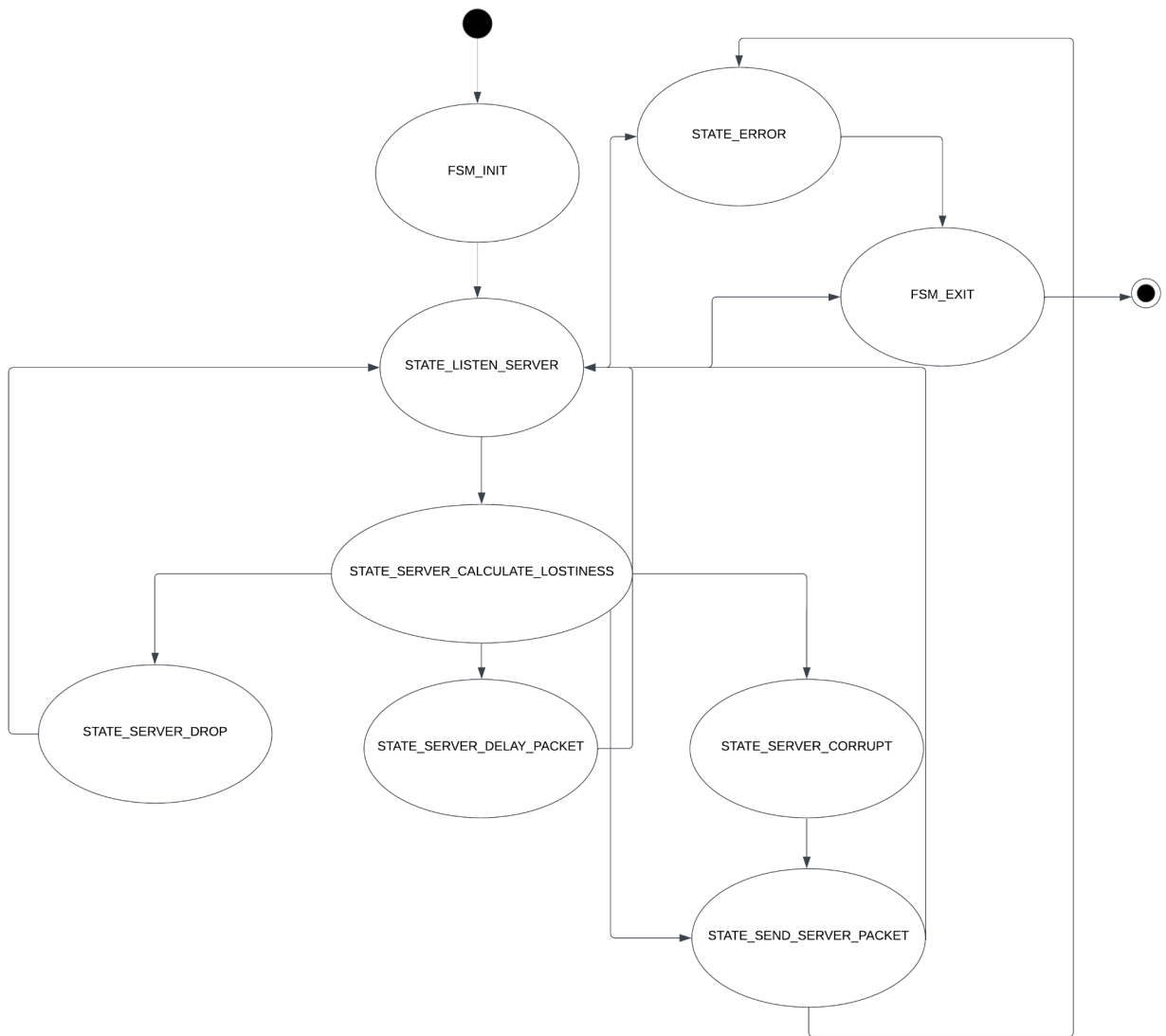




# Proxy Server Lossiness Table

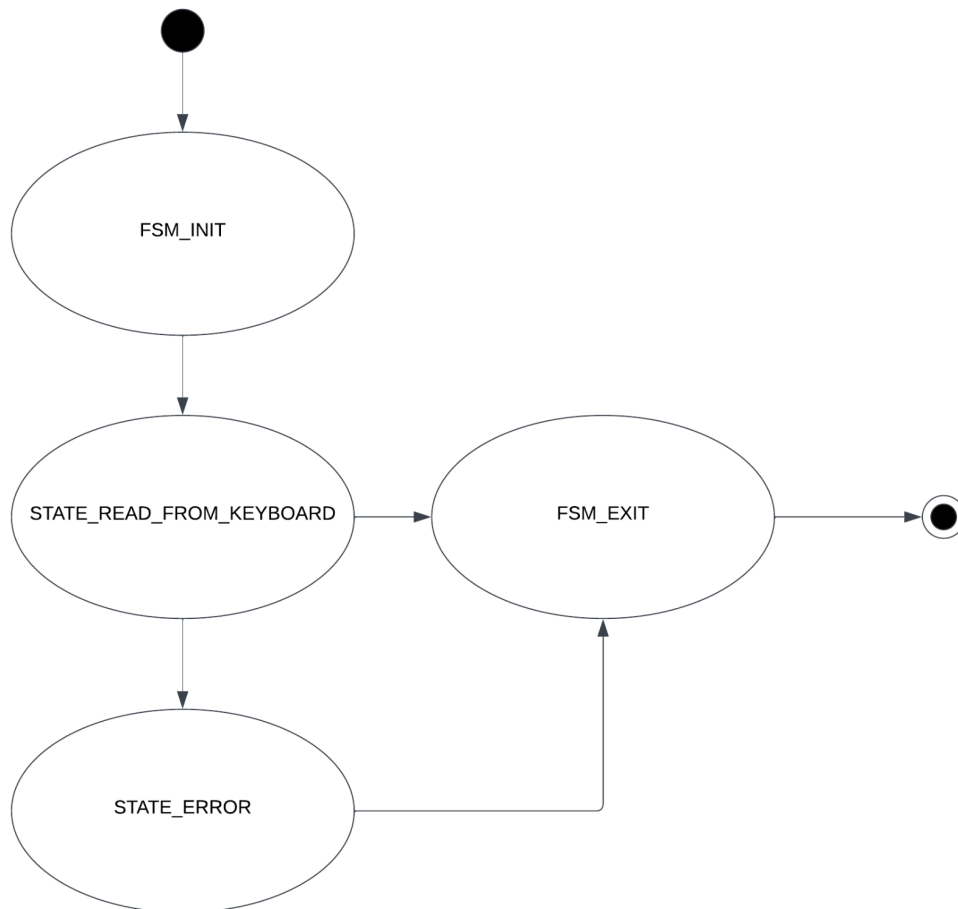
From	To	Handler
FSM_INIT	STATE_LISTEN_SERVER	listen_server_handler
STATE_LISTEN_SERVER	STATE_SERVER_CALCULATE_LOSSINESS	calculate_server_lossiness_handler
STATE_LISTEN_SERVER	STATE_ERROR	error_handler
STATE_SERVER_CALCULATE_LOSSINESS	STATE_SERVER_DROP	server_drop_packet_handler
STATE_SERVER_CALCULATE_LOSSINESS	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	-





