Computer Networks Assignment-03

GO-BACK-N PROTOCOL IMPLEMENTATION USING UDP SOCKETS

Abhay Dagar
2022014

Aditya Sharma
2022038

INTRODUCTION

Goal

To simulate Go-Back-N ARQ protocol using UDP sockets between Sender and Receiver entities

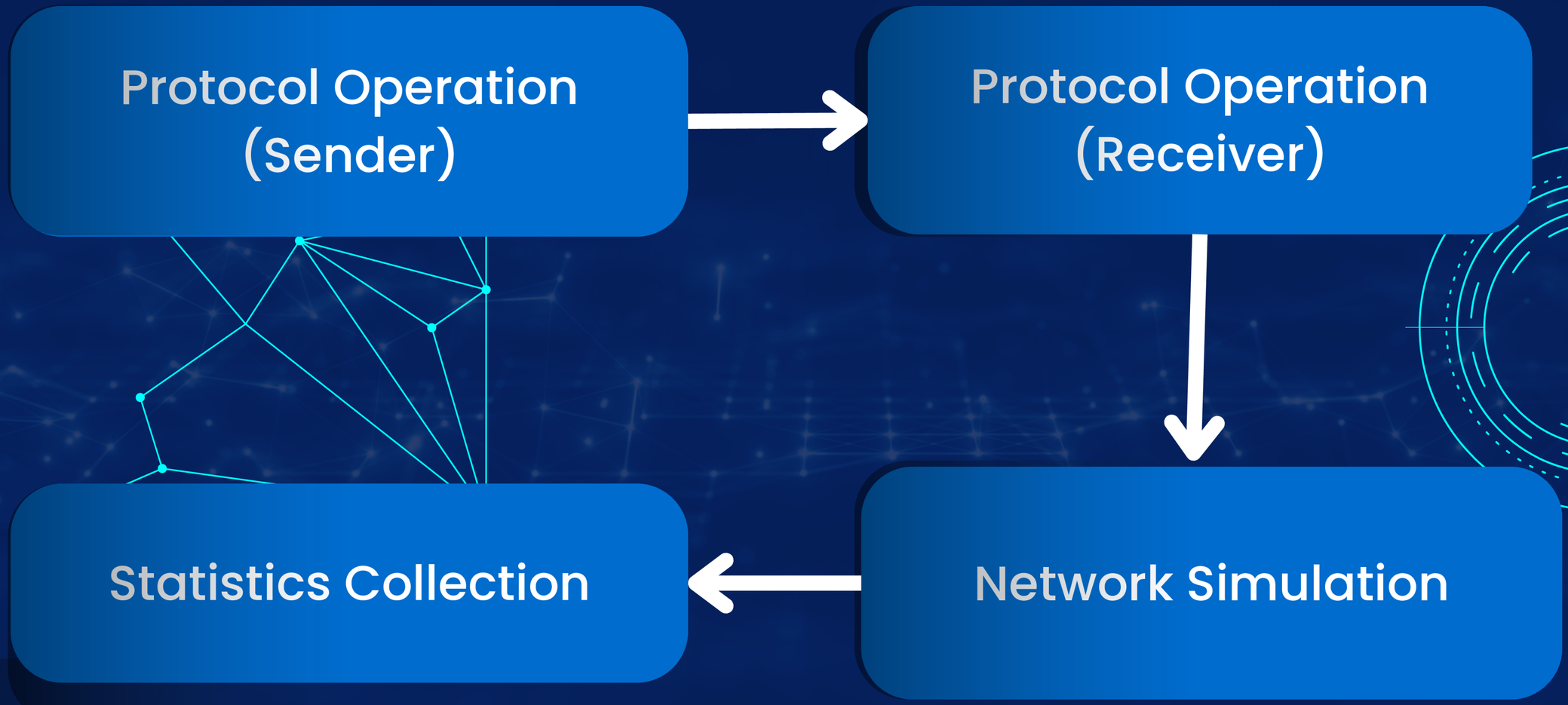
Visuals

Simple architecture diagram showing sender and receiver communicating

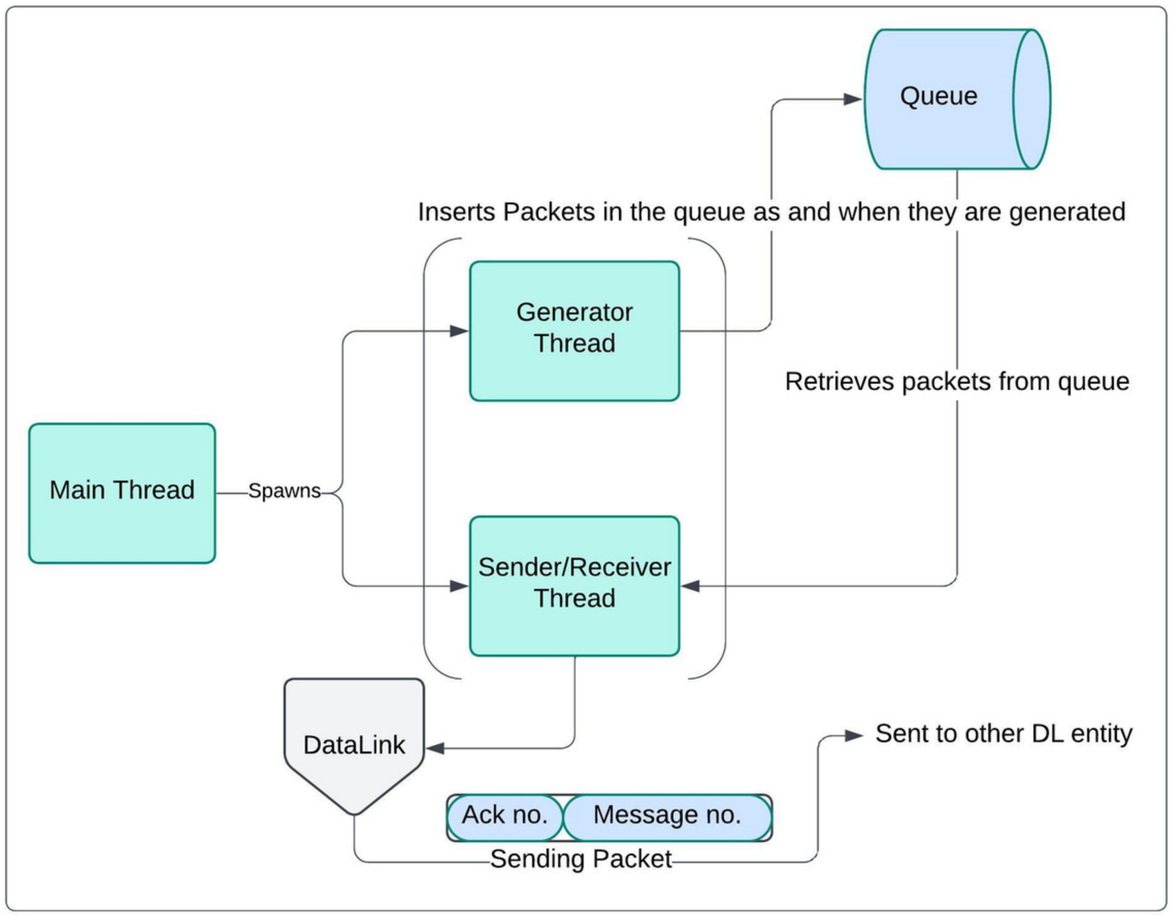
Assumptions

- UDP sockets are used with no guaranteed delivery, and sender/receiver are set to localhost with fixed IPs and ports.
- Packet generation and transmission delays are uniformly distributed within specified intervals (T1-T4). A finite number of packets are simulated.
- Fixed window size (7 frames) with modulo-8 sequence numbering is used.
- Timeout intervals are constant and not adaptive.
- Controlled Environment: Only a single sender-receiver pair is assumed, with simplified, mock data to focus on transmission and sequencing rather than content.

