Reliable Protocol Design

Ashkan Zahedanaraki
Gagan Bahugun
December 12th, 2023

Structures

State of Proxy

Field	Purpose
proxy_init	initializes the proxy with network settings and starts optional GUI
dynamicProb	separate thread asking the user if they want to change the probability of drop and delays for ACKs or data packets
setup_gui_connection	Separate thread Attempt to connect to GUI
send_statistics_to_gui	Sending data to gui
handler	Listening for packets and determines if packet is data or ACK for appropriate next step.
send_sender	For sending ACKs to the sender that the receiver sent to proxy
send_receiver	For sending data packets to the receiver that the sender sent to proxy
error	For printing error message with state information of where error was captured
destroy	Graceful shutdown: print statistics to “statisticsProxy.txt” and close socket if it exists

State of Sender

Field	Purpose
sender_init	initializes the sender with network settings and start optional GUI
setup_gui_connection	Separate thread Attempt to connect to GUI
send_statistics_to_gui	Sending data to gui
handler	sends data in chunks from stdin, manages message sequencing, and handles packet retransmissions if acknowledgments are not received.
send_message	Sends a packet of data
wait_for_ACK	Waiting for ACKs of previously sent data
error	For printing error message with state information of where the error was captured

destroy	Graceful shutdown: print statistics to “statisticsSender.txt,” attempt to let the receiver know about the shutdown and close socket if it exists

State of receiver

Field	Purpose
receiver_init	initializes the receiver with network settings and start optional GUI
setup_gui_connection	Separate thread Attempt to connect to GUI
send_statistics_to_gui	Sending data to gui
handler	sends back acknowledgments with sequence numbers, and keeps track of the number of ACKs sent
wait_for_data	Waiting for data and resets expected sequence numbers if end message is received
error	For printing error message with state information of where error was captured
destroy	Graceful shutdown: print statistics to “statisticsReceiver.txt,” attempt to let the receiver know about shutdown and close socket if it exists

State of GUI

Field	Purpose
GUI_init	For making the windows of the graphs and binding to a socket
connect_client	Waiting for client to make a connection
handle_client	Handle the client data we receive
graph_data	Graph the data on the window
Error	To deal with any errors the states might capture
Destroy	To shut down gracefully.

Finite State Machine

State Table Of Proxy

From State	To State	Action
proxy_init	Setup_gui_ connection	User wants to gui
Setup_gui_ connection	send_statistics_to_ gui	Connected to gui
proxy_init	dynamicProb	init complete
proxy_init	handler	init complete
handler	send_receiver	Packet received is data
handler	send_sender	Packet received is ACK
proxy_init	error	Fail to initilize
handler	error	error
send_receiver	error	error
send_sender	error	error
error	DESTROY_STATE	Error message displayed

State Table Of Sender

From State	To State	Action
sender_init	Setup_gui_ connection	User wants to gui
Setup_gui_ connection	send_statistics_to_ gui	Connected to gui
sender_init	handler	Initialization complete
handler	send_message	Received arguments for message or resend packet
handler	wait_for_ACK	Message already sent and we want ACK for it
send_message	handler	Message sent
wait_for_ACK	handler	Timeout or ACK received
sender_init	error	error
handler	error	error
send_message	error	error
wait_for_ACK	error	error
error	DESTROY_STATE	Error message displayed

