Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Spring, Year:2025), B.Sc. in CSE (Day)

Lab Report NO # 04
Course Title: Computer Networking Lab
Course Code: CSE-312 Section:223-D2

Lab Experiment Name: Implementation of Flow Control and Congestion Control in TCP.

Student Details

Name		ID
1.	Hridoy Mia	223902010

Submission Date : 17/05/2025
Course Teacher's Name : Md. Sabbir Hosen Mamun

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

Title: Implementation of Flow Control and Congestion Control in TCP.

Objectives:

The objectives of the lab report Are:

- Understand TCP **flow control** using the receive window (`rwnd`).
- Implement **congestion control** using the congestion window (`cwnd`).
- Simulate **slow start**, **congestion avoidance**, and **congestion detection**.
- Demonstrate interaction between **rwnd** and **cwnd**.
- Observe **AIMD** behavior in TCP congestion control.

Procedure:

1. **Create `Receiver.java`:**
   Simulate a TCP receiver with a fixed-size buffer. Send ACKs with updated `rwnd` values to the sender.
2. **Create `Sender.java`:**
   Implement TCP sender logic using `cwnd`, `ssthresh`, and `rwnd` to control data transmission. Simulate slow start and congestion avoidance.
3. **Add Delay and Congestion Simulation:**
   Use `Thread.sleep()` to mimic network delay. Trigger congestion events to test `cwnd` reduction.
4. **Run the Simulation:**
   Start the receiver first, then the sender. Observe how data is sent and controlled based on `rwnd` and `cwnd`.
5. **Monitor Output:**
   Track how flow control (`rwnd`) and congestion control (`cwnd`) work together to manage reliable data transfer.

Implementation:

Step 1: `Receiver.java`

```
import java.io.*;

import java.net.*;


public class Receiver {

    private static final int PORT = 5000;

    private static final int BUFFER_SIZE = 50;


    public static void main(String[] args) {

        try (ServerSocket serverSocket = new ServerSocket(PORT)) {

            System.out.println("Receiver started on port " + PORT + ". Waiting for sender...");


            try (Socket socket = serverSocket.accept();

                 BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

                 PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {


                System.out.println("Sender connected.");

                int currentBuffer = 0;


                String message;

                while ((message = in.readLine()) != null) {
```

```
                if (message.startsWith("DATA")) {
                    System.out.println("Received: " + message);
                    currentBuffer++;

                    if (currentBuffer >= BUFFER_SIZE) {
                        System.out.println("Buffer full. Sending rwnd=0 to sender.");
                        out.println("ACK rwnd=0");
                        Thread.sleep(2000);
                        currentBuffer = 0;
                        out.println("ACK rwnd=" + (BUFFER_SIZE - currentBuffer));
                    } else {
                        out.println("ACK rwnd=" + (BUFFER_SIZE - currentBuffer));
                    }
                }
            }
        }

    } catch (IOException | InterruptedException e) {
    }
  }
}
```

