Computer Networks Lab Report

Assignment – 2

Problem Statement:
Design and implement flow control mechanisms of Logical Link Control of Data Link Layer within a simulated network environment.

Student Details

- **Name:** Sumesh Ranjan Majee
- **Roll Number:** 002310501020
- **Year:** 3rd Year B.C.S.E
- **Group:** A1

- **Date:** 12th September 2025

Subject:

Computer Networks

Design

1. Purpose of the Program

The purpose of this program is to **design and implement flow control mechanisms at the Data Link Layer** in a simulated network environment. Specifically, the program implements and compares three Automatic Repeat reQuest (ARQ) schemes:

- **Stop and Wait**
- **Go-Back-N ARQ**
- **Selective Repeat ARQ**

The program uses a **Sender** and **Receiver** connected via a simulated channel that can introduce **random delays** (causing timeouts and retransmissions). Each data frame includes addressing, sequence numbers, payload, and a Frame Check Sequence (FCS) from Assignment 1 (Checksum/CRC).

This lab demonstrates how different ARQ techniques handle **flow control, acknowledgements, and retransmissions** under delay conditions, and allows measuring their relative efficiency and reliability.

Structural Diagram of the Program

The program is organized into **8 files**:

- **6 Java files** – Core ARQ protocol implementation
- **1 Python file** – Analytics and performance visualization
- **1 Shell script** – Compilation and execution automation

File Structure

1. Sender.java

- Main entry point for the sender side
- Handles MAC address conversion from hex to binary format
- Creates frames using FrameBuilder and manages connection setup
- Provides menu-driven interface to select ARQ schemes:
  - Stop and Wait
  - Go-Back-N ARQ
  - Selective Repeat ARQ
- Contains shared constants: TIMEOUT_MS (5000ms), TOTAL_FRAMES (150), PROB (95%), window size N (7)

2. Receiver.java

- Server-side implementation that listens for incoming connections
- Implements receiver logic for all three ARQ schemes:
  - `stop_and_wait()` - Simple acknowledgment with ACK loss simulation
  - `go_back_arq()` - Cumulative ACK with duplicate ACK for out-of-order frames
  - `selective_repeat_arq()` - Individual frame ACKs with NAK support for missing frames
- Simulates ACK/NAK loss with 95% delivery probability
- Handles frame buffering and ordering for selective repeat

3. StopAndWait.java

- Implements Stop-and-Wait ARQ protocol
- Sends one frame at a time and waits for ACK
- Handles timeout and retransmission for lost frames or ACKs
- Measures round-trip time (RTT) for each frame
- Tracks frame transmission times with `frameTimes` and `sendTimeMap`

4. GoBackARQ.java

- Implements Go-Back-N ARQ protocol with sliding window
- Maintains window of unacknowledged frames (size N=2)
- Uses cumulative acknowledgments
- Retransmits entire window on timeout
- Tracks timing for all frames within the sliding window

5. SelectiveRepeatARQ.java

- Implements Selective Repeat ARQ protocol
- Individual acknowledgment for each frame
- Supports NAK (Negative Acknowledgment) for requesting specific lost frames
- Maintains sender-side buffer with sliding window mechanism
- Only retransmits specifically requested frames

6. FrameBuilder.java

- Constructs network frames from input file data
- Creates 60-byte frames (480 bits total) with structure:
  - Source MAC address (48 bits)
  - Destination MAC address (48 bits)
  - Length field (16 bits)
  - Payload data (368 bits from input file)
- Reads input file in 46-character chunks
- Calls error detection methods (references `Utils.getChecksum()` from Assignment 1)

7. analytics.py

- Performance analysis and visualization tool
- Reads CSV files generated by each ARQ scheme:
  - `csvframe_times_stop_and_wait{prob}.csv`
  - `csvframe_times_go_back_n{prob}.csv`
  - `csvframe_times_selective_repeat{prob}.csv`
- Generates comparative bar chart histogram showing average acknowledgment times
- Tests multiple transmission success probabilities (80%, 85%, 90%, 95%, 100%)
- Exports performance metrics to `performance_metrics.csv`
- Auto-scales y-axis based on timeout behavior