SYSTEM FLOW


Protocol Operation (Sender)
Protocol Operation (Receiver)
Network Simulation
Statistics Collection

System Architecture



DATA LINK ENTITY

Receiver (DL_Entity_2): Receives in-order frames, sends cumulative ACKs.

```python
def receive_frames(self):
    """Main loop for receiving frames"""
    print("Receiver started, waiting for frames...")
    while self.running:
        try:
            data, _ = self.socket.recvfrom(1024)
            frame_str = data.decode()

            if frame_str.startswith("SEQ:"):
                # Parse frame data
                seq_part, data_part = frame_str.split(",")
                seq_num = int(seq_part.split(":")[1])
                data = data_part.split(":")[1]

                print(f"Received frame {seq_num} with data: {data}")

                if seq_num == self.expected_seq_num % self.MOD:
                    print(f"Frame {seq_num} received in order")
                    self.send_ack(seq_num)
                    self.expected_seq_num += 1
                else:
                    print(f"Frame {seq_num} out of order, expected {self.expected_seq_num % self.MOD}")
                    if self.expected_seq_num > 0:
                        self.send_ack(self.expected_seq_num - 1)
        except Exception as e:
            print(f"Error in receiver: {e}")
```

Sender (DL_Entity_1): Manages packet generation, sliding window, ACK processing.

Key Parameters

Description: Explanation of key parameters.

- T1-T2: Packet generation interval
- T3-T4: Transmission delay bounds
- drop_prob: Packet/ACK loss probability
- WINDOW_SIZE and MOD (sequence numbering

```
--------------Statistics--------------
Parameters used in this simulation:
Window size: 7
Packet generation interval (T1): 0.1 seconds
Packet generation interval (T2): 0.3 seconds
Min transmission delay (T3): 0.05 seconds
Max transmission delay (T4): 0.15 seconds
Packet drop probability: 0.1
----------------------------------------
Total packets sent: 10
Total retransmissions: 2
Average delivery delay: 0.423 seconds
Average retransmissions per packet: 0.20
PS C:\Users\adity\Desktop\coding\CN-assignments\CN Assignm
```

Parameters Used

Protocol Operation (Sender)

Sender Operation:

- Packet Generation: Random intervals (T1-T2).
- Frame Transmission: Sliding window of 7, timeouts, and retransmissions.
- ACK Processing: Adjusts window based on received ACKs.

```python
def sender_thread(self):
    """Handle sending frames using Go-Back-N protocol"""
    while self.running:
        # Send new frames if window space available
        while (self.next_seq_num < self.send_base + self.WINDOW_SIZE and
               not self.outgoing_queue.empty()):
            data = self.outgoing_queue.get()
            frame = Frame(seq_num=self.next_seq_num % self.MOD, data=data)
            self.window_frames[self.next_seq_num] = (frame, time.time())
            self.send_frame(frame)
            self.next_seq_num += 1
            self.packets_sent += 1

        # Check for timeout and retransmit if necessary
        current_time = time.time()
        timeout = 2.0  # Adjust as needed

        for seq_num in range(self.send_base, self.next_seq_num):
            if seq_num in self.window_frames:
                frame, send_time = self.window_frames[seq_num]
                if current_time - send_time > timeout:
                    print(f"Timeout for frame {frame}")
                    # Retransmit all frames in window
                    for resend_seq in range(self.send_base, self.next_seq_num):
                        if resend_seq in self.window_frames:
                            resend_frame, _ = self.window_frames[resend_seq]
                            self.send_frame(resend_frame)
                            self.retransmissions += 1
                    break

        time.sleep(0.1)  # Prevent busy waiting
```

Protocol Operation (Receiver)

Receiver Operation:

- Frame Reception: Checks sequence numbers, buffers if out of order.
- ACK Transmission: Sends cumulative ACKs with simulated delays and losses.

```python
def send_ack(self, ack_num: int):
    """Send acknowledgment with simulated delay and drop probability"""
    if random.random() >= self.drop_prob:  # Not dropped
        delay = random.uniform(self.T3, self.T4)
        time.sleep(delay)
        ack_msg = f"ACK:{ack_num}"
        self.socket.sendto(ack_msg.encode(), self.peer_addr)
        print(f"Sent {ack_msg}")
    else:
        print(f"Dropped ACK {ack_num}")

def receive_frames(self):
    """Main loop for receiving frames"""
    print("Receiver started, waiting for frames...")
    while self.running:
        try:
            data, _ = self.socket.recvfrom(1024)
            frame_str = data.decode()

            if frame_str.startswith("SEQ:"):
                # Parse frame data
                seq_part, data_part = frame_str.split(",")
                seq_num = int(seq_part.split(":")[1])
                data = data_part.split(":")[1]

                print(f"Received frame {seq_num} with data: {data}")

                if seq_num == self.expected_seq_num % self.MOD:
                    print(f"Frame {seq_num} received in order")
                    self.send_ack(seq_num)
                    self.expected_seq_num += 1
                else:
                    print(f"Frame {seq_num} out of order, expected {self.expected_seq_num % self.MOD}")
                    if self.expected_seq_num > 0:
                        self.send_ack(self.expected_seq_num - 1)
        except Exception as e:
            print(f"Error in receiver: {e}")
```

Network Simulation

Simulated Network Effects:

- UDP Socket Communication: Unreliable transport simulation.
- Simulated Delay: Random transmission delays (T3-T4).
- Packet Drops: Based on drop_prob.

```
def main():
    # Create sender entity
    sender = Sender(
        host="localhost",
        port=5000,
        peer_host="localhost",
        peer_port=5001,
        T1=0.1,  # Min packet generation interval
        T2=0.3,  # Max packet generation interval
        T3=0.005,  # Min transmission delay
        T4=0.015,  # Max transmission delay
        drop_prob=0.01  # 1% packet drop probability
    )
```

Changing the value of Drop_prob , T3 and T4
All are reduced by scale of 1/10

Statistics Collection

Statistics Computed:

- Average Delivery Delay and Total Retransmissions
- Average Retransmissions per Packet

Explanation:

Collection during runtime to analyze efficiency.

```
--------------Statistics--------------
Parameters used in this simulation:
Window size: 7
Packet generation interval (T1): 0.1 seconds
Packet generation interval (T2): 0.3 seconds
Min transmission delay (T3): 0.05 seconds
Max transmission delay (T4): 0.15 seconds
Packet drop probability: 0.1
----------------------------------------
Total packets sent: 10000
Total retransmissions: 10002
Average delivery delay: 1.599 seconds
Average retransmissions per packet: 1.00
```

We Notice Very High Retransmisson as When we are using go-back- protocol as the retransmission stack up for:

1. Out of order packets
2. Packet lost because of drop probabilty.
3. Then the whole 7 message will have to be transmitted

Execution and Usage

Execution

1. Start Receiver: python receiver.py (Data_Link_entity_1)
2. Start Sender: python sender.py (Data_Link_entity_2)

Changing Parameters

- We reduce the drop probability
- We reduce the min and max transmission delay time.
- Therefore, we see a big decrease in the retransmission that happen

```
--------------Statistics--------------
Parameters used in this simulation:
Window size: 7
Packet generation interval (T1): 0.1 seconds
Packet generation interval (T2): 0.3 seconds
Min transmission delay (T3): 0.5 seconds
Max transmission delay (T4): 0.15 seconds
Packet drop probability: 0.01
----------------------------------------
Total packets sent: 10000
Total retransmissions: 3576
Average delivery delay: 0.957 seconds
Average retransmissions per packet: 0.36
```

The results after sending 10,000 packets with new parameters are as expected

THANK YOU FOR YOUR ATTENTION

Abhay Dagar 2022014
Aditya Sharma 2022038