PRN: 21070126039 Name: Jainil Patel Batch: AI/ML A2

CN LAB 6

Aim:

Implementation and Study of Sliding Window Protocols (Go-Back-N & Selective Repeat)

Go-Back-N ARQ Protocol

Code:

```
import time
import random

class GoBackNSimulation:
    def __init__(self, sender_data, packet_loss_prob, m, window_size):
        self.sender_data = sender_data
        self.packet_loss_prob = packet_loss_prob
        self.m = m
        self.window_size = window_size

    def simulate_go_back_n(self):
        time_unit = 1  # Time unit
        time_elapsed = 0

        window_base = 0
        next_seq_num = 0
        unacked_packets = set()  # Use a set to store unacknowledged packets

        while window_base < len(self.sender_data):
            # Send packets within the window
            for i in range(next_seq_num, min(next_seq_num + self.window_size,
len(self.sender_data))):
                packet_number = i
                print(f"\n --> Sending packet: {packet_number}")
                time.sleep(0.5)
                time_elapsed += time_unit

            # Simulate waiting for acknowledgments
            ack_received = True
            for i in range(next_seq_num, min(next_seq_num + self.window_size,
len(self.sender_data))):
                packet_number = i
                if random.random() > self.packet_loss_prob:
                    print(f"✓ ACK received for packet: {packet_number}")
                    unacked_packets.discard(packet_number)  # Remove from unacked set if
received
                else:
                    print(f"✗ ACK lost for packet: {packet_number}")
                    ack_received = False
                    unacked_packets.add(packet_number)  # Add to unacked set if lost

            if ack_received:
                print(f"Time elapsed: {self.window_size * time_unit} units")  # Sending
time
                window_base += self.window_size
                next_seq_num = window_base  # Move the window
            else:
                print("Resending unacked packets...")
                next_seq_num = min(unacked_packets)  # Resend from the smallest unacked
packet
                time_elapsed += len(unacked_packets) * time_unit  # Add retransmission time
```

```
        total_bits_sent = len(self.sender_data) * self.m
        print(f"\n\n🕒 Total time taken: {time_elapsed} seconds")
        print(f"Total bits sent: {total_bits_sent} bits")

# Example usage:
sender_data = list(range(1, 11))  # Example data with 10 packets
packet_loss_prob = 0.6
m = 3  # Example m value (3 bits for sequence numbers)
window_size = 4  # Example window size
simulator = GoBackNSimulation(sender_data, packet_loss_prob, m, window_size)
simulator.simulate_go_back_n()
```