8. compile_and_run_ass2.sh

- Automates the complete compilation and execution workflow
- **Steps performed:**
  1. **User Input Collection:** Prompts for port, input file path, sender MAC, and receiver MAC (with defaults)
  2. **Java Compilation:** Compiles all Java files in Assignment2 directory
  3. **Receiver Startup:** Launches Receiver as background process on specified port
  4. **Sender Execution:** Runs Sender with user-provided parameters
  5. **Process Management:** Gracefully terminates receiver after completion
  6. **Analytics Generation:** Executes Python script to generate performance visualizations
- **Default Configuration:**
  - Port: 5000
  - Input file: `Assignments/Assignment2/inputfile.txt`
  - Sender MAC: `98-BA-5F-ED-66-B7`
  - Receiver MAC: `AA-BA-5F-ED-66-B7`

- **Python Integration:** Uses Anaconda Python path (`/c/Users/User/anaconda3/python.exe`)
- **Output:** Generates `detection_histogram.png` and `performance_metrics.csv`

Key Features

- **Socket-based communication** between sender and receiver
- **Configurable transmission loss simulation** (PROB parameter)
- **Timeout and retransmission handling** with 5-second timeouts
- **Performance measurement** tracking RTT and total transmission times
- **Comparative analysis** across different ARQ schemes and loss probabilities
- **Automated execution pipeline** with shell script orchestration

Dependencies

The implementation references utilities from Assignment 1:

- `Assignments.Assignment1.Utils.exportToCSV()` - CSV export functionality
- `Assignments.Assignment1.Utils.getChecksum()` - Checksum calculation (from error detection assignment)

This represents a comprehensive **ARQ (Automatic Repeat Request) protocol simulation** focusing on **flow control and reliability mechanisms** with automated testing and performance analysis capabilities.

```
$ ./compile_and_run_ass2.sh
Port [5000]:
Input file path [Assignments/Assignment2/inputfile.txt]:
Sender MAC [98-BA-5F-ED-66-B7]:
Receiver MAC [AA-BA-5F-ED-66-B7]:
=== Compiling Java sources ===
=== Starting Receiver on port 5000 ===
Receiver listening on port 5000
=== Starting Sender ===
Enter
1. Stop and Wait
2. Go-Back-N ARQ
3. Selective Repeat ARQ
0. To Exit
Enter choice :
```

Input / Output Specification

- **Input Format:**
  - A text file containing any txt data

```
ANTONIO.
I am th' unhappy subject of these quarrels.

PORTIA.
Sir, grieve not you. You are welcome notwithstanding.

BASSANIO.
Portia, forgive me this enforced wrong,
And in the hearing of these many friends
I swear to thee, even by thine own fair eyes,
Wherein I see myself—
```

  - Choice ch
  - 1. Stop and Wait
  - 2. Go-Back-N ARQ
  - 3. Selective Repeat ARQ
  - 0. To Exit

- **Output Format:**
  - For each transmitted frame depending on schema:
    - Displays whether **Sender Frame lost or sent successfully** and **Receiver Ack lost or sent successfully** detected an error.
  - Final analytics:
    - Histogram plots
    - `performance_metrics.csv` file

Stop and wait

```
Enter choice : 1
Sender : Connected to receiver at localhost:5000
Receiver : Client connected: /127.0.0.1
Sender : Successfully Sent frame 0

Sender : Timer Started ...

Receiver : Received frame 0
Receiver : ACK sent for frame 0
Sender : Frame 0 acknowledged. Time taken: 0 ms
```

Go Back ARQ

```
Enter choice : 2
Receiver : Client connected: /127.0.0.1
Sender : Connected to receiver at localhost:5000

Sender : Timer Started/Restarted ...

Sender : Successfully Sent frame 0
Sender : Successfully Sent frame 1
Receiver : Frame 0 received in order
Receiver : Sent ACK 0
Sender : Received cumulative ACK for frame 0
```