Step 2: `Sender.java`

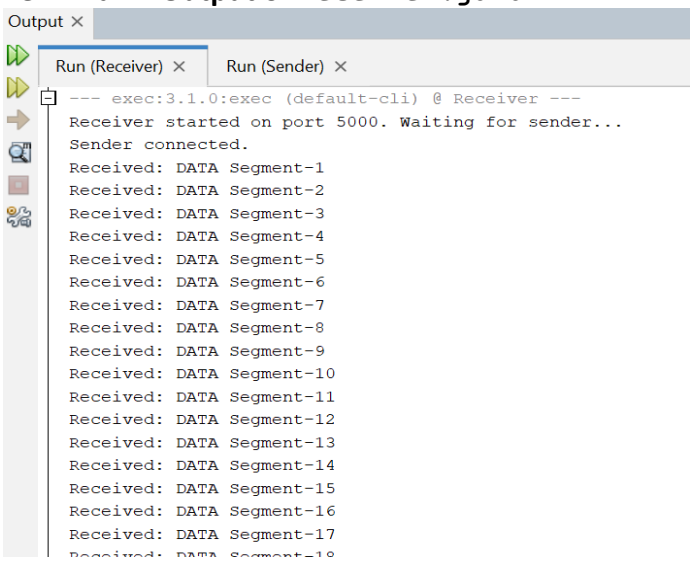
```
import java.io.*;
import java.net.*;
import java.util.Random;
public class Sender {
    private static final String HOST = "localhost";
    private static final int PORT = 5000;

    private static final int MAX_SEGMENTS = 100;
    private static int cwnd = 1;
    private static int ssthresh = 16;
    private static int rwnd = 50;

    public static void main(String[] args) {
        try (Socket socket = new Socket(HOST, PORT);
             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {

            System.out.println("Connected to receiver.");
            int segment = 1;
            Random rand = new Random();

            while (segment <= MAX_SEGMENTS) {
                int windowSize = Math.min(cwnd, rwnd);
```

```
        System.out.println("\n[INFO] cwnd=" + cwnd + ", rwnd=" + rwnd + ", sending " + windowSize
+ " segments");


        for (int i = 0; i < windowSize && segment <= MAX_SEGMENTS; i++) {
            String data = "DATA Segment-" + segment;
            out.println(data);
            System.out.println("Sent: " + data);
            segment++;
        }


        String response = in.readLine();
        if (response != null && response.startsWith("ACK")) {
            try {
                rwnd = Integer.parseInt(response.split("=")[1].trim());
            } catch (NumberFormatException e) {
                rwnd = 50;
                System.out.println("Failed to parse rwnd, using default 50");
            }


            if (rand.nextInt(100) < 10) {
                System.out.println("[WARNING] Congestion detected (simulated). Reducing cwnd.");
                ssthresh = Math.max(cwnd / 2, 1);
                cwnd = 1;
                continue;
            }
```

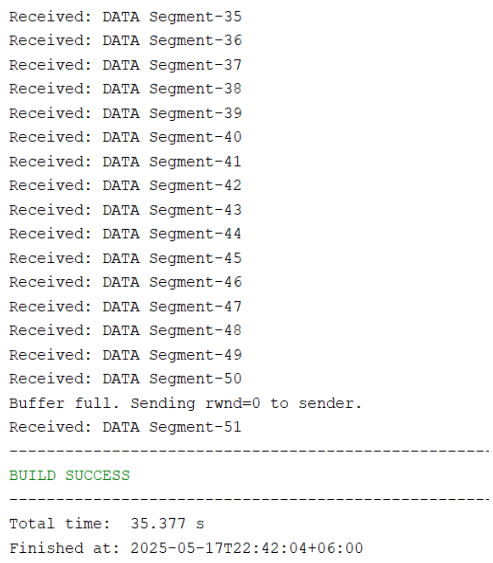
```
                if (cwnd < ssthresh) {
                    cwnd *= 2;
                    System.out.println("Slow Start: cwnd increased to " + cwnd);
                } else {
                    cwnd += 1;
                    System.out.println("Congestion Avoidance: cwnd increased to " + cwnd);
                }
            } else {
                System.out.println("No ACK received. Retrying...");
                Thread.sleep(1000);
            }

            Thread.sleep(500);
        }

        System.out.println("\n[INFO] Transmission complete.");

    } catch (IOException | InterruptedException e) {
    }
  }
}
```

Output:

Terminal 1: Output of `Receiver.java`

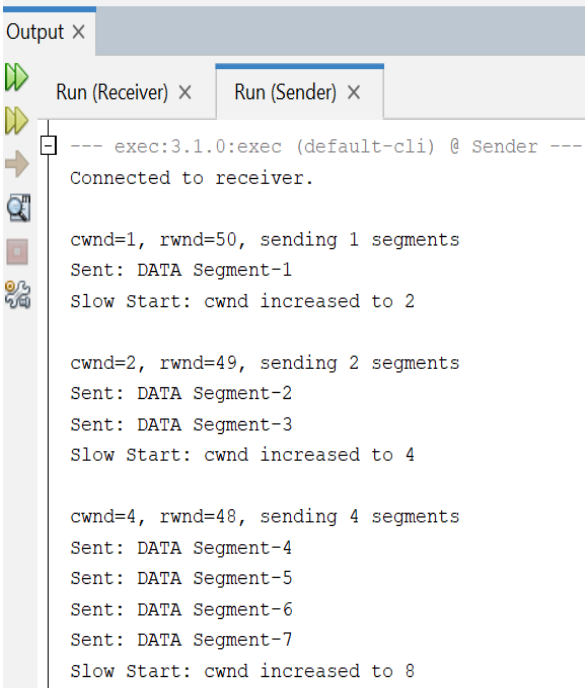


```
Output ×
Run (Receiver) ×   Run (Sender) ×
--- exec:3.1.0:exec (default-cli) @ Receiver ---
Receiver started on port 5000. Waiting for sender...
Sender connected.
Received: DATA Segment-1
Received: DATA Segment-2
Received: DATA Segment-3
Received: DATA Segment-4
Received: DATA Segment-5
Received: DATA Segment-6
Received: DATA Segment-7
Received: DATA Segment-8
Received: DATA Segment-9
Received: DATA Segment-10
Received: DATA Segment-11
Received: DATA Segment-12
Received: DATA Segment-13
Received: DATA Segment-14
Received: DATA Segment-15
Received: DATA Segment-16
Received: DATA Segment-17
```

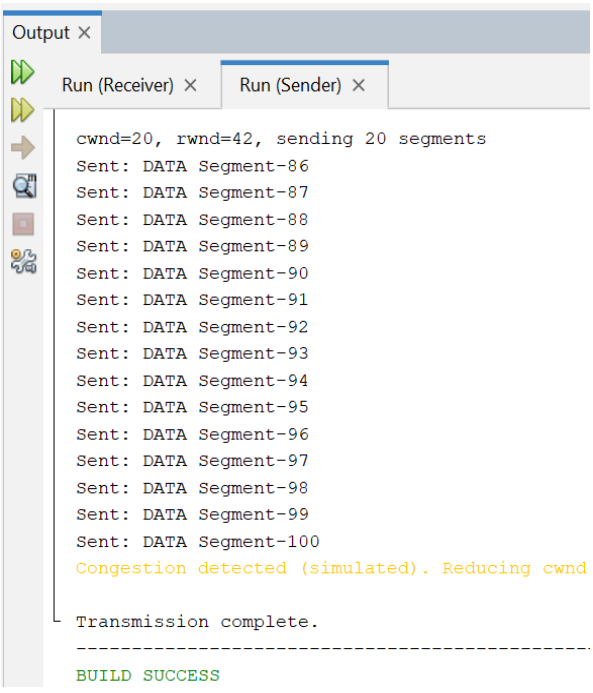


```
Received: DATA Segment-35
Received: DATA Segment-36
Received: DATA Segment-37
Received: DATA Segment-38
Received: DATA Segment-39
Received: DATA Segment-40
Received: DATA Segment-41
Received: DATA Segment-42
Received: DATA Segment-43
Received: DATA Segment-44
Received: DATA Segment-45
Received: DATA Segment-46
Received: DATA Segment-47
Received: DATA Segment-48
Received: DATA Segment-49
Received: DATA Segment-50
Buffer full. Sending rwnd=0 to sender.
Received: DATA Segment-51
------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------
Total time:  35.377 s
Finished at: 2025-05-17T22:42:04+06:00
```

Terminal 2: Output of `Sender.java`



```
Output ×
Run (Receiver) ×   Run (Sender) ×
--- exec:3.1.0:exec (default-cli) @ Sender ---
Connected to receiver.

cwnd=1, rwnd=50, sending 1 segments
Sent: DATA Segment-1
Slow Start: cwnd increased to 2

cwnd=2, rwnd=49, sending 2 segments
Sent: DATA Segment-2
Sent: DATA Segment-3
Slow Start: cwnd increased to 4

cwnd=4, rwnd=48, sending 4 segments
Sent: DATA Segment-4
Sent: DATA Segment-5
Sent: DATA Segment-6
Sent: DATA Segment-7
Slow Start: cwnd increased to 8
```



```
Output ×
Run (Receiver) ×   Run (Sender) ×
cwnd=20, rwnd=42, sending 20 segments
Sent: DATA Segment-86
Sent: DATA Segment-87
Sent: DATA Segment-88
Sent: DATA Segment-89
Sent: DATA Segment-90
Sent: DATA Segment-91
Sent: DATA Segment-92
Sent: DATA Segment-93
Sent: DATA Segment-94
Sent: DATA Segment-95
Sent: DATA Segment-96
Sent: DATA Segment-97
Sent: DATA Segment-98
Sent: DATA Segment-99
Sent: DATA Segment-100
Congestion detected (simulated). Reducing cwnd

Transmission complete.
------------------------------------------------
BUILD SUCCESS
```

