COMP 7005
Project 1

Design

Brandon Rada
A01345707
Dec 2nd, 2025

Purpose

Understand the limitations of UDP and the need for reliability mechanisms. Design a message-based protocol with identifiers and acknowledgments. Implement and evaluate retry logic, timeout handling, and error cases. Simulate packet loss and delay using a configurable proxy server. Measure and describe how the system performs under degraded conditions.
Implement reliable communication over UDP by simulating network unreliability and developing a simple protocol that utilizes retransmissions and acknowledgments.
Three programs: a client, a server, and a proxy server that introduces packet loss and delay.

Data Types

Arguments

Purpose: To hold the unparsed command-line argument information

Field	Type	Description
Client		
target-ip	addr	IP address to send packets to
target-port	int	Port to send packets to
timeout	float	timeout in seconds for ACKs
max-retries	int	maximum number of retries for sending a message
Server		
listen-ip	addr	IP address to bind for incoming packets
listen-port	int	Port to listen on for client packets
Proxy		
listen-ip	addr	IP address to bind for client packets
listen-port	int	Port to listen on for client packets
target-ip	addr	Server IP address to forward packets to
target-port	int	Server port number
client-drop	int	percentage chance of dropping client -> server packets

server-drop	int	percentage chance of dropping server -> client packets
client-delay	int	percentage chance of delaying client -> server packets
server-delay	int	percentage chance of delaying server -> client packets
client-delay-time-min	int	minimum delay time in milliseconds for client -> server packets
client-delay-time-max	int	maximum delay time in milliseconds for client -> server packets
server-delay-time-min	int	minimum delay time in milliseconds for server -> client packets
server-delay-time-max	int	maximum delay time in milliseconds for server -> client packets

Settings

Purpose: To hold the settings the program needs to run.

Field	Type	Description
sock	socket	The socket on which to send packets
seq	int	The sequence number for the current packet
packet	packet	The packet that is being sent around
retries	int	The retry number (increased on message send failure)
max_retries	int	The maximum number of retries for a message.
PACKET_FORMAT	string	"!IIH", the packet format.
HEADER_SIZE	Header size	The size of the Header that is calculated from the Packet Format
FLAG_ACK	int	The ACK flag

Context

Purpose: To hold the arguments, settings, and exit information

Field	Type	Description

args	arguments	The parsed command line arguments
data	data	An incoming communication.
pkt	packet	The decoded packet
payload	string	The encoded user-entered message
ack_packet	packet	ACK to send to client
server_addr	addr	The server's address
client_addr	addr	The dynamically set client address

Functions

Function	Description
Client	
main()	The main running loop for the client to send and receive messages.
Server	
main()	The main running loop for the server to receive and respond to messages
encode_ack	Create an ACK packet
Proxy	
main()	The running loop for the client to forward messages and simulate drops and delays.
Common functions	
encode_packet	Encode a packet into Bytes
decode_packet	Decode Bytes into a packet
make_logger	Create a logger to use to log to a specified file

States

State	Description
PARSE_ARGS	Parse command line arguments

HANDLE_ARGS	Verify and convert the command-line arguments for use
USAGE	Display an error message when the command line arguments have an issue
DISPLAY_MESSAGES	Display the message passed on the command line the specified number of times
CLEANUP	Cleanup before exit
Client	
MESSAGE	New message to send
FAILED_SEND	Failed to send message
RESEND	Send the message again
Server	
Proxy	
DROP	Drop a packet
DELAY	Delay a packet
Common	
PARSE_ARGS	Parse command line arguments
SOCKET	Create a socket
SEND	Sending packet
RECEIVE	Receiving from the socket
CLEANUP	Cleanup before exit