Selective Repeat ARQ

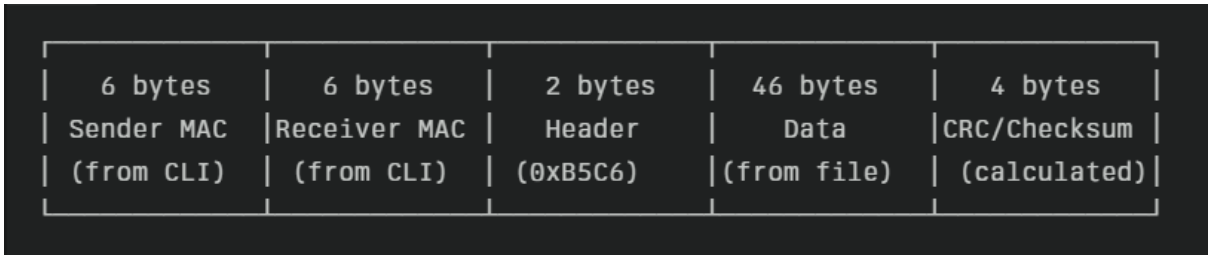
```
Enter choice : 3
Receiver : Client connected: /127.0.0.1
Sender : Connected to receiver at localhost:5000
Sender : Successfully Sent frame 0
Sender : Successfully Sent frame 1
Receiver : Frame 0 received and buffered

Sender : Timer Started/Restarted ...

Receiver : ACK sent for frame 0
```

Implementation

Sender Side

1. Frame Creation

1. The **input file** is read entirely and divided into **frames of 64 Bytes (512 bits)**.



2. The **payload (46 Bytes)** is extracted from the text file.
3. A function `createFrames()` is used to construct frames:
    - **a. Frame Construction**
        - The 46 Bytes is converted to binary 0 and 1
        - Each frame is built with:
            - Sender MAC address
            - Receiver MAC address
            - 2-byte header
            - Payload (46 Bytes)
        - If the payload is **less than 46 Bytes**, it is padded with **zeros**.
    - **b. Frame List Creation**
        - All frames are stored in a **list**.
    - **c. Error Detection Code Appending**
        - For each frame, **Checksum** values are calculated.
        - These values are appended to the frame, making it a complete **64 Byte frame**.
        - Finally, all frames are stored in a **List<String>** structure.
        - The list is returned to the **Sender**.

3. Frame Transmission

a. Stop-and-Wait ARQ

- **How it Works:**
  - The sender transmits **one frame at a time**.
  - After sending, a **timer starts** and the sender **waits for an ACK** from the receiver.
  - **If ACK arrives within timeout:**
    - The sender stops the timer and sends the **next frame**.
    - Timer restarts for the new frame.
  - **If no ACK within timeout:**
    - The timer expires, indicating possible loss or delay.
    - The sender **retransmits the same frame** and restarts the timer.
- **In the Program:**
  - Implemented in `StopAndWait.java` using a single send/receive loop.
  - A dedicated **Timer** class or method tracks the round-trip time.
  - The sender moves to the next frame only after a valid ACK is received.

---

b. Go-Back-N ARQ

- **How it Works:**
  - The sender transmits a **window of N frames** without waiting for individual ACKs.
  - It then waits for a **cumulative ACK** from the receiver.
  - **If cumulative ACK arrives within timeout:**
    - The sender slides the window forward and transmits new frames.
    - Timer restarts from the frame after the last acknowledged one.
  - **If timeout occurs:**
    - The sender retransmits **all frames starting from the last unacknowledged frame**.
- **In the Program:**
  - Implemented in `GoBackARQ.java` with a **sender window** and a **receiver window of size 1**.
  - Uses a loop to send up to N frames and a timer to monitor the oldest unacknowledged frame.
  - Cumulative acknowledgements shift the window forward.
  - On timeout, frames are resent in order starting at the lost frame.