# Proxy Keyboard Table

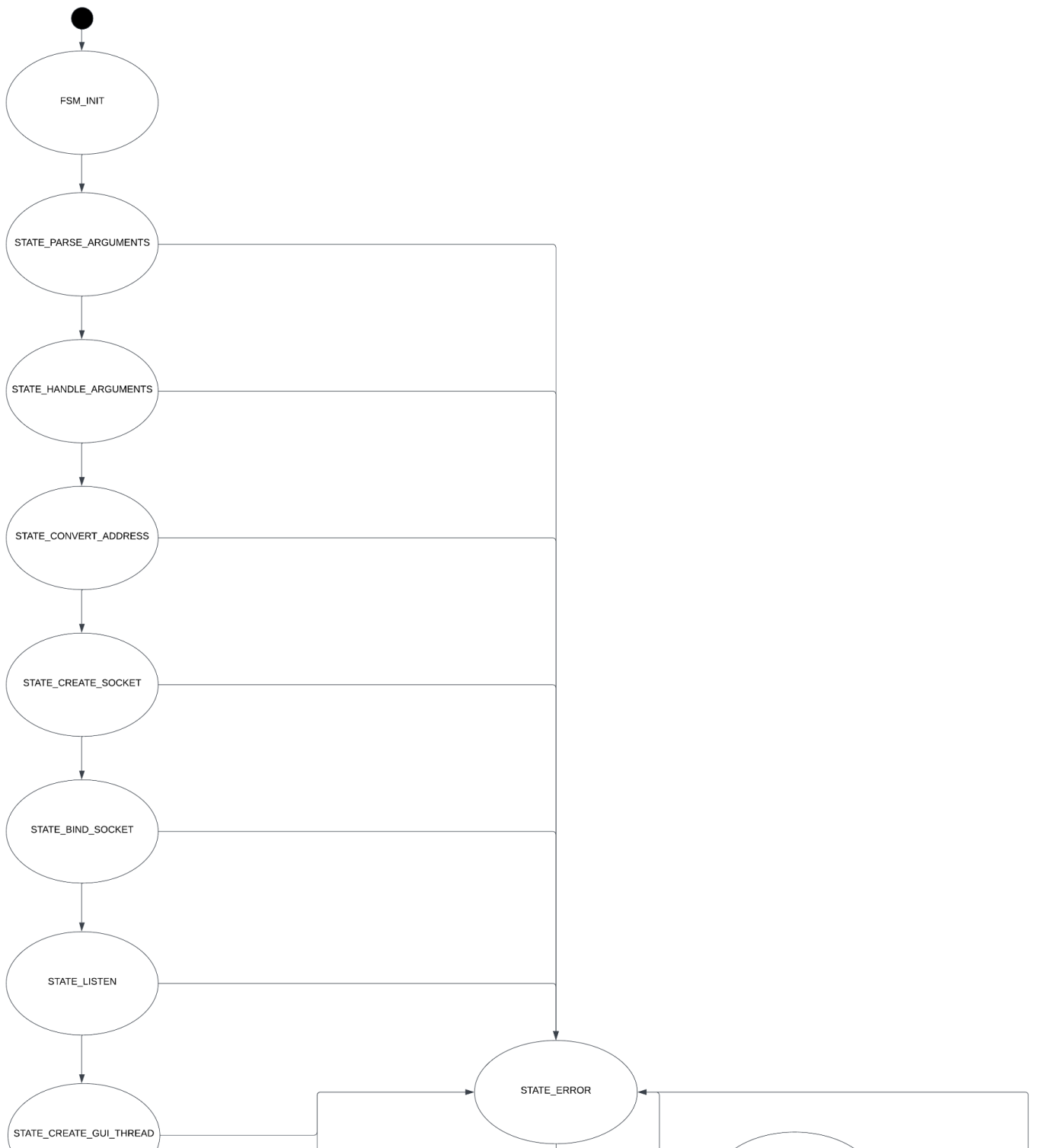
From	To	Handler
FSM_INIT	STATE_READ_FROM_KEYBOARD	read_from_keyboard_handler
STATE_READ_FROM_KEYBOARD	FSM_EXIT	-
STATE_READ_FROM_KEYBOARD	STATE_ERROR	error_handler
STATE_ERROR	FSM_EXIT	-



# 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

<b>From</b>	<b>To</b>	<b>Handler</b>
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	-





# 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\_rcv\_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
ENDIF
IF window_size is less than 3 THEN
    RETURN -1
ENDIF
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  $\text{recv\_seq\_number} + \text{data\_size}$



# **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 sa\_family is AF\_INET

    RETURN sizeof(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\_lossiness\_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 equal -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

    RETURN sizeof(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