Batch: A1	**Roll No.: 16010123012**
Experiment / assignment / tutorial No.: 05	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Experiment No.:5

TITLE: Flow control Mechanism: Go-Back-N ARQ Sliding Window Protocol using Socket programming

AIM: Implementation of Flow Control Mechanism: Stop and Wait ARQ / Go-Back- N/ Selective Repeat Sliding Window Protocol ARQ using sockets.

Expected Outcome of Experiment:
CO:

Books/ Journals/ Websites referred:

1. A. S. Tanenbaum, "Computer Networks", Pearson Education, Fourth Edition
2. B. A. Forouzan, "Data Communications and Networking", TMH, Fourth Edition

Pre-Lab/ Prior Concepts:
Java Socket Programming, Flow Control, Go-Back-Stop and Wait

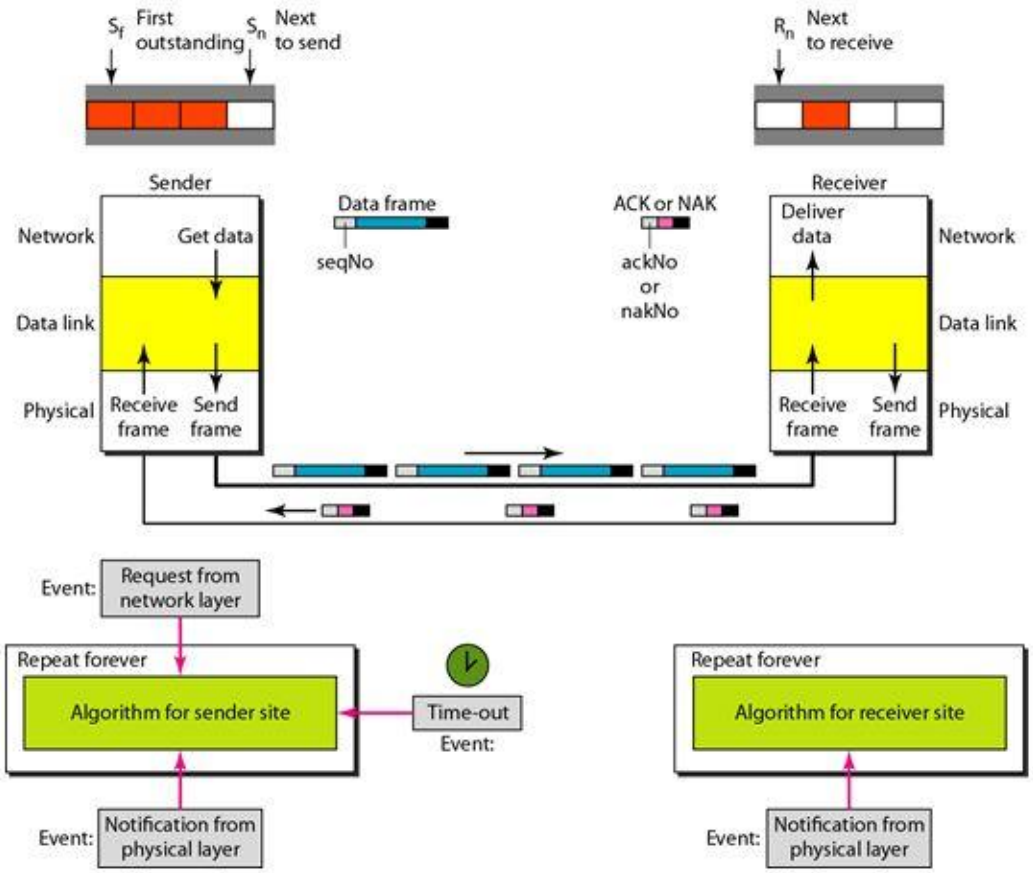
New Concepts to be learned: Window Flow Control

Go-Back-N ARQ is mainly a specific instance of Automatic Repeat Request (ARQ) protocol where the sending process continues to send a number of frames as specified by the window size even without receiving an acknowledgement(ACK) packet from the receiver. The sender keeps a copy of each frame until the arrival of acknowledgement. This protocol is a practical approach to the sliding window.