---

c. Selective Repeat ARQ

- **How it Works:**
  - The sender transmits a **window of N frames** like Go-Back-N, but acknowledgements are **individual, not cumulative**.
  - The receiver buffers out-of-order frames and acknowledges them separately.
  - **If ACKs arrive within timeout:**

        - The sender only slides the window forward for acknowledged frames.
    - **If timeout occurs for specific frames:**
        - The sender **retransmits only those unacknowledged frames**.
    - **If NAK is received:**
        - The sender immediately retransmits the specific frame without waiting for a timeout.
- **In the Program:**
    - Implemented in `SelectiveRepeatARQ.java` with **equal sender and receiver window sizes**.
    - Maintains a buffer for out-of-order frames at the receiver.
    - Uses per-frame timers or a map of timers to track unacknowledged frames individually.
    - Selective retransmission minimizes bandwidth waste and improves performance under delay.

Protocol	Sending Pattern	ACK Handling	Timeout Handling	Retransmission Strategy
Stop-and-Wait	One frame at a time	Single ACK	Whole frame timeout	Retransmit same frame
Go-Back-N	Window of N frames	Cumulative ACK	Timer for oldest frame	Retransmit from last unacknowledged frame onward
Selective Repeat	Window of N frames	Individual ACK per frame	Per-frame timeout	Retransmit only unacknowledged/NAK'd frames

Receiver Side Implementation

The **Receiver** program complements the Sender by accepting incoming frames, checking for errors, and sending acknowledgements (ACK/NAK) back according to the ARQ protocol being used. It is implemented in `Receiver.java`.

---

1. Common Receive Process (All ARQ Schemes)

- The Receiver listens on a socket for incoming frames from the Sender.
- For each received frame:
    - **Extract header fields** (Source MAC, Destination MAC, Sequence Number, etc.).
    - **Extract payload** (46 bytes or more, depending on configuration).
    - **Check for errors** using Checksum/CRC (from Assignment 1).
    - If error-free → Accept the frame and deliver payload.
    - If error detected → Discard frame (or buffer for Selective Repeat if applicable).

---

2. Stop-and-Wait ARQ (Receiver)

- **Expected Sequence Number:** The Receiver keeps track of the next expected frame number.
- **On Correct Frame:**
    - If the frame sequence number matches the expected number and passes CRC/Checksum:
        - Deliver the payload to the upper layer.
        - Send an **ACK** back to the Sender for that frame.
        - Increment the expected sequence number.
- **On Error/Unexpected Frame:**
    - Discard the frame (no ACK sent).
    - Sender will timeout and retransmit.

---

3. Go-Back-N ARQ (Receiver)

- **Window Size = 1** (Receiver accepts only in-order frames).
- **On Correct Frame:**
    - If the frame sequence number matches the expected number and passes CRC/Checksum:
        - Deliver payload.
        - Send **cumulative ACK** indicating the last correctly received in-order frame.
        - Increment expected sequence number.
- **On Out-of-Order or Corrupted Frame:**
    - Discard the frame.

- **Repeat the ACK for the last correctly received frame** (cumulative ACK).
- Sender retransmits from that frame onwards after timeout.

---

4. Selective Repeat ARQ (Receiver)

- **Window Size = N** (Receiver accepts multiple frames, including out-of-order).
- Maintains a **buffer** for out-of-order frames.
- **On Correct Frame (within window):**
  - Store the frame in buffer if it is out of order.
  - If it matches the lowest unreceived frame number:
    - Deliver it to the upper layer.
    - Also deliver any consecutively buffered frames.
  - Send an **ACK** specifically for the received frame number.
- **On Corrupted Frame:**
  - Discard the frame.
  - Optionally send a **NAK** immediately to request retransmission (if NAK is enabled).
- This selective acknowledgement mechanism allows the Sender to retransmit **only the missing/errored frames**, reducing bandwidth waste.

