Go-Back-N Protocol Implementation

Ritika Thakur (2022408) — Swarnima Prasad (2022525)

1 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 acknowledgement (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."

2 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.

3 Entities and Packet Exchange

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 5002 and sends data to Entity 2's port (5002).
- Entity 2 binds to port 6002 and sends data to Entity 1's port (5001).

Each entity is responsible for both sending and receiving packets and acknowledgements (ACKs), allowing bidirectional data exchange.

4 Packet Transmission and Acknowledgment

4.1 Frame Format

Each frame contains:

- **Sequence Number (seq)**: Identifying the frame uniquely.
- **Acknowledgment Number (ack)**: Indicating received frames.
- **Timestamp (timestamp)**: Indicating the sending and receiving times of the frames.

Frames are represented as strings, where the sequence and acknowledgement numbers are separated by a space.

4.2 Sending and Receiving Packets

- **Sending Frames**: Both entities use `send_frame()` to construct and send frames. Each frame undergoes packet drop simulation based on a probability value.
- **Acknowledging Frames**: The receiver acknowledges frames by sending an ACK with the next expected sequence number. This acknowledgement allows the sender to slide its window, marking frames as successfully received.
- **Timeout and Retransmission**: Each sent frame starts a timer. If the timer expires before an ACK is received or because a packet was dropped, the `timeout_manager` triggers a retransmission for all frames from the last unacknowledged frame.

5 Timeout and Error Handling

To handle packet drops and delays:

- **Timeouts** are managed using Python's `threading.Timer`, with each frame starting a timer upon transmission. When a timeout occurs, the sender retransmits all frames from the last acknowledged frame up to the current `next_frame_to_send`.
- **Packet Drops** are simulated with a probability function, where frames are randomly dropped to test protocol resilience.

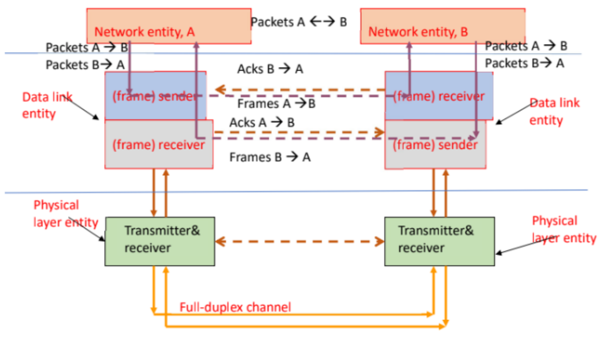


Figure 1: Figure

This diagram illustrates the architecture of the Go-Back-N (GBN) protocol in a full-duplex communication system which ahs been implemented in our code through the use of python libraries.

6 Client and Server threads

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.

7 Code Explanation

7.1 Configuration Parameters and Socket Setup

The code begins by defining configuration parameters such as time intervals for packet generation (`T1, T2`), processing delays (`T3, T4`), and other protocol constants. Sockets are created for communication between two entities.

Listing 1: Configuration parameters and socket setup

```
import time
import threading
import random
import socket

T1, T2 = 1, 3  # Time interval bounds for packet generation
T3, T4 = 1, 2  # Delay bounds for packet processing

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)

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?")
```

7.2 Statistics Tracking

A `Statistics` class is used to track the performance of the protocol, including the number of frames sent, received, dropped, and retransmitted.

Listing 2: Statistics class for tracking performance

```
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}")
```

7.3 Frame Class and Packet Generation

The `Frame` class represents a frame with sequence and acknowledgment numbers. It includes methods for converting frame data to strings and generating packets.

Listing 3: Frame class and packet generation

```
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
            queue.append(packet)

    @staticmethod
    def from_string(data):
        parts = data.split()
        if len(parts) == 2:
            seq, ack = parts
            timestamp = 0
        else:
            seq, ack, timestamp = parts
        return Frame(int(seq), int(ack), float(timestamp))
```

7.4 Timeout Manager

The `TimeoutManager` class handles timeouts for frames and manages retransmission when a timeout occurs.