- In Go-Back-N ARQ, the size of the sender is N and the size of the receiver window is always 1.
- This protocol makes the use of **cumulative acknowledgements** means here the receiver maintains an acknowledgement timer; whenever the receiver receives a new frame from the sender then it starts a new acknowledgement timer. When the timer expires then the receiver sends the **cumulative acknowledgement** for all the frames that are unacknowledged by the receiver at that moment.
- It is important to note that the new acknowledgement timer only starts after the receiving of a new frame, it does not start after the expiry of the **old acknowledgement timer**.

- If the receiver receives a corrupted frame, then it silently discards that corrupted frame and the correct frame is retransmitted by the sender after the timeout timer expires. Thus receiver silently discards the corrupted frame. By discarding silently we mean that: “Simply rejecting the frame and not taking any action for the frame".
- In case after the expiry of the acknowledgement timer, suppose there is only one frame that is left to be acknowledged. In that case, the receiver sends the independent acknowledgement for that frame.
- In case if the receiver receives the out of order frame then it simply discards all the frames.
- In case if the sender does not receive any acknowledgement then the entire window of the frame will be retransmitted in that case.
- Using the Go-Back-N ARQ protocol leads to the retransmission of the lost frames after the expiry of the **timeout timer.**

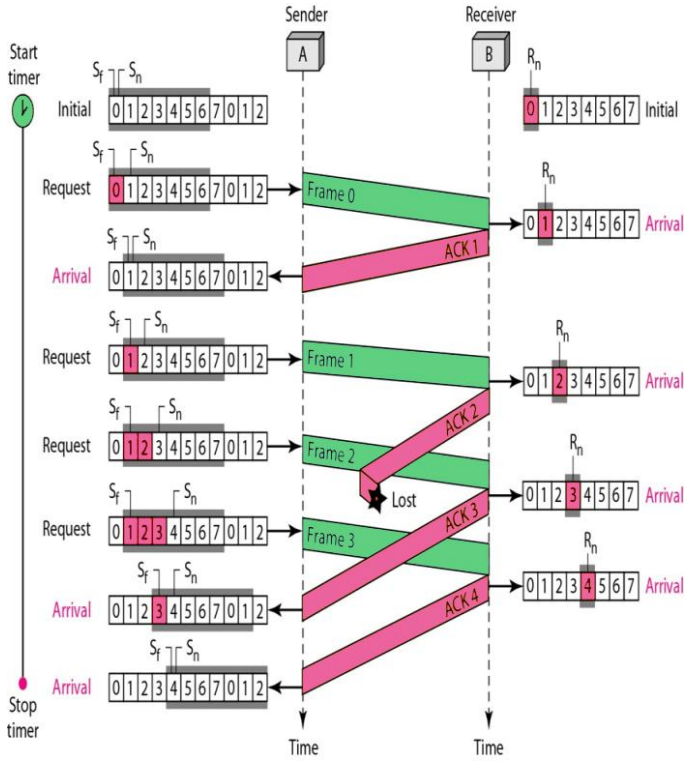
Design of Go-Back-N ARQ



1. Take data from user about how many bit windows is case of go back n and selective repeat.

2. Generate frames randomly and show the transmission
3. Generate the random number for the frame to be lost.
4. For Go – Back – N transmit all the frames after that number till max number
5. For Selective repeat transmit the selected frame which is not received by the receiver.

Flow Diagram:



IMPLEMENTATION: (printout of code)
Server (Receiver):

```
import socket
import json
from enum import Enum

class PacketType(Enum):
    DATA = "DATA"
    ACK = "ACK"

class GoBackNReceiver:
    def __init__(self, host='localhost', port=8081):
        self.host = host
        self.port = port
        self.expected_seq = 1
        self.socket = None
        self.total_packets = 0
        self.received_packets = 0
```

```
    def start_receiver(self):
        """Start the receiver server"""
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

        try:
            self.socket.bind((self.host, self.port))
            self.socket.listen(1)
            print(f"Go-Back-N Receiver listening on
{self.host}:{self.port}")
            print("Waiting for sender connection...\n")
            conn, addr = self.socket.accept()
            print(f"Connected to sender at {addr}")
            print("=" * 60)
            print(f"{'PACKET':<10} {'EXPECTED':<10} {'STATUS':<15}
{'ACTION'}")
            print("=" * 60)

            self.handle_packets(conn)

        except Exception as e:
            print(f"Receiver error: {e}")
        finally:
            if self.socket:
                self.socket.close()

    def handle_packets(self, conn):
        """Handle incoming packets from sender"""
        try:
            while True:
                data = conn.recv(1024).decode('utf-8')
                if not data:
                    break
                try:
                    packet = json.loads(data)
                    if packet['type'] == PacketType.DATA.value:
                        self.process_data_packet(packet, conn)
                    elif packet['type'] == 'END':
                        print("\n" + "=" * 60)
                        print("Transmission complete!")
                        print(f"Total packets received correctly:
{self.received_packets}")
                        break
                except json.JSONDecodeError:
                    continue
```

```
        except Exception as e:
            print(f"Error handling packets: {e}")
        finally:
            conn.close()

    def process_data_packet(self, packet, conn):
        """Process received data packet and send ACK"""
        seq_num = packet['seq_num']

        if seq_num == self.expected_seq:
            print(f"{seq_num:<10} {self.expected_seq:<10} {'ACCEPTED':<15}
Send ACK {seq_num}")
            self.expected_seq += 1
            self.received_packets += 1
            self.send_ack(conn, seq_num)
        else:
            last_correct_ack = self.expected_seq - 1
            print(f"{seq_num:<10} {self.expected_seq:<10}
{'DISCARDED':<15} Send ACK {last_correct_ack}")
            self.send_ack(conn, last_correct_ack)

    def send_ack(self, conn, ack_num):
        """Send acknowledgment packet"""
        ack_packet = {
            'type': PacketType.ACK.value,
            'ack_num': ack_num
        }
        try:
            conn.send(json.dumps(ack_packet).encode('utf-8'))
        except Exception as e:
            print(f"Error sending ACK: {e}")

if __name__ == "__main__":
    receiver = GoBackNReceiver()
    try:
        receiver.start_receiver()
    except KeyboardInterrupt:
        print("\nReceiver stopped by user")
    except Exception as e:
        print(f"Receiver error: {e}")
```

```
PS D:\KJSCE\BTech\TY\Sem V\CN> cd "d:\KJSCE\BTech\TY\Sem V\CN"; python gobackn_receiver.py
Go-Back-N Receiver listening on localhost:8081
Waiting for sender connection...

Connected to sender at ('127.0.0.1', 59669)
============================================================
PACKET     EXPECTED   STATUS          ACTION
============================================================
1          1          ACCEPTED        Send ACK 1
2          2          ACCEPTED        Send ACK 2
3          3          ACCEPTED        Send ACK 3
4          4          ACCEPTED        Send ACK 4
3          5          DISCARDED       Send ACK 4
4          5          DISCARDED       Send ACK 4
6          6          ACCEPTED        Send ACK 6
8          7          DISCARDED       Send ACK 6
7          7          ACCEPTED        Send ACK 7
8          8          ACCEPTED        Send ACK 8
9          9          ACCEPTED        Send ACK 9

============================================================
Transmission complete!
Total packets received correctly: 9
```