Protocol	Receiver Window	Handling of Out-of-Order Frames	Acknowledgement Type	Error Handling
Stop-and-Wait	1	Discards all unexpected frames	Single ACK per frame	Silent discard → Sender timeout
Go-Back-N	1	Discards out-of-order frames	Cumulative ACK of last in-order frame	Silent discard → Sender retransmits from last ACK
Selective Repeat	N	Buffers out-of-order frames	Individual ACK per frame + optional NAK	Retransmit only missing frames

6. Integration with Sender

- The Receiver continuously runs its `Recv()` method to accept frames.
- For each valid frame, it calls `Check()` to verify CRC/Checksum.
- Depending on the ARQ scheme selected, it calls `Send()` to transmit either ACKs, cumulative ACKs, or NAKs.
- When all frames are processed, it closes the socket connection.

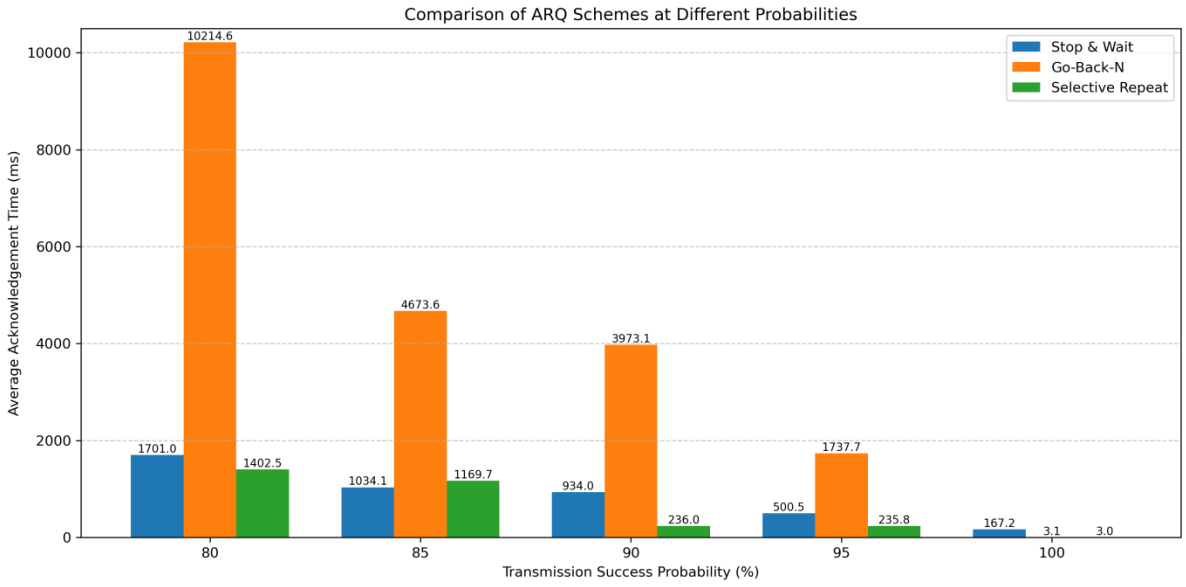
Test Cases

Since only **delay (no bit errors)** is implemented, the test cases focus on how each protocol behaves under different delay conditions:

Test Case	Description	What is Checked	Expected Outcome
1. No Delay (Ideal Channel)	Transmit frames with no artificial delay	Baseline throughput and correctness	All frames received and acknowledged with no retransmissions. Stop-and-Wait is slowest but no loss.
2. Small Random Delay (Low probability)	Introduce small random delays (probability 0.05)	Timeout handling and efficiency	Few or no timeouts. All three schemes deliver successfully; Go-Back-N and Selective Repeat show better utilization than Stop-and-Wait.
3. Medium Delay (probability 0.1)	Moderate random delays between sender and receiver	Sliding window efficiency, retransmission rate	Stop-and-Wait begins to show larger idle times. Go-Back-N retransmits windows after timeout. Selective Repeat retransmits only delayed frames.
4. High Delay (probability 0.15)	High random delay simulating a congested channel	Stress test of timeout/retransmission logic	High retransmission rates. Selective Repeat expected to perform best since it only retransmits affected frames.
5. Compare Average Round Trip Time (RTT)	Measure time between sending a frame and receiving its ACK under different delay settings	Accuracy of timer and timeout calculations	Proper RTT computation, dynamic adjustment of timeout interval.