Output:

```
 --> Sending packet: 0

 --> Sending packet: 1

 --> Sending packet: 2

 --> Sending packet: 3
✗ ACK lost for packet: 0
✗ ACK lost for packet: 1
✗ ACK lost for packet: 2
✓ ACK received for packet: 3
Resending unacked packets...

 --> Sending packet: 0

 --> Sending packet: 1

 --> Sending packet: 2

 --> Sending packet: 3
✗ ACK lost for packet: 0
✗ ACK lost for packet: 1
✓ ACK received for packet: 2
✗ ACK lost for packet: 3
Resending unacked packets...

 --> Sending packet: 0

 --> Sending packet: 1

 --> Sending packet: 2

 --> Sending packet: 3
✗ ACK lost for packet: 0
✗ ACK lost for packet: 1
✓ ACK received for packet: 2
✓ ACK received for packet: 3
Resending unacked packets...

 --> Sending packet: 0

 --> Sending packet: 1

 --> Sending packet: 2

 --> Sending packet: 3
✗ ACK lost for packet: 0
✓ ACK received for packet: 1
```

```
✗ ACK lost for packet: 2
✗ ACK lost for packet: 3
Resending unacked packets...

 --> Sending packet: 0

 --> Sending packet: 1

 --> Sending packet: 2

 --> Sending packet: 3
✓ ACK received for packet: 0
✓ ACK received for packet: 1
✓ ACK received for packet: 2
✗ ACK lost for packet: 3
Resending unacked packets...

 --> Sending packet: 3

 --> Sending packet: 4

 --> Sending packet: 5

 --> Sending packet: 6
✓ ACK received for packet: 3
✗ ACK lost for packet: 4
✗ ACK lost for packet: 5
✓ ACK received for packet: 6
Resending unacked packets...

 --> Sending packet: 4

 --> Sending packet: 5

 --> Sending packet: 6

 --> Sending packet: 7
✓ ACK received for packet: 4
✓ ACK received for packet: 5
✗ ACK lost for packet: 6
✗ ACK lost for packet: 7
Resending unacked packets...

 --> Sending packet: 6

 --> Sending packet: 7

 --> Sending packet: 8

 --> Sending packet: 9
✓ ACK received for packet: 6
✓ ACK received for packet: 7
✗ ACK lost for packet: 8
✗ ACK lost for packet: 9
Resending unacked packets...

 --> Sending packet: 8

 --> Sending packet: 9
✗ ACK lost for packet: 8
✓ ACK received for packet: 9
Resending unacked packets...

 --> Sending packet: 8
```

```
 --> Sending packet: 9
✗ ACK lost for packet: 8
✓ ACK received for packet: 9
Resending unacked packets...

 --> Sending packet: 8

 --> Sending packet: 9
✓ ACK received for packet: 8
✓ ACK received for packet: 9
Time elapsed: 4 units

 --> Sending packet: 4

 --> Sending packet: 5

 --> Sending packet: 6

 --> Sending packet: 7
✓ ACK received for packet: 4
✓ ACK received for packet: 5
✗ ACK lost for packet: 6
✗ ACK lost for packet: 7
Resending unacked packets...

 --> Sending packet: 6

 --> Sending packet: 7

 --> Sending packet: 8

 --> Sending packet: 9
✗ ACK lost for packet: 6
✓ ACK received for packet: 7
✗ ACK lost for packet: 8
✓ ACK received for packet: 9
Resending unacked packets...

 --> Sending packet: 6

 --> Sending packet: 7

 --> Sending packet: 8

 --> Sending packet: 9
✓ ACK received for packet: 6
✗ ACK lost for packet: 7
✗ ACK lost for packet: 8
✗ ACK lost for packet: 9
Resending unacked packets...

 --> Sending packet: 7

 --> Sending packet: 8

 --> Sending packet: 9
✓ ACK received for packet: 7
✓ ACK received for packet: 8
✗ ACK lost for packet: 9
Resending unacked packets...

 --> Sending packet: 9
✓ ACK received for packet: 9
Time elapsed: 4 units
```

```
 --> Sending packet: 8

 --> Sending packet: 9
✓ ACK received for packet: 8
✓ ACK received for packet: 9
Time elapsed: 4 units


🕒 Total time taken: 84 seconds
Total bits sent: 30 bits
```

Selective Repeat Protocol

Code:

```
import time
import random

class SelectiveRepeatSimulation:
    def __init__(self, sender_data, ack_success_prob, m, window_size):
        self.sender_data = sender_data
        self.ack_success_prob = ack_success_prob
        self.m = m
        self.window_size = window_size

    def simulate_selective_repeat(self):
        time_unit = 1  # Time unit
        time_elapsed = 0

        window_base = 0
        next_seq_num = 0
        acked_packets = []  # List to track acknowledged packets in the current window

        while window_base < len(self.sender_data):
            unacked_packets = set(range(next_seq_num, min(next_seq_num + self.window_size,
len(self.sender_data))))

            # Send packets within the window that are not acked
            for packet_number in unacked_packets:
                if packet_number not in acked_packets:
                    print(f"\n --> Sending packet: {packet_number}")
                    time.sleep(0.5)
                    time_elapsed += time_unit

            # Simulate waiting for acknowledgments for all packets in the current window
            for packet_number in unacked_packets.copy():
                if random.random() < self.ack_success_prob:
                    print(f"✓ ACK received for packet: {packet_number}")
                    acked_packets.append(packet_number)  # Add to acknowledged packets list
                    unacked_packets.remove(packet_number)  # Remove from unacked set if
received
                else:
                    print(f"✗ ACK lost for packet: {packet_number}")

            if not unacked_packets:
                print(f"Time elapsed: {self.window_size * time_unit} units")  # Sending
time
                window_base += self.window_size
                next_seq_num = window_base  # Move the window
                acked_packets = []  # Clear the acknowledged packets list for the new
window
            else:
                print("Resending unacked packets...")
                time_elapsed += len(unacked_packets) * time_unit  # Add retransmission time

        total_bits_sent = len(self.sender_data) * self.m
```

```
        print(f"\n\n🕒 Total time taken: {time_elapsed} seconds")
        print(f"Total bits sent: {total_bits_sent} bits")

# Example usage:
sender_data = list(range(1, 11))  # Example data with 10 packets
ack_success_prob = 0.7  # Example probability of successful acknowledgment (90% success
rate)
m = 3  # Example m value (3 bits for sequence numbers)
window_size = 4  # Example window size
simulator = SelectiveRepeatSimulation(sender_data, ack_success_prob, m, window_size)
simulator.simulate_selective_repeat()
```

