Computer Network

Go Back N Protocol

Ritika Thakur(2022408) Swarnima Prasad (2022525)

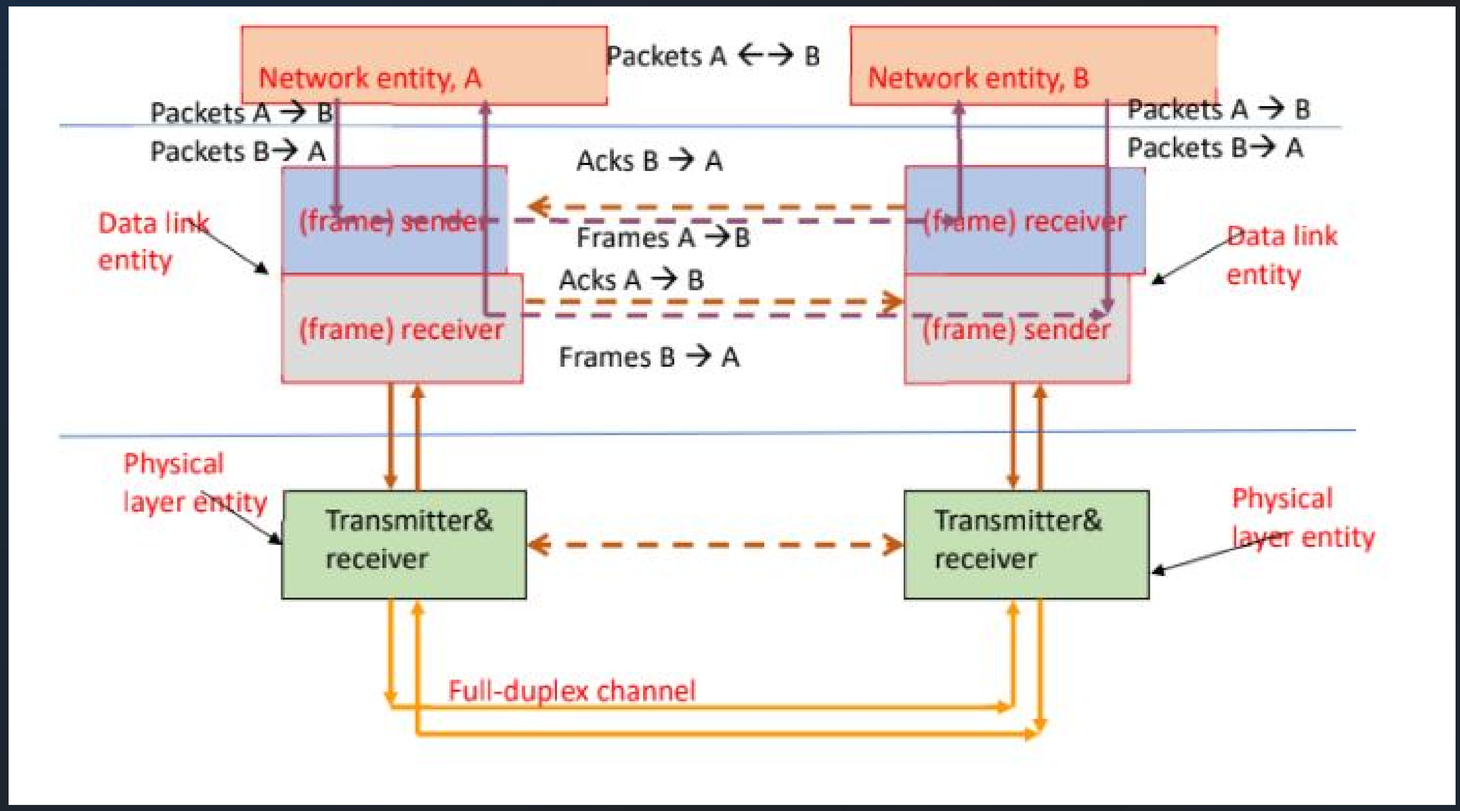
About Go-Back-N Protocol

The Go-Back-N (GBN) protocol is a data link layer protocol used for reliable packet delivery over unreliable
networks. It is a type of sliding window protocol designed to handle packet loss, corruption, and reordering.
The sliding window approach allows the sender to send multiple packets before waiting for an acknowledgment
(ACK) from the receiver, thus increasing throughput by keeping the channel busy while waiting for ACKs.
In Go-Back-N, the sender can transmit up to N packets (determined by the window size) without receiving an
ACK, allowing multiple frames to be in transit. However, if an error or timeout occurs for any packet within this
window, the sender retransmits that packet and all subsequent packets in sequence, hence the name "Go-Back-N."