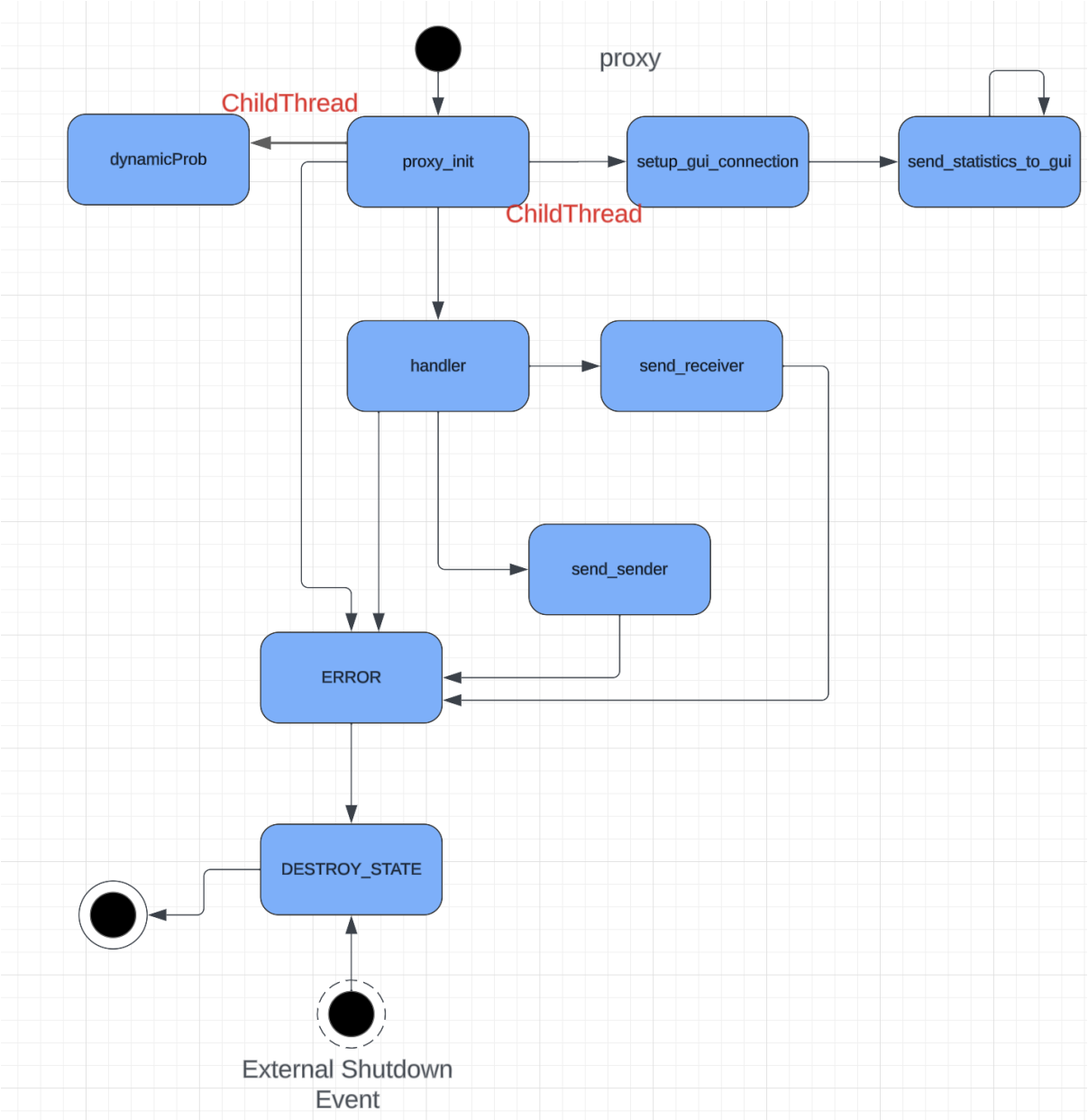
State Table Of Receiver

From State	To State	Action
receiver_init	Setup_gui_ connection	User wants to gui
Setup_gui_ connection	send_statistics_to_ gui	Connected to gui
receiver_init	handler	Initialization complete
handler	wait_for_data	Wait for data
wait_for_data	handler	Received message and send ACK
receiver_init	error	error
handler	error	error
wait_for_data	error	error
error	DESTROY_STATE	Error Message