Client (Sender):

```python
import socket
import threading
import time
import json
from enum import Enum

class PacketType(Enum):
    DATA = "DATA"
    ACK = "ACK"

class GoBackNSender:
    def __init__(self, host='localhost', port=8081, total_packets=9,
window_size=3, loss_rate=5):
        self.host = host
        self.port = port
        self.total_packets = total_packets
        self.window_size = window_size
        self.loss_rate = loss_rate
        self.base = 1
        self.next_seq_num = 1
        self.sender_attempts = 0
        self.received_acks = 0
        self.socket = None
        self.ack_received = threading.Event()
        self.lock = threading.Lock()
```

```
        self.running = True
        self.timeout = 2.0
        self.timer = None

    def connect_to_receiver(self):
        """Connect to the receiver"""
        try:
            self.socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            self.socket.connect((self.host, self.port))
            print(f"Connected to Go-Back-N Receiver at
{self.host}:{self.port}")
            return True
        except Exception as e:
            print(f"Failed to connect to receiver: {e}")
            return False

    def start_sender(self):
        """Start the Go-Back-N sender"""
        if not self.connect_to_receiver():
            return

        print("\nGo-Back-N Protocol Simulation")
        print("=" * 70)
        print(f"Total Packets: {self.total_packets}, Window Size:
{self.window_size}, Loss Pattern: Every {self.loss_rate}th packet")
        print("=" * 70)
        print(f"{'ATTEMPT':<10} {'SEND_PKT':<10} {'STATUS':<15}
{'ACTION'}")
        print("=" * 70)

        ack_thread = threading.Thread(target=self.receive_acks)
        ack_thread.daemon = True
        ack_thread.start()

        while self.base <= self.total_packets and self.running:
            while (self.next_seq_num < self.base + self.window_size and
                   self.next_seq_num <= self.total_packets):
                self.send_packet(self.next_seq_num)
                self.next_seq_num += 1
            if self.base <= self.total_packets:
                self.ack_received.wait(timeout=self.timeout)
                self.ack_received.clear()

                if self.base <= self.total_packets and self.running:
```

```
                    print(f"{'TIMEOUT':<10} {'--':<10} {'TIMEOUT':<15}
Retransmit from {self.base}")
                    self.next_seq_num = self.base

        if self.running:
            end_packet = {'type': 'END'}
            try:
                self.socket.send(json.dumps(end_packet).encode('utf-8'))
            except Exception:
                pass

        print("\n" + "=" * 70)
        print(f"Final attempts (sender transmissions):
{self.sender_attempts}")
        print(f"Final ACKs received: {self.received_acks}")

        self.cleanup()

    def send_packet(self, seq_num):
        """Send a data packet"""
        self.sender_attempts += 1
        packet_lost = (self.sender_attempts % self.loss_rate == 0)

        packet = {
            'type': PacketType.DATA.value,
            'seq_num': seq_num,
            'data': f"Data packet {seq_num}"
        }

        if packet_lost:
            print(f"{self.sender_attempts:<10} {seq_num:<10} {'LOST':<15}
Packet dropped")
        else:
            try:
                self.socket.send(json.dumps(packet).encode('utf-8'))
                print(f"{self.sender_attempts:<10} {seq_num:<10}
{'SENT':<15} Packet transmitted")
                self.last_packet_time = time.time()
            except Exception as e:
                print(f"Error sending packet {seq_num}: {e}")
                self.running = False

    def receive_acks(self):
        """Receive ACK packets from receiver"""
        try:
```

```
            while self.running:
                data = self.socket.recv(1024).decode('utf-8')
                if not data:
                    break

                try:
                    packet = json.loads(data)
                    if packet['type'] == PacketType.ACK.value:
                        ack_num = packet['ack_num']
                        self.process_ack(ack_num)
                except json.JSONDecodeError:
                    continue

        except Exception as e:
            if self.running:
                print(f"Error receiving ACKs: {e}")

    def process_ack(self, ack_num):
        """Process received ACK"""
        with self.lock:
            if ack_num >= self.base:
                print(f"{'ACK':<10} {ack_num:<10} {'RECEIVED':<15} ACK for
packet {ack_num}")
                self.base = ack_num + 1
                self.received_acks += 1
                self.ack_received.set()
                if self.base == self.next_seq_num:
                    if self.timer:
                        self.timer.cancel()
            else:
                print(f"{'ACK':<10} {ack_num:<10} {'DUPLICATE':<15}
Duplicate ACK ignored")

    def cleanup(self):
        """Clean up resources"""
        self.running = False
        if self.timer:
            self.timer.cancel()
        if self.socket:
            self.socket.close()

def main():
    print("Go-Back-N Sender")
    print("================")
```

```
    use_defaults = input("Use default settings? (y/n): ").lower().strip()

    if use_defaults == 'y':
        sender = GoBackNSender()
    else:
        try:
            host = input("Enter receiver host (localhost): ").strip() or
'localhost'
            port = int(input("Enter receiver port (8081): ").strip() or
8081)
            packets = int(input("Enter total packets (9): ").strip() or 9)
            window = int(input("Enter window size (3): ").strip() or 3)
            loss_rate = int(input("Enter loss pattern - every Kth packet
(5): ").strip() or 5)

            sender = GoBackNSender(host, port, packets, window, loss_rate)
        except ValueError:
            print("Invalid input. Using default settings.")
            sender = GoBackNSender()

    try:
        sender.start_sender()
    except KeyboardInterrupt:
        print("\nSender stopped by user")
        sender.cleanup()
    except Exception as e:
        print(f"Sender error: {e}")
        sender.cleanup()

if __name__ == "__main__":
    main()
```

```
PS D:\KJSCE\BTech\TY\Sem V\CN> python gobackn_sender.py
Go-Back-N Sender
================
Use default settings? (y/n): y
Connected to Go-Back-N Receiver at localhost:8081

Go-Back-N Protocol Simulation
======================================================================
Total Packets: 9, Window Size: 3, Loss Pattern: Every 5th packet
======================================================================
ATTEMPT    SEND_PKT   STATUS          ACTION
======================================================================
1          1          SENT            Packet transmitted
2          2          SENT            Packet transmitted
ACK        1          RECEIVED        ACK for packet 1
3          3          SENT            Packet transmitted
ACK        2          RECEIVED        ACK for packet 2
4          4          SENT            Packet transmitted
ACK        3          RECEIVED        ACK for packet 3
5          5          LOST            Packet dropped
ACK        4          RECEIVED        ACK for packet 4
6          6          SENT            Packet transmitted
ACK        4          DUPLICATE       Duplicate ACK ignored
7          7          SENT            Packet transmitted
ACK        4          DUPLICATE       Duplicate ACK ignored
TIMEOUT    --         TIMEOUT         Retransmit from 5
8          5          SENT            Packet transmitted
ACK        5          RECEIVED        ACK for packet 5
9          6          SENT            Packet transmitted
ACK        6          RECEIVED        ACK for packet 6
10         7          LOST            Packet dropped
ACK        6          DUPLICATE       Duplicate ACK ignored
12         9          SENT            Packet transmitted
ACK        6          DUPLICATE       Duplicate ACK ignored
TIMEOUT    --         TIMEOUT         Retransmit from 7
13         7          SENT            Packet transmitted
ACK        7          RECEIVED        ACK for packet 7
14         8          SENT            Packet transmitted
ACK        8          RECEIVED        ACK for packet 8
15         9          LOST            Packet dropped
TIMEOUT    --         TIMEOUT         Retransmit from 9
16         9          SENT            Packet transmitted
ACK        9          RECEIVED        ACK for packet 9

======================================================================
Final attempts (sender transmissions): 16
Final ACKs received: 9
```