Listing 4: Timeout Manager class for retransmissions

```
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]
```

7.5 Sending Frames and Simulating Packet Drops

The `send_frame` function sends frames, simulates packet drops based on probability, and initiates timers for frames that need retransmission.

Listing 5: Sending frames with packet drop simulation

```
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=t
    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.timesta
    stats.total_sent += 1
    timeout_manager.start_timer(frame_num, retransmit_frame)
```

7.6 Client and Server Threads

Two threads are defined: one for the client to send packets and handle acknowledgments, and one for the server to receive packets and send acknowledgments.

Listing 6: Client and Server threads

```
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))

        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.")

def server_thread():
    global frame_expected
    while stats.total_received < TOT_PACKETS:
        try:
```

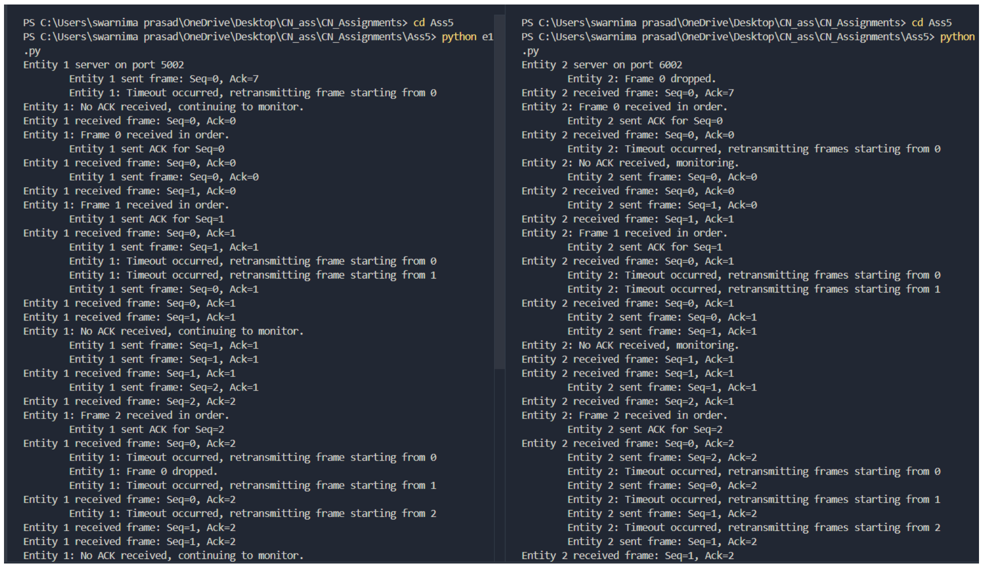
```
        data, _ = server_socket.recvfrom(1024)
        frame = Frame.from_string(data.decode())
        print(f"Entity 1 received frame: Seq={frame.seq}, Ack={frame.ack}, Timestamp={fra

        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.timesta
            stats.total_acks_sent += 1

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

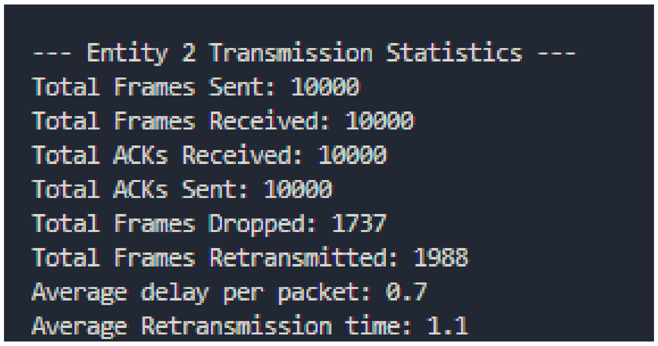
8 Ouput snippets