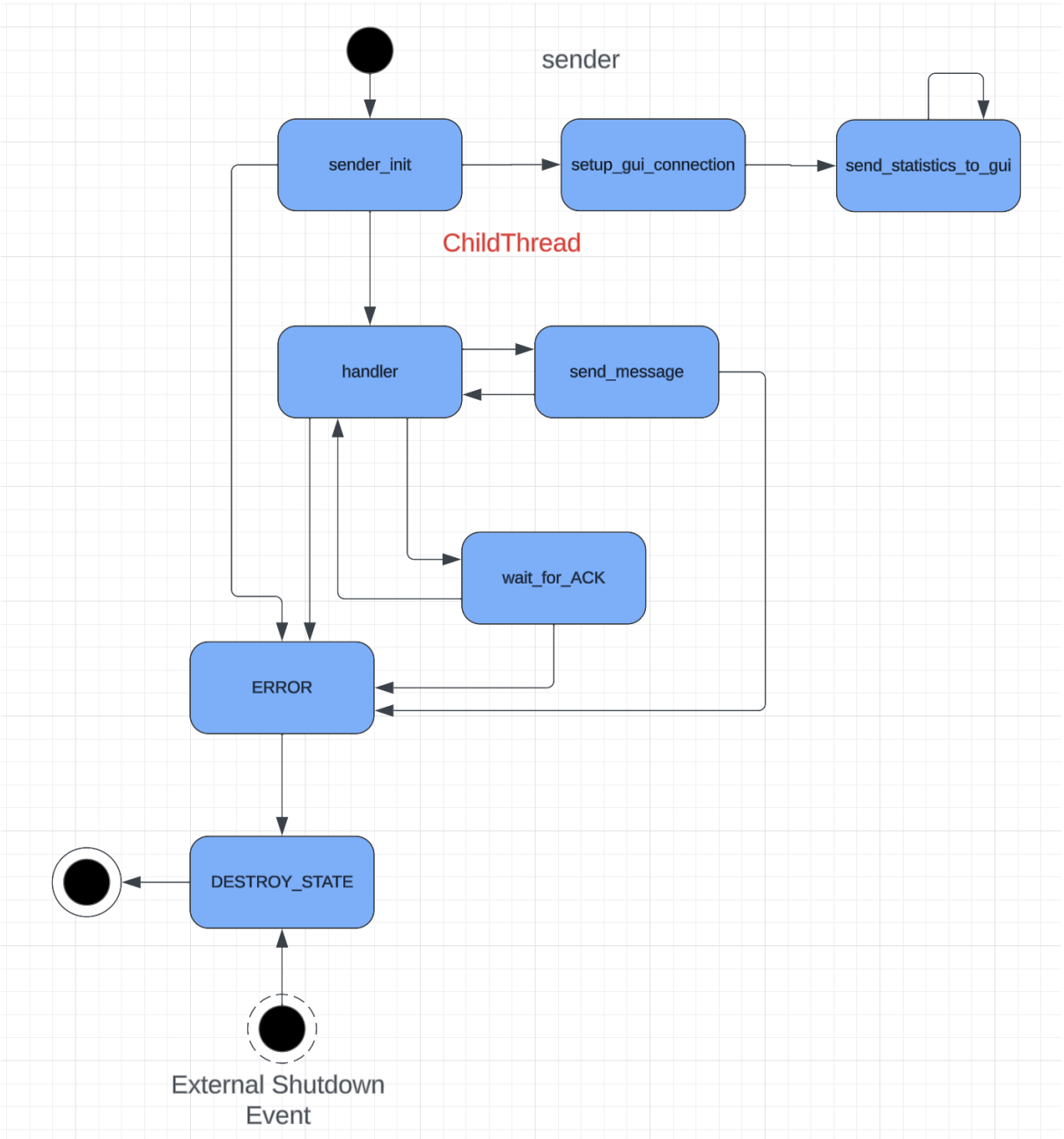
State Table Of GUI

From State	To State	Action
GUI_init	connect_client	Socket bind success
connect_client	handle_client	Connection established
handle_client	Graph data	Data received
Graph data	handle_client	Graphed data
GUI_init	error	error
connect_client	error	error
handle_client	error	error
Graph data	error	error