Discussion and Conclusion:

This lab demonstrated how TCP's flow control (using rwnd) prevents the sender from overwhelming the receiver, while congestion control (using cwnd) adjusts the sending rate to avoid network congestion. Our Java simulation showed that both mechanisms work together to ensure efficient and reliable data transfer. The slow start and congestion avoidance phases effectively manage transmission rates, balancing throughput and network stability. Without congestion control, network performance would degrade despite flow control. This exercise reinforced the importance of these complementary TCP features in maintaining smooth and reliable communication.

Concise Answers:

- **What is the purpose of the receive window (rwnd) in TCP?**

To inform the sender of the available buffer space at the receiver, preventing buffer overflow.

- **What happens when the rwnd value reaches zero and is advertised to the sender?**

The sender must stop sending new data to avoid overwhelming the receiver.

- **How does the sender resume transmission after receiving an rwnd value of zero?**

It waits until the receiver advertises a non-zero rwnd, indicating buffer space is available again.

- **What is the role of the sliding window protocol in TCP flow control?**

It allows efficient, continuous data flow by sliding the window forward as ACKs arrive, controlling how much data can be sent without overflow.

- **Explain the equation: LastByteSent − LastByteAcked ≤ rwnd.**

The amount of unacknowledged data in transit must not exceed the receive window to avoid overwhelming the receiver.

- **What is Silly Window Syndrome (SWS), and how can it be avoided?**

SWS occurs when small amounts of data are sent frequently, causing inefficiency. It is avoided by delaying sending small segments or by increasing buffer thresholds before advertising window updates.

- **What is the role of the congestion window (cwnd) in TCP?**

To limit the amount of data sent into the network based on network congestion conditions.

- **How does TCP differentiate between flow control and congestion control?**

Flow control is receiver-driven (rwnd), preventing buffer overflow; congestion control is sender-driven (cwnd), avoiding network congestion.

- **What is the significance of the slow start threshold (ssthresh)?**

It marks the transition point between exponential (slow start) and linear (congestion avoidance) growth of cwnd.

- **What are the three phases of TCP congestion control, and how does cwnd behave in each?**

  - **Slow Start:** cwnd increases exponentially.
  - **Congestion Avoidance:** cwnd increases linearly.
  - **Fast Recovery:** cwnd temporarily reduces and then grows linearly.

- **Compare the behavior of TCP Tahoe and TCP Reno upon detecting congestion.**

  - Tahoe resets cwnd to 1 and enters slow start on loss detection.
  - Reno performs fast retransmit and fast recovery, reducing cwnd less aggressively.

- **How does TCP detect network congestion (i.e., timeout vs. triple duplicate ACKs)?**

  - Timeout indicates packet loss (severe congestion).
  - Triple duplicate ACKs indicate likely packet loss but less severe congestion.

- **What is AIMD (Additive Increase, Multiplicative Decrease) and how does it relate to TCP congestion control?**

AIMD controls cwnd: increase cwnd additively (linearly) when no congestion and decrease multiplicatively (halving) upon congestion detection.

- **What is the purpose of the Fast Recovery phase in TCP Reno, and why is it not used in Tahoe?**

Fast Recovery avoids slow start after packet loss by maintaining higher throughput; Tahoe does not implement this and drops to slow start.

- **What would happen if TCP ignored congestion control and only used flow control?**

Network congestion would likely lead to packet loss, retransmissions, and degraded network performance, as flow control only prevents receiver overflow but not network overload.