CONCLUSION:

I have successfully implemented and simulated the Go-Back-N ARQ protocol. Through this experiment, I understood how sliding window techniques improve channel utilization compared to Stop-and-Wait, and how retransmission is handled in case of lost or corrupted frames. I also learned how acknowledgments and timeouts play a crucial role in ensuring reliable data transmission.

Post Lab Questions

1. **Compare Go-Back-N and Stop and Wait.**

   Stop-and-Wait ARQ: In this protocol, the sender transmits one frame and waits for its acknowledgment (ACK) before sending the next. It is simple but inefficient for long-distance or high-latency networks, as the channel remains idle while waiting for ACK.

   Go-Back-N ARQ: The sender can transmit multiple frames (up to a window size) without waiting for ACK of each one. If an error is detected in a frame, all subsequent frames are retransmitted. This improves efficiency compared to Stop-and-Wait but increases retransmission overhead in case of errors.

   Key difference: Stop-and-Wait has a window size of 1, while Go-Back-N uses a larger sliding window for better utilization of the channel.

2. **What is Flow Control and why it is necessary?**

   Flow control is a technique used in data communication to ensure that the sender does not overwhelm the receiver by sending data faster than it can be processed or stored. It manages the rate of data transmission between sender and receiver. It is necessary because:

   - Receivers may have limited buffer capacity.
   - Prevents data loss due to buffer overflow.
   - Ensures efficient utilization of network resources.
   - Maintains reliable communication between devices with different processing speeds.

3. **The maximum window size for data transmission using the selective reject protocol with n-bit frame sequence numbers is**

   a) 2n b) 2n-1 c) 2n-1 d)2n-2

Date: 04 / 09 / 2025 **Signature of Faculty In-charge**