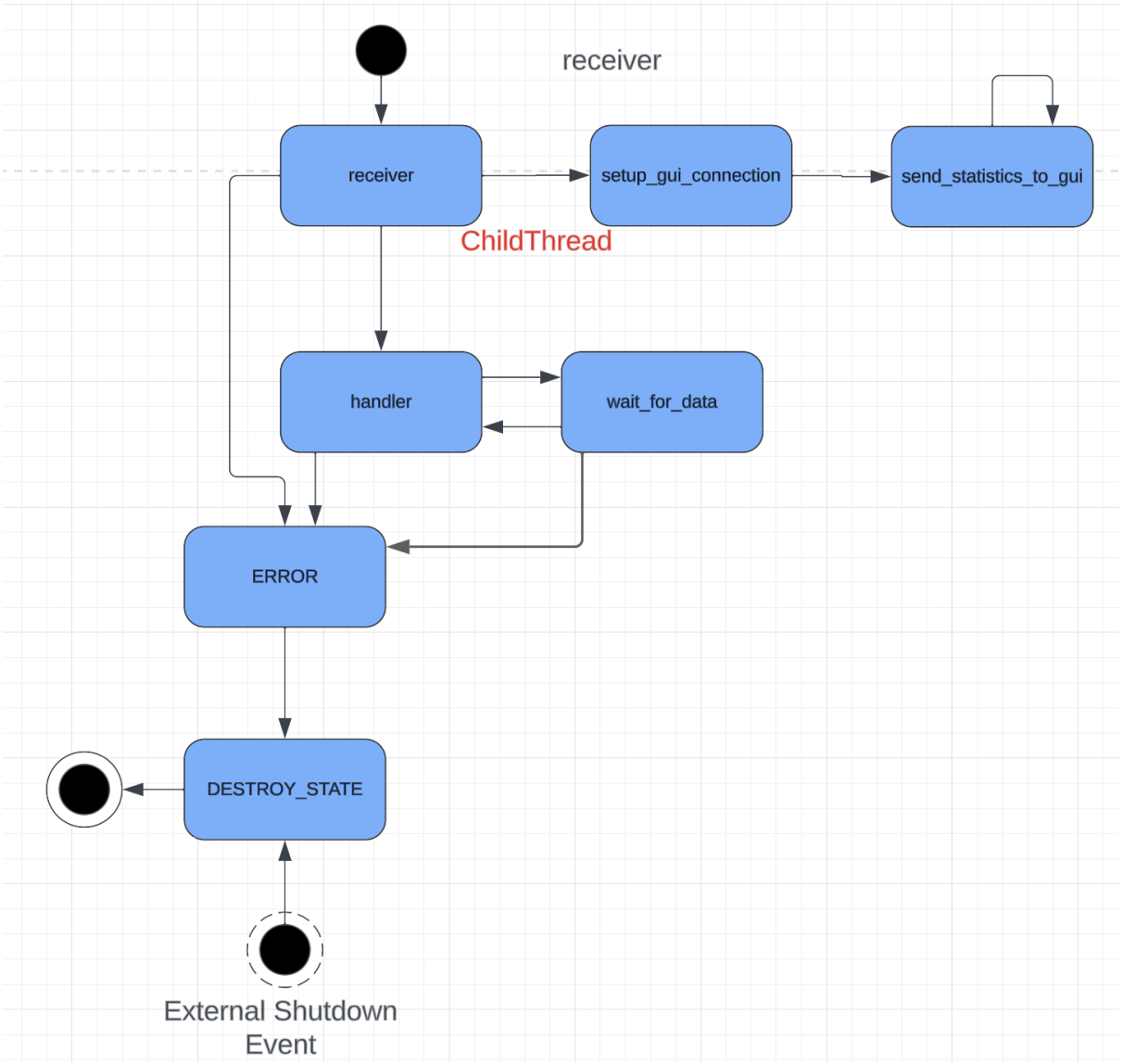
State Transition Diagram of proxy



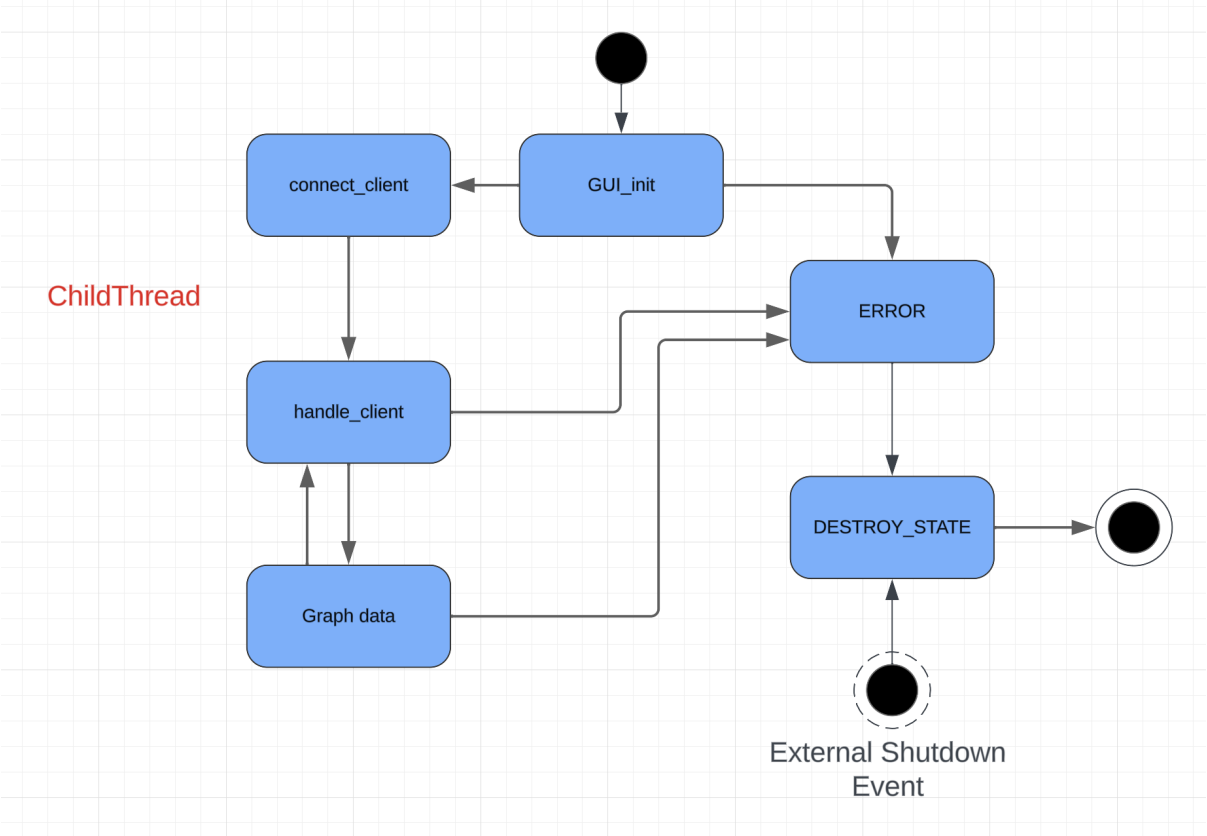
State Transition Diagram of sender



State Transition Diagram of Receiver



State Transition Diagram of GUI



Functions

Proxy

proxy_init

Purpose

initializes the proxy with network settings and starts optional GUI

Return

Type	Next State
Success	`handler, Setup_gui_connection(child thread), dynamicProb(child thread)`
Failure	`error`

Pseudocode

```
Function proxy_init:
    If the number of command line arguments is not 4:
        Raise an error with the message "Usage: python          proxy.py
[Receiver IP] [Receiver Port] [Listen Port]"


Assign receiver_ip, receiver_port, and listen_port from the first three
command line arguments, converting as necessary.

    Create a new UDP socket sock_listen
    Bind sock_listen to the specified listen_port

    Ask the user if they want to connect to the GUI ("yes/no")
    If the answer is "yes":
        If setup_gui_connection() is successful:
            Start a new thread for send_statistics_to_gui

    Ask the user to enter percentages for drop_data_prob, drop_ack_prob,
delay_data_prob, delay_ack_prob
    Convert these percentages from strings to floats and normalize them to
0-1 range

    Start a new daemon thread for dynamicProb

    Call the handler function

    If any exception occurs:
        Call the error function with the exception details
```

dynamicProb

Purpose

separate thread asking the user if they want to change the probability of drop and delays for ACKs or data packets