Output:

```
--> Sending packet: 0

 --> Sending packet: 1

 --> Sending packet: 2

 --> Sending packet: 3
✓ ACK received for packet: 0
✗ ACK lost for packet: 1
✗ ACK lost for packet: 2
✓ ACK received for packet: 3
Resending unacked packets...

 --> Sending packet: 1

 --> Sending packet: 2
✓ ACK received for packet: 0
✓ ACK received for packet: 1
✗ ACK lost for packet: 2
✓ ACK received for packet: 3
Resending unacked packets...

 --> Sending packet: 2
✓ ACK received for packet: 0
✗ ACK lost for packet: 1
✓ ACK received for packet: 2
✓ ACK received for packet: 3
Resending unacked packets...
✓ ACK received for packet: 0
✓ ACK received for packet: 1
✓ ACK received for packet: 2
✓ ACK received for packet: 3
Time elapsed: 4 units

 --> Sending packet: 4

 --> Sending packet: 5

 --> Sending packet: 6

 --> Sending packet: 7
✗ ACK lost for packet: 4
✓ ACK received for packet: 5
✓ ACK received for packet: 6
✓ ACK received for packet: 7
Resending unacked packets...
```

```
 --> Sending packet: 4
✓ ACK received for packet: 4
✗ ACK lost for packet: 5
✓ ACK received for packet: 6
✓ ACK received for packet: 7
Resending unacked packets...
✓ ACK received for packet: 4
✓ ACK received for packet: 5
✓ ACK received for packet: 6
✓ ACK received for packet: 7
Time elapsed: 4 units

 --> Sending packet: 8

 --> Sending packet: 9
✗ ACK lost for packet: 8
✓ ACK received for packet: 9
Resending unacked packets...

 --> Sending packet: 8
✗ ACK lost for packet: 8
✓ ACK received for packet: 9
Resending unacked packets...

 --> Sending packet: 8
✗ ACK lost for packet: 8
✓ ACK received for packet: 9
Resending unacked packets...

 --> Sending packet: 8
✓ ACK received for packet: 8
✓ ACK received for packet: 9
Time elapsed: 4 units


🕓 Total time taken: 26 seconds
Total bits sent: 30 bits
```

Self-Assessment :

Q1. **Frame an algorithm for the Selective Repeat protocol as done in the above- mentioned manner for the Go-Back-N protocol.**

Ans-

Algorithm: Selective Repeat Protocol

Initialization:

1. Initialize sender and receiver windows, buffers, timers, and control variables.
2. Set sender's window size (N).
3. Set receiver's window size (N) as well, which remains fixed.

Frame Transmission:

4. Sender transmits frames sequentially, starting from sequence number 0.
5. Each frame contains a sequence number that the receiver will acknowledge.

Receiver Processing:

6. Receiver receives frames in any order.
7. If a frame is in order (i.e., its sequence number matches the expected sequence number), it is immediately placed in the receiver's buffer.
8. If a frame is out of order, it is buffered until missing frames arrive.
9. Receiver acknowledges received frames by sending ACKs. The ACK may include sequence numbers of all successfully received frames.

Sender Processing:

10. Sender maintains a timer for each unacknowledged frame in its window.
11. When a frame is transmitted, its timer is started.
12. Sender waits for ACKs for frames in its window.
13. If an ACK is received for a frame, the sender clears its timer for that frame.