3. Results

- **No Delay:** All three schemes delivered frames successfully with no retransmissions. Stop-and-Wait showed the lowest throughput due to idle waiting time; Go-Back-N and Selective Repeat achieved higher throughput.
- **Low Delay (p=0.05):** Very few timeouts occurred. Go-Back-N and Selective Repeat retained high efficiency.
- **Medium Delay (p=0.1):** Stop-and-Wait experienced noticeable idle times; Go-Back-N retransmitted windows on timeout; Selective Repeat retransmitted only delayed frames, thus saving bandwidth.
- **High Delay (p=0.15):** Stop-and-Wait throughput dropped significantly; Go-Back-N wasted bandwidth due to window retransmissions; Selective Repeat handled delays more gracefully and achieved the highest throughput.
- **RTT Measurement:** The timer and timeout mechanism correctly computed the round-trip times, and timeout thresholds adjusted accordingly.



	Probability(%)	StopAndWait_avgAck(ms)	GoBackN_avgAck(ms)	SelectiveRepeat_avgAck(ms)
1	80	1700.9533333333334	10214.64	1402.52
2	85	1034.0533333333333	4673.64	1169.66
3	90	934.0066666666667	3973.08	235.98666666666668
4	95	500.48	1737.6533333333334	235.84666666666666
5	100	167.21333333333334	3.1	3.006666666666667

4. Analysis

- **Stop-and-Wait ARQ:** Simple to implement but inefficient under delay because only one frame is outstanding at a time; sender idle during RTT.
- **Go-Back-N ARQ:** Better utilization since multiple frames are in flight. However, if one frame is delayed or lost, the entire window must be retransmitted.
- **Selective Repeat ARQ:** Most efficient under delay because only the affected frames are retransmitted. Requires more complex receiver buffering and independent acknowledgements but reduces wasted bandwidth.
- **Overall Trend:** As delay increases, protocols with **larger window sizes and selective retransmission (Selective Repeat)** outperform simpler schemes (Stop-and-Wait).

5. Comments

This lab provided practical insight into **flow control and error recovery mechanisms**. It demonstrated how different ARQ schemes behave under delayed acknowledgements and how timeouts drive retransmissions.

- **Learning Outcome:**
  - Gained hands-on experience with timers, sequence numbers, acknowledgements, and window-based transmission.
  - Learned how delays affect throughput and reliability.
- **Difficulty:**
  - Moderate. Understanding timers and implementing sliding windows required careful logic but was manageable.
- **Improvements Suggested:**
  - Add **bit error simulation** from Assignment 1 for a more realistic test.
  - Provide a **graphical visualization of window sliding** to make debugging easier.
  - Include a **configurable timeout calculation** (adaptive algorithms).

6. Conclusion

In this assignment, three flow control mechanisms—Stop-and-Wait, Go-Back-N ARQ, and Selective Repeat ARQ—were implemented and compared under different delay conditions.

Results showed that while **Stop-and-Wait** is simplest, it becomes inefficient as delays grow. **Go-Back-N ARQ** improves throughput but wastes bandwidth during retransmissions of entire windows. **Selective Repeat ARQ** delivers the best performance under delay since it retransmits only the frames actually delayed or lost.

This lab highlights the importance of **sliding windows and selective retransmission** in modern networking protocols, demonstrating how more complex flow control mechanisms improve network efficiency and reliability in real-world conditions.