Return

Type	Next State
Success	END of Thread
Failure	END of Thread

Pseudocode

```
Function dynamicProb:
    Access the global variables: drop_data_prob, drop_ack_prob,
delay_data_prob, delay_ack_prob

    Enter an infinite loop:
        Print a menu to the user with options to change various
probabilities:
            1. Drop Data Probability
            2. Drop ACK Probability
            3. Delay Data Probability
            4. Delay ACK Probability

        Ask the user to enter their choice (1-4)

        If the choice is "1":
            Ask the user to enter a new Drop Data Probability (0-100)
            Update drop_data_prob with the new value

        Else if the choice is "2":
            Ask the user to enter a new Drop ACK Probability (0-100)
            Update drop_ack_prob with the new value

        Else if the choice is "3":
            Ask the user to enter a new Delay Data Probability (0-100)
            Update delay_data_prob with the new value

        Else if the choice is "4":
            Ask the user to enter a new Delay ACK Probability (0-100)
            Update delay_ack_prob with the new value

        Else:
            Print an error message indicating an invalid choice
```

setup_gui_connection

Purpose

Attempt to connect to GUI with TCP socket

Return

Type	Next State
Success	`send_statistics_to_gui`
Failure	`END of Thread`

Pseudocode

```
Function setup_gui_connection:

    Prompt the user to enter the GUI IP address and store it
in gui_ip
    Prompt the user to enter the GUI port, convert it to an
integer, and store it in gui_port

    Create a new TCP socket gui_socket
    Attempt to connect gui_socket to the specified gui_ip and
gui_port

    If the connection is successful:
        Return 0 to indicate success

    If an exception occurs:
        Print the exception message and a "GUI not connected"
error
        Return 1 to indicate failure
```

send_statistics_to_gui

Purpose

Send statistic to GUI via TCP socket

Return

Type	Next State
Success	Endless loop(until destroy is called)
Failure	END of Thread

Pseudocode

```
Function send_statistics_to_gui:
    Access the global variables: sent_data_packets,
received_ACK_packets, gui_socket,
                                 dropped_ACK_packets,
dropped_data_packets, delayed_ACK_packets, delayed_data_packets

    Enter an infinite loop:
        If gui_socket is connected:
            Prepare a dictionary named stats containing:
                "client_id" set to "proxy"
                "dropped_ACK_packets" set to the global
dropped_ACK_packets count
                "dropped_data_packets" set to the global
dropped_data_packets count
                "delayed_ACK_packets" set to the global
delayed_ACK_packets count
                "delayed_data_packets" set to the global
delayed_data_packets count
                "total_ACK_packets" calculated as
dropped_ACK_packets plus sent_ACK_packets
                "total_data_packets" calculated as
dropped_data_packets plus sent_data_packets

            Convert the stats dictionary to a JSON string and
encode it
            Send the encoded data through gui_socket

            Wait for 1 second to avoid overwhelming the network

        If an exception occurs:
            Print the error message
            Close and nullify gui_socket
```

handler

Purpose

Listening for packets and determines if packet is data or ACK for appropriate next step.

Parameters

ACK being sent to the sender

Return

Type	Next State
Success	`send_sender, send_receiver`
Failure	`error`

Pseudocode

```
Function handler:
drop_ack_prob, delay_data_prob, delay_ack_prob, max_delay
    Initialize a counter (count) to 0

    While True (infinite loop):
        Wait to receive data from the socket (sock_listen) along with the
address (Addr)

        If count is 0 (first time receiving data):
            Set senderAddr to Addr (address of the sender)
            Increment count by 1

        If the received data matches the pattern '(?:\d+|end):ACK'
(indicating it's an ACK packet):
            Start a new thread to handle sending ACK back to the sender
(send_sender function)
            Pass the received data as an argument to the thread

        Else (indicating it's a data packet):
            Update senderAddr to Addr (address of the sender)
            Start a new thread to handle sending data to the receiver
(send_receiver function)
            Pass the received data and the receiver's address as arguments to
the thread
```

send_sender

Purpose

For sending ACKs to the sender that the receiver sent to proxy

Parameters

ACK being sent to the sender

Return

Type	Next State
Success	`End of Thread`
Failure	`error`

Pseudocode

```
Function send_sender with parameter ACK:
    Access the global variables: dropped_ACK_packets, delayed_ACK_packets,
sent_ACK_packets, drop_ack_prob, delay_ack_prob, max_delay, sock, senderAddr

       If the probability of dropping an ACK packet (drop_ack_prob) is met:
            Print a message indicating the ACK packet is being dropped
            Increment the dropped_ACK_packets counter
            Return from the function (do not proceed to send the ACK)

      If the probability of delaying an ACK packet (delay_ack_prob) is met:
            Print a message indicating the ACK packet is being delayed
            Increment the delayed_ACK_packets counter
            Pause execution for a random time up to max_delay seconds

        Create a new UDP socket (sock)

        Send the ACK packet to the sender's address (senderAddr) using the
socket (sock)

        Increment the sent_ACK_packets counter

        If an exception occurs:
            Call the error function with the exception details
```

send_receiver

Purpose

For sending data packets to the receiver that the sender sent to proxy

Parameters

data and the address of the receiver

Return

Type	Next State
Success	`End of Thread`
Failure	`error`

Pseudocode

```
Function send_receiver with parameters data, receiverAddr:
    Access the global variables: dropped_data_packets,
delayed_data_packets, sent_data_packets, drop_data_prob,
delay_data_prob, max_delay, sock_listen

    Try:
        If the probability of dropping a data packet
(drop_data_prob) is met:
            Print a message indicating the data packet is being
dropped
            Increment the dropped_data_packets counter
            Return 1 to indicate a drop occurred

        If the probability of delaying a data packet
(delay_data_prob) is met:
            Print a message indicating the data packet is being
delayed
            Increment the delayed_data_packets counter
            Pause execution for a random time up to max_delay
seconds

        Send the data packet to the receiver's address
(receiverAddr) using the socket (sock_listen)

        Increment the sent_data_packets counter

        If an exception occurs:
            Call the error function with the exception details
```

error

Purpose

For printing error message with state information of where error was captured

Parameters

Message of exception captured

Return

Type	Next State
Success	`Destory`
Failure	`Destory`

Pseudocode

```
Function error with parameters message, stateName:
Print the error message and the name of the state where the error occurred

    Call the destroy function to perform cleanup and exit
```