Sliding Window Mechanism

The sliding window in Go-Back-N maintains:
• A sending window at the sender’s side that keeps track of packets sent but not yet acknowledged.
• A receiving window at the receiver’s side that only accepts packets in order, discarding out-of-sequence packets.
The sender’s window size is typically N , allowing up to N unacknowledged frames to be sent. The receiver, on the other hand, maintains a single-slot window, meaning it only accepts the next expected frame in sequence.
In our implementation, the window size is set to 7 with a modulo-8 sequence numbering scheme.

In this project, two entities—Entity 1 and Entity 2—simulate the sender and receiver using UDP sockets for communication, binding to separate ports:
• Entity 1 binds to port 5001 and sends data to Entity 2’s port (5002).
• Entity 2 binds to port 5002 and sends data to Entity 1’s port (5001).
Each entity is responsible for both sending and receiving packets and acknowledgments (ACKs), allowing bidirectional data exchange. In our simulation:



1. Network Entities (A and B)

• The diagram shows two network entities, A and B, each responsible for sending and receiving packets. The network entities operate at the network layer, passing data packets to the data link layer for reliable transmission.

• Each network entity sends packets to the other entity, so packets are transmitted in both directions (A to B and B to A).

2. Data Link Entities
• Within each network entity, there is a data link layer comprising two main components: a (frame) sender
and a (frame) receiver.
• The data link layer's job is to handle the transmission of frames (data packets) over the physical link,
ensuring reliable communication despite potential errors.
• Frame Sender: The sender in each entity handles the transmission of frames, utilizing the sliding window
mechanism of the Go-Back-N protocol.
2
• Frame Receiver: The receiver component acknowledges received frames. When a frame is correctly received,
it sends an acknowledgment (ACK) back to the sender.

3. Frame Transmission and Acknowledgments
• Frames A → B and Frames B → A: Frames are transmitted in both directions (A to B and B to A) through
the data link layer. Each frame has a sequence number that helps manage the order of frames in the sliding
window.
• Acks B → A and Acks A → B: After receiving frames correctly, each data link receiver sends ACKs to
confirm the successful reception of frames. This allows the sender to slide the window forward and send
new frames.
• Error Handling: If a frame is lost or an ACK is not received within a timeout period, the Go-Back-N
protocol requires the sender to retransmit that frame and all subsequent frames in the window.

4. Physical Layer Entities
• Each data link layer connects to a physical layer entity responsible for actual data transmission over a communication channel.
• Transmitter Receiver: Each network entity has a transmitter and receiver at the physical layer. This component converts frames into signals suitable for transmission over the physical channel and receives signals for interpretation back into frames.
• The communication channel between the physical layers is full-duplex, meaning it can simultaneously send and receive data in both directions (A to B and B to A).