```
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.
```

```
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
```

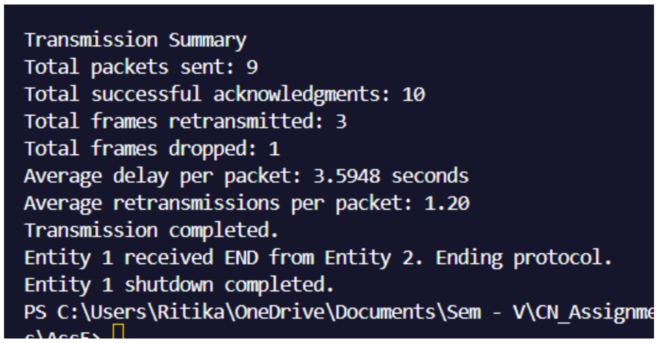
Figure 2: Bidirectional Transmission



```
--- 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
```

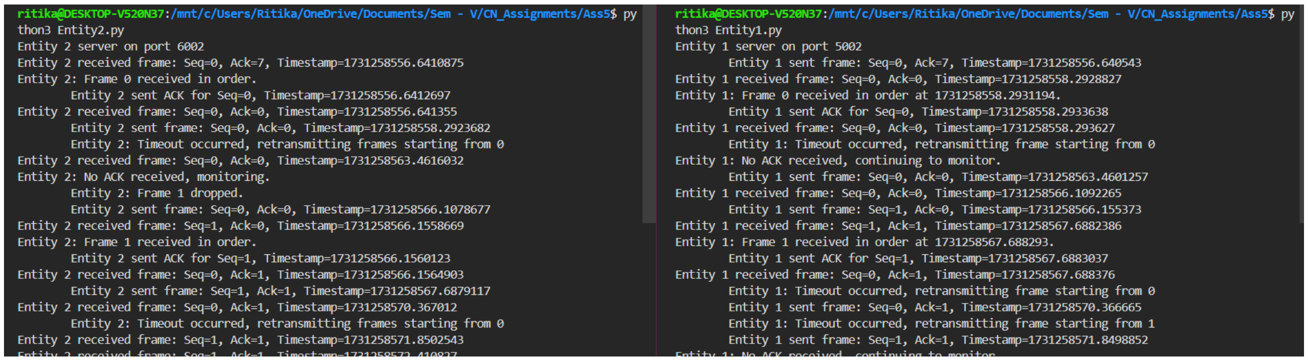
Figure 3: Statistics for 10000 frames for 0.1 drop probability

The average delay per packet was 1.6s and average RTT was 1.44 for drop probability = 0.3



```
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
```

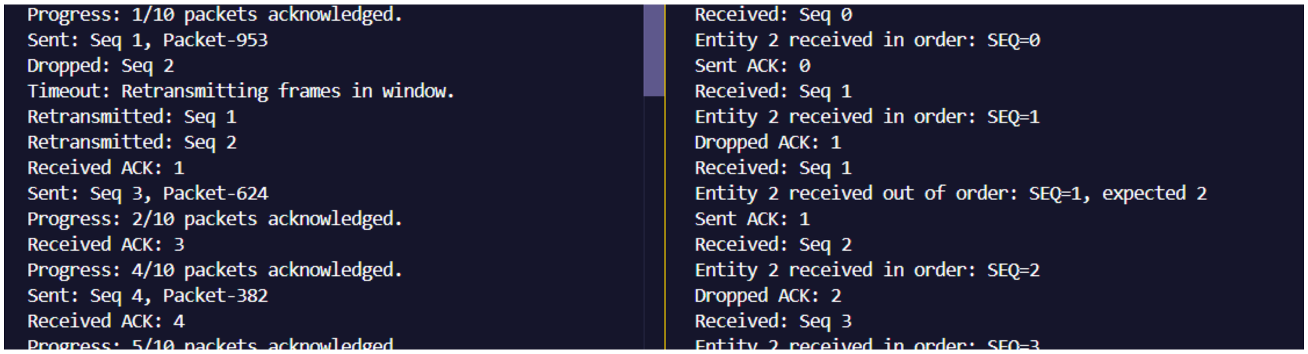
Figure 4: Statistics for 10 frames



```
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
```

```
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
```

Figure 5: Transmission with timestamps: We notice the delay between the first packet being sent and being received is approximately 0.0005445s



```
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
```

```
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
```

Figure 6: We initially started with a half duplex structure