Destory

Purpose

Graceful shutdown: print statistics to “statisticsProxy.txt,” attempt to let the receiver know about shutdown and close socket if it exists

Return

Type	Next State
Success	EXIT
Failure	EXIT

Pseudocode

```
Function destroy:
    Print a message indicating the proxy is closing

    If the global socket variable (sock_listen) exists:
        Print a message indicating the socket is closing
        Close the socket (sock_listen)

    Get the current date and time in "YYYY-MM-DD HH:MM:SS" format

    Open a file named "statisticsProxy.txt" in append mode
    Write a header with the current date and time to the file
    Write the statistics to the file, including:
        - Dropped ACK packets
        - Dropped data packets
        - Delayed ACK packets
        - Delayed data packets
        - Total ACK packets (sum of sent and dropped ACK packets)
        - Total data packets (sum of sent and dropped data
packets)
        - Sent ACK packets
        - Sent data packets
    Add a separator line for readability

    Print a message indicating statistics are saved to the file

    Exit the program with status 0
```

Sender

sender_init

Purpose

initializes the sender with network settings and start optional GUI

Return

Type	Next State
Success	`handler, Setup_gui_connection(child thread)`
Failure	`error`

Pseudocode

```
Function sender_init:
    Initialize global variables: proxy_ip, proxy_port, sock

    Parse command line arguments using arg_handler function
    Assign proxy_ip from the first argument
    Assign proxy_port from the second argument converted to
integer
    Create a new UDP socket sock

    Ask the user if they want to connect to the GUI ("yes/no")
    If the answer is "yes":
        If setup_gui_connection() is successful:
            Start a new daemon thread for
send_statistics_to_gui

    Call the handler function with the parsed arguments

    If an exception occurs:
        Call the error function with the exception details
```

setup_gui_connection

Purpose

Attempt to connect to GUI with TCP socket

Return

Type	Next State
Success	`send_statistics_to_gui`
Failure	`END of Thread`

Pseudocode

```
Function setup_gui_connection:

    Prompt the user to enter the GUI IP address and store it
in gui_ip
    Prompt the user to enter the GUI port, convert it to an
integer, and store it in gui_port

    Create a new TCP socket gui_socket
    Attempt to connect gui_socket to the specified gui_ip and
gui_port

    If the connection is successful:
        Return 0 to indicate success

    If an exception occurs:
        Print the exception message and a "GUI not connected"
error
        Return 1 to indicate failure
```

send_statistics_to_gui

Purpose

Send statistic to GUI via TCP socket

Return

Type	Next State
Success	Endless loop(until destroy is called)
Failure	END of Thread

Pseudocode

```
Function send_statistics_to_gui:
    Access the global variables: sent_data_packets,
received_ACK_packets, gui_socket

    Enter an infinite loop:
        If gui_socket is connected:
            Prepare a dictionary named stats containing:
                "client_id" set to "sender"
                "sent_data_packets" set to the global
sent_data_packets count
                "received_ACK_packets" set to the global
received_ACK_packets count

            Convert the stats dictionary to a JSON string and
encode it
            Send the encoded data through gui_socket

            Wait for 1 second to avoid overwhelming the
network

        If an exception occurs:
            Print the error message
            Close and nullify gui_socket
```

handler

Purpose

sends data in chunks from stdin, manages message sequencing, and handles packet retransmissions if acknowledgments are not received.

Return

Type	Next State
Success	`send_message, wait_for_ACK`
Failure	`error`

Pseudocode

```
Function handler with parameter args:
    Initialize sequence number to 0
    Initialize an empty message buffer
    Set max_chunk_size to 3000 bytes

    Loop through each line of standard input:
        If the first character of the line is "<":
            Remove the first character and strip whitespace
            Open the file specified in the line and read its content into the line
variable

        If the line is not empty:
            Set message_buffer to the content of the line

            While the length of message_buffer is greater or equal to
max_chunk_size:
                Split message_buffer into a chunk of max_chunk_size and the
remaining part
                Increment the sequence number by 1
                Send the chunk with its sequence number
                Initialize a counter for resend attempts

                While not receiving an ACK for the chunk:
                    If the resend attempts exceed 20:
                        Call the error function indicating excessive resend
attempts
                    Print a message indicating resending of the chunk
                    Increment the resend counter
                    Resend the chunk with the sequence number

            If there is any remaining content in the message_buffer:
                Increment the sequence number by 1
                Send the remaining content with its sequence number
                While not receiving an ACK for the remaining content:
                    Print a message indicating resending of the content
                    Resend the remaining content with the sequence number

    Print a message indicating the end of input
    Call the destroy function
```