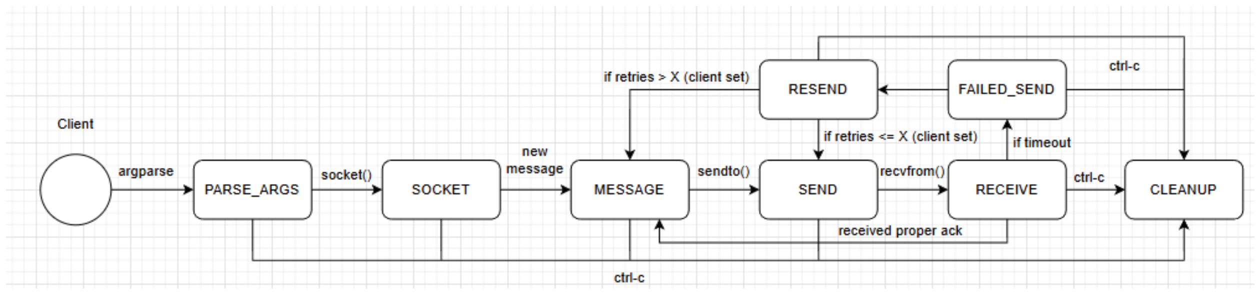
State Table

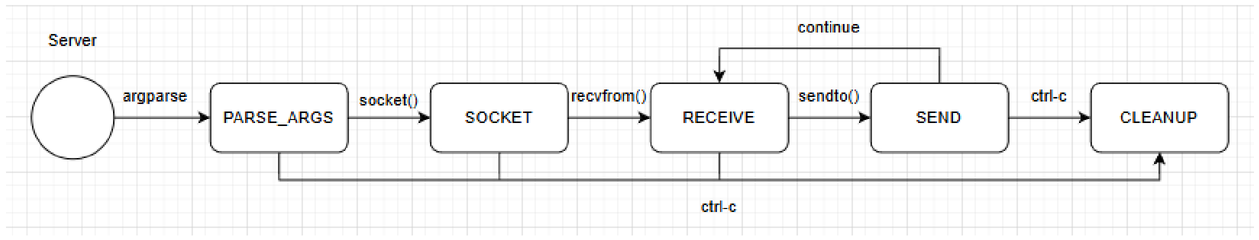
From State	To State	Function
Client		
PARSE_ARGS	SOCKET	socket()
SOCKET	MESSAGE	stdin
MESSAGE	SEND	sendto()
SEND	RECEIVE	recvfrom()
RECEIVE	MESSAGE	Received proper ack, ack == seq
RECEIVE	FAILED_SEND	if timeout
FAILED_SEND	RESEND	Have to resend
RESEND	SEND	If retries <= max_retries
RESEND	MESSAGE	if retries > max_retries
ANY	CLEANUP	ctrl-c
Server		
PARSE_ARGS	SOCKET	socket()
SOCKET	RECEIVE	recvfrom()
RECEIVE	SEND	sendto()
SEND	RECEIVE	continue
ANY	CLEANUP	ctrl-c
Proxy		
PARSE_ARGS	SOCKET	socket()
SOCKET	RECEIVE	recvfrom()
RECEIVE	SEND	sendto()
RECEIVE	DELAY	if delay
RECEIVE	DROP	if drop
SEND	RECEIVE	recvfrom()

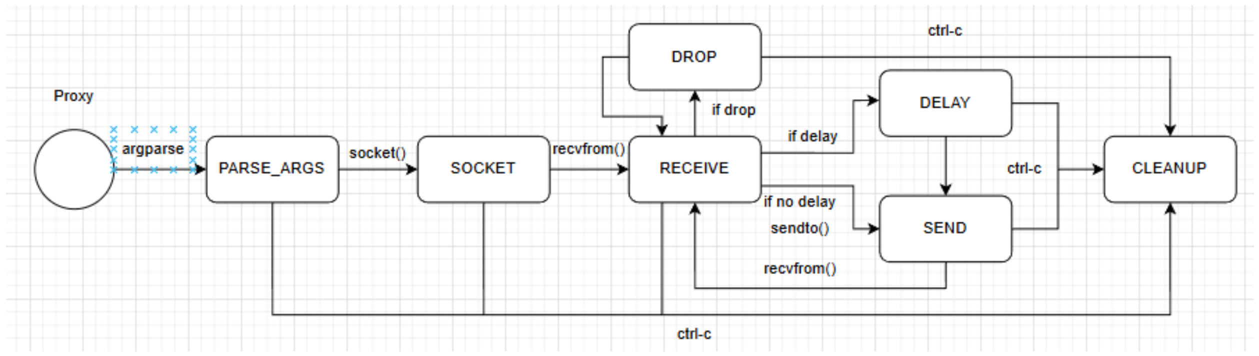
DELAY	SEND	After a random (user-bound) delay
DROP	RECEIVE	Drop and receive
ANY	CLEANUP	ctrl-c

State Transition Diagram

(Note that you can use [this](#) to create the diagram)







Pseudocode

(***Pseudocode is a language/platform-independent way to communicate what functions are supposed to do***).

main (client)