Entity 1

```python
import time
import threading
import random
import socket
# Configuration parameters
T1, T2 = 1, 3  # Time interval bounds for packet generation
T3, T4 = 1, 2  # Delay bounds for packet processing
# Configuration parameters
MAX_SEQ = 7
TIMEOUT_DURATION = 5
DROP_PROBABILITY = 0.1
WINDOW_SIZE_SENT = 7
WINDOW_SIZE_RECV = 1
TOT_PACKETS = 10
# Socket setup for sending and receiving
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# client_socket.bind(('127.0.0.1', 6002))  # Client (sending) on port 5001
try:
    server_socket.bind(('127.0.0.1', 5002))  # Server (receiving) on port 6001
    print("Entity 1 server on port 5002")
except OSError:
    print("Unable to bind server socket. Is the port already in use?")

# Statistics tracking
class Statistics:
    def __init__(self):
        self.total_sent = 0
        self.total_received = 0
        self.total_acks_recieved = 0
        self.total_acks_sent = 0
        self.total_dropped = 0
        self.total_retransmitted = 0
        self.total_delay = 0

    def print_stats(self):
        print("\n--- Entity 2 Transmission Statistics ---")
        print(f"Total Frames Sent: {self.total_sent}")
        print(f"Total Frames Received: {self.total_received}")
        print(f"Total ACKs Received: {self.total_acks_recieved}")
        print(f"Total ACKs Sent: {self.total_acks_sent}")
        print(f"Total Frames Dropped: {self.total_dropped}")
        print(f"Total Frames Retransmitted: {self.total_retransmitted}")
        print(f"Average delay per packet: {self.total_delay/self.total_received}")
        print(f"Average Retransmission time: {self.total_delay/self.total_retransmitted}")
```

```python
stats = Statistics()
queue = []
# Frame class
class Frame:
    def __init__(self, seq=0, ack=0, timestamp=None):
        self.seq = seq
        self.ack = ack
        self.timestamp = timestamp or time.time()

    def to_string(self):
        return f"{self.seq} {self.ack}"

    def packet_generator(self):
        for _ in range(TOT_PACKETS):
            time_to_wait = random.uniform(T1, T2)
            time.sleep(time_to_wait)
            packet= self.seq + self.ack
            #add packet to queue
            queue.append(packet)
    @staticmethod
    def from_string(data):
        parts = data.split()
        if len(parts) == 2:
            # If only seq and ack are present, set timestamp to 0
            seq, ack = parts
            timestamp = 0
        else:
            # If all three parts are present, unpack them normally
            seq, ack, timestamp = parts
        return Frame(int(seq), int(ack), float(timestamp))
```

```python
class TimeoutManager:
    def __init__(self):
        self.timers = {}
    def start_timer(self, frame_num, callback):
        if frame_num in self.timers:
            self.cancel_timer(frame_num)
        timer = threading.Timer(TIMEOUT_DURATION, callback, [frame_num])
        self.timers[frame_num] = timer
        timer.start()

    def cancel_timer(self, frame_num):
        if frame_num in self.timers:
            self.timers[frame_num].cancel()
            del self.timers[frame_num]

timeout_manager = TimeoutManager()
# State variables
next_frame_to_send = 0
frame_expected = 0
ack_expected = 0
n_buffered = 0

# Helper function to check if a sequence number is within the window
def between(a, b, c):
    return ((a <= b < c) or (c < a <= b) or (b < c < a))
# Packet drop simulation
def should_drop_packet():
    return random.random() < DROP_PROBABILITY

# Function to send a frame
def send_frame(frame_num):
    time_to_wait = random.uniform(T1, T2)
    time.sleep(time_to_wait)

    if should_drop_packet():
        print(f"\tEntity 1: Frame {frame_num} dropped.")
        stats.total_dropped += 1
        timeout_manager.start_timer(frame_num, retransmit_frame)
        return

    # Create frame with timestamp
    frame = Frame(seq=frame_num, ack=(frame_expected + MAX_SEQ) % (MAX_SEQ + 1), timestamp=time.time())
    client_socket.sendto(frame.to_string().encode(), ('127.0.0.1', 6002))
    print(f"\tEntity 1 sent frame: Seq={frame.seq}, Ack={frame.ack}, Timestamp={frame.timestamp}")
    stats.total_sent += 1
    timeout_manager.start_timer(frame_num, retransmit_frame)

# Function to retransmit a frame on timeout
```

```python
def retransmit_frame(frame_num):
    print(f"\tEntity 1: Timeout occurred, retransmitting frame starting from {frame_num}")
    frame = frame_num
    while frame != next_frame_to_send:
        send_frame(frame)
        stats.total_retransmitted += 1
        frame = (frame + 1) % (MAX_SEQ + 1)

# Client thread function for sending packets and managing the sliding window
def client_thread():
    global next_frame_to_send, ack_expected, n_buffered
    while stats.total_sent < TOT_PACKETS:
        if n_buffered < WINDOW_SIZE_SENT:
            send_frame(next_frame_to_send)
            next_frame_to_send = (next_frame_to_send + 1) % (MAX_SEQ + 1)
            n_buffered += 1
            time.sleep(random.uniform(T3, T4))
        # Receiving ACKs
        try:
            ack_data, _ = client_socket.recvfrom(1024)
            frame = Frame.from_string(ack_data.decode())
            print(f"Entity 1 received ACK for Seq={frame.ack}, Timestamp={frame.timestamp}")

            while between(ack_expected, frame.ack, next_frame_to_send):
                timeout_manager.cancel_timer(ack_expected)
                ack_expected = (ack_expected + 1) % (MAX_SEQ + 1)
                n_buffered -= 1
                stats.total_acks_received += 1
        except socket.timeout:
            print("Entity 1: No ACK received, continuing to monitor.")

# Server thread function for receiving packets and sending ACKs
def server_thread():
    global frame_expected
    while stats.total_received < TOT_PACKETS:
        try:
            data, _ = server_socket.recvfrom(1024)
            frame = Frame.from_string(data.decode())
            print(f"Entity 1 received frame: Seq={frame.seq}, Ack={frame.ack}, Timestamp={frame.timestamp}")

            if frame.seq == frame_expected:
                print(f"Entity 1: Frame {frame.seq} received in order at {time.time()}.")
                frame_expected = (frame_expected + 1) % (MAX_SEQ + 1)
                stats.total_received += 1

                # Send an ACK for the received frame
                ack_frame = Frame(seq=0, ack=frame.seq, timestamp=time.time())
                server_socket.sendto(ack_frame.to_string().encode(), ('127.0.0.1', 5002))
                print(f"\tEntity 1 sent ACK for Seq={frame.seq}, Timestamp={ack_frame.timestamp}")
                stats.total_acks_sent += 1

        except socket.timeout:
            print("Entity 1: Socket timed out while waiting for frame.")
```

```python
# Set timeout for client socket
client_socket.settimeout(5)

# Start client and server threads
client = threading.Thread(target=client_thread)
server = threading.Thread(target=server_thread)

client.start()
server.start()

client.join()
server.join()

# Print statistics after communication
stats.print_stats()
```