send_message

Purpose

Sends a packet of data

Parameters

args being sent

Return

Type	Next State
Success	`handler`
Failure	`error`

Pseudocode

```
Function handler with parameter args:
    Loop through each line of standard input:
        If the first character of the line is "<":
            Remove the first character and strip whitespace
            Open the file specified in the line and read its content into the
line variable
        If the line is not empty:
            Set message_buffer to the content of the line
            While the length of message_buffer is greater or equal to
max_chunk_size:
                Split message_buffer into a chunk of max_chunk_size and the
remaining part
                Increment the sequence number by 1
                Send the chunk with its sequence number
                Initialize a counter for resend attempts

                While not receiving an ACK for the chunk:
                    If the resend attempts exceed 20:
                        Call the error function indicating excessive resend
attempts
                    Print a message indicating resending of the chunk
                    Increment the resend counter
                    Resend the chunk with the sequence number

            If there is any remaining content in the message_buffer:
                Increment the sequence number by 1
                Send the remaining content with its sequence number
                While not receiving an ACK for the remaining content:
                    Print a message indicating resending of the content
                    Resend the remaining content with the sequence number

    Print a message indicating the end of input
    Call the destroy function
```

wait_for_ACK

Purpose

sends data in chunks from stdin, manages message sequencing, and handles packet retransmissions if acknowledgments are not received.

Parameters

Expected sequence number

Return

Type	Next State
Success	`handler`
Failure	`error`

Pseudocode

```
Function wait_for_ACK with parameter sequence:
        Set a timeout of 1.0 seconds on the socket (sock)

        Wait to receive data from the socket
        Decode the received data and strip any whitespace

        If the received data matches the expected format "sequence:ACK":
            Increment the received_ACK_packets counter
            Print a message confirming receipt of ACK
            Return 0 indicating successful receipt of ACK

        Else:
            Print a message indicating a mismatch in received data and
expected ACK
            Return 1 indicating ACK was not received or did not match

        If a timeout occurs while waiting for data:
            Print a message indicating no ACK was received
            Return 1 indicating failure to receive ACK

        If any other exception occurs:
            Call the error function with the exception details

    Return 1 as default to indicate failure in receiving the expected ACK
```

error

Purpose

For printing error message with state information of where error was captured

Parameters

Message of exception captured

Return

Type	Next State
Success	`Destory`
Failure	`Destory`

Pseudocode

```
Function error with parameters message, stateName:
Print the error message and the name of the state where the error occurred

    Call the destroy function to perform cleanup and exit
```

Destory

Purpose

Graceful shutdown: print statistics to “statisticsSender.txt,” attempt to let the receiver know about the shutdown and close socket if it exists

Return

Type	Next State
Success	`send_message, wait_for_ACK`
Failure	`EXIT`

Pseudocode

```
Function destroy:
    Print a message indicating the sender is closing

    If the global socket variable (sock) exists:
        Print a message indicating the socket is closing

        Send a special message "0:end" to signal end of transmission
        Initialize a counter (count) to 1

        While not receiving an acknowledgment for the "end" message:
            If the counter reaches 5 attempts:
                Print a message to manually close the receiver
                Break the loop

            Resend the "end" message
            Increment the counter

        Close the socket (sock)

    Print a header for statistics

    Access the global variables: sent_data_packets, received_ACK_packets
    Print the number of sent data packets and received ACK packets

    Get the current date and time in "YYYY-MM-DD HH:MM:SS" format

    Open a file named "statisticsSender.txt" in append mode
    Write a header with the current date and time to the file
    Write the number of sent data packets and received ACK packets to the
file
    Append a separator line for readability

    Print a message indicating statistics are saved to the file

    Exit the program with status 0
```

Receiver

receiver_init

Purpose

initializes the receiver with network settings and start optional GUI

Return

Type	Next State
Success	`handler, Setup_gui_connection(child thread)`
Failure	`error`

Pseudocode

```
Function sender_init:
    Parse command line arguments using arg_handler function
    Assign proxy_ip from the first argument
    Assign proxy_port from the second argument converted to
integer
    Create a new UDP socket sock

    Ask the user if they want to connect to the GUI ("yes/no")
    If the answer is "yes":
        If setup_gui_connection() is successful:
            Start a new daemon thread for
send_statistics_to_gui

    Call the handler function with the parsed arguments

    If an exception occurs:
        Call the error function with the exception details
```

handler

Purpose

sends back acknowledgments with sequence numbers, and keeps track of the number of ACKs sent

Return

Type	Next State
Success	`wait_for_data`
Failure	`error`

Pseudocode

```
Function handler:
    Enter an infinite loop:
        Call wait_for_data function to receive data and
extract address and sequence number

        Prepare an acknowledgment message with the format
"sequence:ACK"
        Increment the sent_ACK_packets counter

        Send the acknowledgment message to the address
received from wait_for_data

        If an exception occurs:
            Call the error function with the exception details
```

wait_for_data

Purpose

Waiting for data and resets expected sequence numbers if end message is received

Return

Type	Next State
Success	`handler`
Failure	`error`

Pseudocode

Function wait_for_data:

    Wait to receive data from the socket (sock), along with the sender's address (addr)
    If no data is received:
        Raise a ValueError indicating "No data received."