Retransmission Handling:

14. If a sender's timer for a particular frame expires, it assumes that the frame was lost or corrupted.
15. The sender retransmits only the frame(s) for which the timer has expired.
16. The receiver can receive duplicate frames, but it discards them because it has already acknowledged them.

Negative Acknowledgment (NAK) Handling:

17. If the receiver detects a gap in the received sequence numbers or a corrupted frame, it may send a NAK to request retransmission of specific frame(s).

Timeouts and Retransmissions:

18. Both sender and receiver use timeouts to manage frame retransmissions.
19. When a sender's timer for a frame expires, it is retransmitted.
20. The receiver may also have timeouts for detecting missing frames and requesting retransmissions through NAKs.

Buffer Management:

21. Sender and receiver maintain buffers for frames within their respective windows.
22. The receiver buffers out-of-order frames until any missing frames arrive.
23. The sender buffers unacknowledged frames for possible retransmission.

Completion:

24. The protocol continues until all frames have been successfully received and acknowledged.

Q.2. Imagine a scenario where a Go-Back-N sender with a window size of 3 is communicating with a Selective Repeat receiver with a window size of 4. If the sender has sent frames 1 to 6, and the receiver has acknowledged frames 1 to 3, what actions will both the sender and receiver take?

Ans ->

In this scenario, we have a Go-Back-N sender with a window size of 3 and a Selective Repeat receiver with a window size of 4. The sender has sent frames 1 to 6, and the receiver has acknowledged frames 1 to 3. Let's go through the actions both the sender and receiver will take:

Sender's Actions:

1. Sender's window size is 3 (as per Go-Back-N), so it can transmit up to frame 3 (frames 1, 2, and 3) without waiting for acknowledgments.
2. Frames 4, 5, and 6 are in the sender's buffer, but they have not been acknowledged yet.
3. The sender's timer for frames 1, 2, and 3 is running, waiting for acknowledgments. Since frames 1 to 3 have been acknowledged, their timers are cleared.
4. The sender does not transmit any new frames because its window size (3) is full, and it's waiting for acknowledgments for frames 4, 5, and 6.

Receiver's Actions:

1. The receiver has a window size of 4 (Selective Repeat). It can receive frames out of order and store them in its buffer.

2. The receiver has acknowledged frames 1 to 3, indicating that it has successfully received and processed them.
3. When the receiver receives frames 4, 5, and 6, it will buffer them because they are out of order and wait for frame 7 to complete its window.
4. The receiver sends cumulative acknowledgments for frames 1 to 3, indicating that it has successfully received and processed them. However, it doesn't send individual acknowledgments for frames 4, 5, and 6 because they are out of order.
5. The receiver does not discard frames 4, 5, and 6, even though they are out of order. It will wait for frame 7 to complete its window and acknowledge all frames in order.
6. The receiver may use timers to detect missing frames and request retransmissions if necessary. However, since frames 4, 5, and 6 are not yet acknowledged, they are not retransmitted by the sender.

In summary, the sender is waiting for acknowledgments for frames 4, 5, and 6, while the receiver has acknowledged frames 1 to 3 and is waiting for frame 7 to complete its window before acknowledging frames 4, 5, and 6. Frames 4, 5, and 6 are buffered by the receiver, but they are not considered acknowledged or delivered until they are acknowledged in order.

Conclusion:

In conclusion, Go-Back-N (GBN) and Selective Repeat (SRP) are both sliding window protocols used for reliable data transmission in computer networks, each with its own set of advantages and use cases. GBN allows the sender to transmit multiple frames in sequence without waiting for acknowledgments, simplifying the protocol but potentially causing inefficient bandwidth usage in the presence of errors. In contrast, SRP offers finer-grained control by allowing the selective retransmission of only lost or corrupted frames, making it more efficient for high-error environments. SRP, with its fixed-size receiver window and buffer management, ensures out-oforder delivery while reducing network congestion. The choice between GBN and SRP depends on the specific network conditions and requirements, with GBN suited for lower-error networks and SRP preferred for environments where efficient bandwidth utilization and reliability are paramount. Both protocols play vital roles in achieving reliable data transfer in modern networking applications.