Change in code for entity 2

```
try:
    server_socket.bind(('127.0.0.1', 6002))  # Server (receiving) on port 6002
    print("Entity 2 server on port 6002")
except OSError:
    print("Unable to bind server socket. Is the port already in use?")
```

```
# Server thread function for receiving packets and sending ACKs
def server_thread():
    global frame_expected
    while stats.total_received < TOT_PACKETS:
        try:
            data, _ = server_socket.recvfrom(1024)
            frame = Frame.from_string(data.decode())
            print(f"Entity 2 received frame: Seq={frame.seq}, Ack={frame.ack}, Timestamp={frame.timestamp}")

            if frame.seq == frame_expected:
                print(f"Entity 2: Frame {frame.seq} received in order.")
                frame_expected = (frame_expected + 1) % (MAX_SEQ + 1)
                stats.total_received += 1

                # Send an ACK for the received frame
                ack_frame = Frame(seq=0, ack=frame.seq, timestamp=time.time())
                server_socket.sendto(ack_frame.to_string().encode(), ('127.0.0.1', 6002))
                print(f"\tEntity 2 sent ACK for Seq={frame.seq}, Timestamp={ack_frame.timestamp}")
                stats.total_acks_sent += 1

        except socket.timeout:
            print("Entity 2: Socket timed out while waiting for frame.")
```

changing ports for different entity

Output Snippets

```
Progress: 1/10 packets acknowledged.
Sent: Seq 1, Packet-953
Dropped: Seq 2
Timeout: Retransmitting frames in window.
Retransmitted: Seq 1
Retransmitted: Seq 2
Received ACK: 1
Sent: Seq 3, Packet-624
Progress: 2/10 packets acknowledged.
Received ACK: 3
Progress: 4/10 packets acknowledged.
Sent: Seq 4, Packet-382
Received ACK: 4
Progress: 5/10 packets acknowledged
```

```
Received: Seq 0
Entity 2 received in order: SEQ=0
Sent ACK: 0
Received: Seq 1
Entity 2 received in order: SEQ=1
Dropped ACK: 1
Received: Seq 1
Entity 2 received out of order: SEQ=1, expected 2
Sent ACK: 1
Received: Seq 2
Entity 2 received in order: SEQ=2
Dropped ACK: 2
Received: Seq 3
Entity 2 received in order: SEQ=3
```