    Decode the received data bytes into a string (data_str)

    Split the data string into sequence number and message parts
    Convert the sequence number part into an integer

    If the sequence number is less than or equal to received_sequences and is not 0:
        Return the sender's address and the sequence number

    Update the received_sequences with the new sequence number
    Increment the received_data_packets counter

    If the data string is "0:end":
        Reset received_sequences to 0 (indicating end of transmission)
    Else:
        Print the message part

    Return the sender's address and the sequence number

  If an exception occurs:
    Call the error function with the exception details

wait_for_data

Purpose

Waiting for data and resets expected sequence numbers if end message is received

Return

Type	Next State
Success	handler
Failure	error

Pseudocode

```
Function wait_for_data:
    Access the global variables: sock, received_data_packets,
received_sequences
        Receive data from the socket sock, storing the data and address
        If no data is received, raise a ValueError indicating no data was
received

        Decode the received bytes into a string (data_str)

        Split data_str into sequence number and message based on the first
colon (':')
        Convert the sequence number to an integer

        If the sequence number is less than or equal to received_sequences
and not zero:
            Return the address and sequence number without printing the
message

        Update received_sequences to the current sequence number
        Increment received_data_packets count

        If the message is "0:end", reset received_sequences to zero
        Else, print the message

        Return the address and sequence number

        If an exception occurs:
            Call the error function with the exception details
```

error

Purpose

For printing error message with state information of where error was captured

Parameters

Message of exception captured

Return

Type	Next State
Success	`Destory`
Failure	`Destory`

Pseudocode

```
Function error with parameters message, stateName:
Print the error message and the name of the state where the error
occurred

    Call the destroy function to perform cleanup and exit
```

Destory

Purpose

Graceful shutdown: print statistics to “statisticsReceiver.txt,” attempt to let the receiver know about shutdown and close socket if it exists

Return

Type	Next State
Success	EXIT
Failure	EXIT

Pseudocode

```
Function destroy:
    Print a message indicating the receiver is closing

    If the global socket variable (sock) exists:
        Print a message indicating the socket is closing
        Close the socket (sock)

    Print a header for statistics

    Access the global variables: sent_ACK_packets,
received_data_packets
    Print the number of sent ACK packets and received data packets

    Get the current date and time in "YYYY-MM-DD HH:MM:SS" format

    Open a file named "statisticsReceiver.txt" in append mode
    Write a header with the current date and time to the file
    Write the number of sent ACK packets and received data packets
to the file
    Append a separator line for readability

    Print a message indicating statistics are saved to the file

    Exit the program with status 0
```

setup_gui_connection

Purpose

Attempt to connect to GUI with TCP socket

Return

Type	Next State
Success	`send_statistics_to_gui`
Failure	`END of Thread`

Pseudocode

```
Function setup_gui_connection:

    Prompt the user to enter the GUI IP address and store it
in gui_ip
    Prompt the user to enter the GUI port, convert it to an
integer, and store it in gui_port

    Create a new TCP socket gui_socket
    Attempt to connect gui_socket to the specified gui_ip and
gui_port

    If the connection is successful:
        Return 0 to indicate success

    If an exception occurs:
        Print the exception message and a "GUI not connected"
error
        Return 1 to indicate failure
```

send_statistics_to_gui

Purpose

Send statistic to GUI via TCP socket

Return

Type	Next State
Success	Endless loop(until destroy is called)
Failure	END of Thread

Pseudocode

```
Function send_statistics_to_gui:
    Access the global variable: gui_socket

    Enter an infinite loop:
        If gui_socket is connected:
            Prepare a dictionary named stats containing:
                "client_id" set to "receiver"
                "sent_ACK_packets" set to the global
sent_ACK_packets count
           "received_data_packets" set to the global
received_data_packets count

   Convert the stats dictionary to a JSON string and encode it
            Send the encoded data through gui_socket

           Wait for 1 second to avoid overwhelming the network

        If an exception occurs:
            Print the error message
            Close and nullify gui_socket
```

GUI

start_server

Purpose

Starts the GUI server and starts listening for client connections to receive data

Return

Type	Next State
Success	`handle_client`
Failure	`error`

Pseudocode

```
Function start_server:
     Create the socket
     Bind the Socket
     Start listening for connections
     Print start message

     Enter an infinite loop:
          Accept connection
          Print connected message
          Start thread to handle client

     If there was an error print an error message
```

handle_client

Purpose

Receives data from client

Return

Type	Next State
Success	`graph_data`
Failure	`error`

Pseudocode

```
Function handle_client:
     Enter an infinite loop:
          Receive data from client
          If there isn’t any data break the loop
          Print received data message
          Store the data received in different queues
          depending on which client the data was received from
```

create_client_window

Purpose

Creates the windows where the graphs are displayed

Return

Type	Next State
Success	`graph_data`
Failure	`error`

Pseudocode

```
Function create_client_window:
     Create the window
     Set the window title

     Initialize lists for plotting data
     Update the graph after 1 second
```

update_graph

Purpose

initializes the receiver with network settings and start optional GUI

Return

Type	Next State
Success	`handle_client`
Failure	`error`

Pseudocode

```
Function update_graph:
     While there is still data in the queue:
          Get the data
          Print updating graph message

          If the data received is from the sender:
               Add the data received to the sender lists
               If there is data:
                    Plot the data on the graph

          If the data received is from the receiver:
               Add the data received to the receiver lists
               If there is data:
                    Plot the data on the graph

          If the data received is from the proxy:
               Add the data received to the proxy lists
               If there is data:
                    Plot the data on the graph

     Draw the Graph
     If there was an error print the error message
     Update the window after 1 second
```