Parameters

Parameter	Type	Description
args	arguments	The program program arguments

Return

Value	Reason

Pseudo Code

```
Parse arguments and set = args

create socket and set = sock
Set sock.timeout to argos.timeout

enter main loop
     read from stdin
     create var payload set = stdin line
     set retries = 0
     try:
     while retries <= argos.max_retries
          packet = encode_packet(payload)

          try:
               sendto() with packet to target address
          except OSError:
               retries += 1
               Continue
          try:
               data and addr = recvfrom(4096)
               pkt = decode_packet(data)
          except socket.timeout:
               retries += 1
               continue

          If pkt[“flags”] & FLAG_ACK and pkt[“ack”] == seq:
               Break
          Elif pkt[“ack”] < seq:
               Continue
          Else:
               Retries += 1

          If retries > argos.max_retries:
               log("Failed to send message after {args.max_retries}
retries: seq={seq}")
          Else:
               log("Message seq={seq} delivered")

          Seq += 1

     Except KeyboardInterrupt:
          print("KeyboardInterrupt, exiting")
     Finally:
          sock.close()
```

main (server)

Parameters

Parameter	Type	Description
args	arguments	The program program arguments

Return

Value	Reason

Pseudo Code

```
Parse arguments and set = args

create socket and set = sock
Set sock.timeout to argos.timeout

Enter main loop
     try:
          data and addr = recvfrom(4096)
     Except KeyboardInterupt:
          Break
     Except Exception:
          log("RECV error")
          Continue

      Try:
          Pkt = decode_packet(data)
      Except:
          log("failed to decode")
          Continue

      Seq = pkt["seq"]
      Payload = pkt["payload"]

      Try:
          print(payload.decode("utf-8"))
      Except:
          print(payload)

      Try:
          Ack_packet = encode_ack(seq)
          sock.sendto() with ack_packet to addr
      Except:
          log("Error sending ACK")

sock.close()
```

encode_ack (server)

Parameters

Parameter	Type	Description
seq	int	The sequence number

Return

Value	Reason

Pseudo Code

```
Set FLAG_ACK = "!IIH"
Return encode_packet(seq=0, ack=seq, flags=FLAG_ACK, payload=b"")
```

main (proxy)

Parameters

Parameter	Type	Description
args	arguments	The program program arguments

Return

Value	Reason

Pseudo Code

```
Parse arguments and set = args

create socket and set = sock
Set sock.timeout to argos.timeout

Set server_addr = (args.target_ip, argos.target_port)
Set client_addr = None

try:
Enter main loop
     Data, addr = sock.recvfrom(4096)

     If client_addr is None:
          Set Client_addr = addr

     If addr == client_addr:
          If random.randint(1,100) inclusive is <=
argos.client_drop:
               Continue # drop packet
          If argos.client_delay:
               Set delay =
random.uniform(args.client_delay_time_min,
argos.client_delay_time_max)
               time.sleep(delay)

          sock.sendto() with data to server_addr

     Else:
          If random.randint(1,100) <= argos.server_drop:
                Continue # drop packet
           If args.server_delay:
                Set delay =
random.uniform(args.server_delay_time_min,
argos.server_delay_time_max)
                time.sleep(delay)

           If client_addr:
                sock.sendto() with data to client_addr
Except KeyboardInterrupt:
     print(“proxy shutting down”)
Finally:
     sock.close()
```

encode_packet

Parameters

Parameter	Type	Description
seq	int	The sequence number
ack	int	The ack number
flags	int	The flags of the packet
payload	bytes	The payload

Return

Value	Reason
Header + payload	This is the encoded packet that is ready to be sent

Pseudo Code

```
Set PACKET_FORMAT = “!IIH”
Set header = struct.pack(PACKET_FORMAT, seq, ack, flags)
Return header + payload
```

decode_packet

Parameters

Parameter	Type	Description
data	bytes	The data to be decoded

Return

Value	Reason
seq	The seq number in the decoded packet
ack	The ack value in the decoded packet
flags	The flags found in the decoded packet
payload	The payload of the decoded packet

Pseudo Code

```
If length of data < HEADER_SIZE:
     Raise ValueError(“Packet too short”)

Header = data[:HEADER_SIZE]
Payload = data[HEADER_SIZE:]

Set seq, ack and flags = struct.unpack() the header according to the
HEADER_FORMAT

return {
     Seq,
     Ack,
     Flags,
     payload
}
```

make_logger

Parameters

Parameter	Type	Description
filename	string	The name of the file that will be created

Return

Value	Reason
log	A log of what happened

Pseudo Code

```
os.makedirs(“logs”, exist_ok=True)
Set f = open(f”logs/{filename}”, “a”)

Def log(msg):
     Set timestamp = datetime.datetime.now().strftime("%Y-%m-%d
%H:%M:%S")
     Set formatted = f"[{timestamp}] {msg}"
     f.write(formatted + “\n”)
     f.flush()
Return log
```