We initially started with a half duplex structure

Output Snippets

```
PS C:\Users\swarnima prasad\OneDrive\Desktop\CN_ass\CN_Assignments> cd Ass5
PS C:\Users\swarnima prasad\OneDrive\Desktop\CN_ass\CN_Assignments\Ass5> python e1
.py
Entity 1 server on port 5002
        Entity 1 sent frame: Seq=0, Ack=7
        Entity 1: Timeout occurred, retransmitting frame starting from 0
Entity 1: No ACK received, continuing to monitor.
Entity 1 received frame: Seq=0, Ack=0
Entity 1: Frame 0 received in order.
        Entity 1 sent ACK for Seq=0
Entity 1 received frame: Seq=0, Ack=0
        Entity 1 sent frame: Seq=0, Ack=0
Entity 1 received frame: Seq=1, Ack=0
Entity 1: Frame 1 received in order.
        Entity 1 sent ACK for Seq=1
Entity 1 received frame: Seq=0, Ack=1
        Entity 1 sent frame: Seq=1, Ack=1
        Entity 1: Timeout occurred, retransmitting frame starting from 0
        Entity 1: Timeout occurred, retransmitting frame starting from 1
        Entity 1 sent frame: Seq=0, Ack=1
Entity 1 received frame: Seq=0, Ack=1
Entity 1 received frame: Seq=1, Ack=1
Entity 1: No ACK received, continuing to monitor.
        Entity 1 sent frame: Seq=1, Ack=1
        Entity 1 sent frame: Seq=1, Ack=1
Entity 1 received frame: Seq=1, Ack=1
        Entity 1 sent frame: Seq=2, Ack=1
Entity 1 received frame: Seq=2, Ack=2
Entity 1: Frame 2 received in order.
        Entity 1 sent ACK for Seq=2
Entity 1 received frame: Seq=0, Ack=2
        Entity 1: Timeout occurred, retransmitting frame starting from 0
        Entity 1: Frame 0 dropped.
        Entity 1: Timeout occurred, retransmitting frame starting from 1
Entity 1 received frame: Seq=0, Ack=2
        Entity 1: Timeout occurred, retransmitting frame starting from 2
Entity 1 received frame: Seq=1, Ack=2
Entity 1 received frame: Seq=1, Ack=2
Entity 1: No ACK received, continuing to monitor.
```

Bidirectional Transmission

```
PS C:\Users\swarnima prasad\OneDrive\Desktop\CN_ass\CN_Assignments> cd Ass5
PS C:\Users\swarnima prasad\OneDrive\Desktop\CN_ass\CN_Assignments\Ass5> python
.py
Entity 2 server on port 6002
        Entity 2: Frame 0 dropped.
Entity 2 received frame: Seq=0, Ack=7
Entity 2: Frame 0 received in order.
        Entity 2 sent ACK for Seq=0
Entity 2 received frame: Seq=0, Ack=0
        Entity 2: Timeout occurred, retransmitting frames starting from 0
Entity 2: No ACK received, monitoring.
        Entity 2 sent frame: Seq=0, Ack=0
Entity 2 received frame: Seq=0, Ack=0
        Entity 2 sent frame: Seq=1, Ack=0
Entity 2 received frame: Seq=1, Ack=1
Entity 2: Frame 1 received in order.
        Entity 2 sent ACK for Seq=1
Entity 2 received frame: Seq=0, Ack=1
        Entity 2: Timeout occurred, retransmitting frames starting from 0
        Entity 2: Timeout occurred, retransmitting frames starting from 1
Entity 2 received frame: Seq=0, Ack=1
        Entity 2 sent frame: Seq=0, Ack=1
        Entity 2 sent frame: Seq=1, Ack=1
Entity 2: No ACK received, monitoring.
Entity 2 received frame: Seq=1, Ack=1
Entity 2 received frame: Seq=1, Ack=1
        Entity 2 sent frame: Seq=1, Ack=1
Entity 2 received frame: Seq=2, Ack=1
Entity 2: Frame 2 received in order.
        Entity 2 sent ACK for Seq=2
Entity 2 received frame: Seq=0, Ack=2
        Entity 2 sent frame: Seq=2, Ack=2
        Entity 2: Timeout occurred, retransmitting frames starting from 0
        Entity 2 sent frame: Seq=0, Ack=2
        Entity 2: Timeout occurred, retransmitting frames starting from 1
        Entity 2 sent frame: Seq=1, Ack=2
        Entity 2: Timeout occurred, retransmitting frames starting from 2
        Entity 2 sent frame: Seq=1, Ack=2
Entity 2 received frame: Seq=1, Ack=2
```

```
--- Entity 2 Transmission Statistics ---
Total Frames Sent: 10000
Total Frames Received: 10000
Total ACKs Received: 10000
Total ACKs Sent: 10000
Total Frames Dropped: 1737
Total Frames Retransmitted: 1988
Average delay per packet: 0.7
Average Retransmission time: 1.1
```

Statistics for 10000 frames

Output Snippets

```
Transmission Summary
Total packets sent: 9
Total successful acknowledgments: 10
Total frames retransmitted: 3
Total frames dropped: 1
Average delay per packet: 3.5948 seconds
Average retransmissions per packet: 1.20
Transmission completed.
Entity 1 received END from Entity 2. Ending protocol.
Entity 1 shutdown completed.
PS C:\Users\Ritika\OneDrive\Documents\Sem - V\CN_Assignme
```

Statistics for 10 frames

```
ritika@DESKTOP-V520N37:/mnt/c/Users/Ritika/OneDrive/Documents/Sem - V/CN_Assignments/Ass5$ py
thon3 Entity2.py
Entity 2 server on port 6002
Entity 2 received frame: Seq=0, Ack=7, Timestamp=1731258556.6410875
Entity 2: Frame 0 received in order.
        Entity 2 sent ACK for Seq=0, Timestamp=1731258556.6412697
Entity 2 received frame: Seq=0, Ack=0, Timestamp=1731258556.641355
        Entity 2 sent frame: Seq=0, Ack=0, Timestamp=1731258558.2923682
        Entity 2: Timeout occurred, retransmitting frames starting from 0
Entity 2 received frame: Seq=0, Ack=0, Timestamp=1731258563.4616032
Entity 2: No ACK received, monitoring.
        Entity 2: Frame 1 dropped.
        Entity 2 sent frame: Seq=0, Ack=0, Timestamp=1731258566.1078677
Entity 2 received frame: Seq=1, Ack=0, Timestamp=1731258566.1558669
Entity 2: Frame 1 received in order.
        Entity 2 sent ACK for Seq=1, Timestamp=1731258566.1560123
Entity 2 received frame: Seq=0, Ack=1, Timestamp=1731258566.1564903
        Entity 2 sent frame: Seq=1, Ack=1, Timestamp=1731258567.6879117
Entity 2 received frame: Seq=0, Ack=1, Timestamp=1731258570.367012
        Entity 2: Timeout occurred, retransmitting frames starting from 0
Entity 2 received frame: Seq=1, Ack=1, Timestamp=1731258571.8502543
Entity 2 received frame: Seq=1, Ack=1, Timestamp=1731258572.410827
```

```
ritika@DESKTOP-V520N37:/mnt/c/Users/Ritika/OneDrive/Documents/Sem - V/CN_Assignments/Ass5$ py
thon3 Entity1.py
Entity 1 server on port 5002
        Entity 1 sent frame: Seq=0, Ack=7, Timestamp=1731258556.640543
Entity 1 received frame: Seq=0, Ack=0, Timestamp=1731258558.2928827
Entity 1: Frame 0 received in order at 1731258558.2931194.
        Entity 1 sent ACK for Seq=0, Timestamp=1731258558.2933638
Entity 1 received frame: Seq=0, Ack=0, Timestamp=1731258558.293627
        Entity 1: Timeout occurred, retransmitting frame starting from 0
Entity 1: No ACK received, continuing to monitor.
        Entity 1 sent frame: Seq=0, Ack=0, Timestamp=1731258563.4601257
Entity 1 received frame: Seq=0, Ack=0, Timestamp=1731258566.1092265
        Entity 1 sent frame: Seq=1, Ack=0, Timestamp=1731258566.155373
Entity 1 received frame: Seq=1, Ack=1, Timestamp=1731258567.6882386
Entity 1: Frame 1 received in order at 1731258567.688293.
        Entity 1 sent ACK for Seq=1, Timestamp=1731258567.6883037
Entity 1 received frame: Seq=0, Ack=1, Timestamp=1731258567.688376
        Entity 1: Timeout occurred, retransmitting frame starting from 0
        Entity 1 sent frame: Seq=0, Ack=1, Timestamp=1731258570.366665
        Entity 1: Timeout occurred, retransmitting frame starting from 1
        Entity 1 sent frame: Seq=1, Ack=1, Timestamp=1731258571.8498852
Entity 1: No ACK received, continuing to monitor
```

Transmission with timestamps: We notice the delay between the first packet being sent and being received is approximately 0.0005445s

some stats

average time delay 1.6 seconds for 0.3 drop probability

average time delay 0.7 seconds for 0. 2drop probability

average retransmission: 1.1s for 0.1 drop probability

average retransmission: 1.4s for 0.3 drop probabiliy

We maintained two separate threads, one for client and one for server in both the entities. Both entities simulated the role of a server by binding themselves to two different sockets, and the clients of these entities sent frames to the server port of the other entity. Thus, each server thread received frames and sent acknowledgements while each client thread sent frames and received acks.