Department of Computer Science and Engineering

University of Dhaka

Implementation of TCP Congestion Control Mechanism: TCP Tahoe and TCP Reno and Their Performance Analysis

Submitted By:

Sara Faria Sundra (58)
Asuma Bhuiyan (85)

Submitted To:

Dr. Sabbir Ahmed
Dr. Ismat Rahman
Mr. Palash Roy (PR)

Contents

Chapter 1

Introduction

Transmission Control Protocol (TCP) is one of the most widely used transport layer protocols designed to ensure reliable, ordered, and congestion-aware data transfer across the internet. As network traffic grows, avoiding congestion becomes essential to maintain stability and throughput.

TCP Tahoe and TCP Reno are two foundational congestion control algorithms. Both use mechanisms such as Slow Start, Congestion Avoidance, and packet loss detection, but they differ in how they react to packet loss events.

This report presents the implementation and evaluation of TCP Tahoe and TCP Reno by simulating file transfers, triggering controlled packet losses, and analyzing performance.

Chapter 2

Objectives

1. To implement the core congestion control algorithms—TCP Tahoe and TCP Reno.
2. To visualize and compare changes in congestion window (cwnd) over time.
3. To measure and evaluate key performance metrics such as:
   - Throughput
   - Round-trip time (RTT)
   - Packet loss rate
   - Congestion window growth
4. To identify how congestion control behavior affects overall network performance.
5. To compare real-time logs from client and server sides.

Chapter 3

Design Details

The experiment involves the following sequential workflow:

1. Development of client and server applications using socket programming.
2. Establishment of stable TCP connections before data transmission.
3. Simulated packet loss scenarios using software-based delays and dropped acknowledgments.
4. Implementation of congestion control states:
    - **TCP Tahoe**: Slow Start, Congestion Avoidance, cwnd reset on loss.
    - **TCP Reno**: Slow Start, Fast Retransmit, Fast Recovery.
5. Detailed event logging including:
    - cwnd updates
    - Sequence transmission
    - Re-transmissions
    - RTT measurements
    - Loss detection

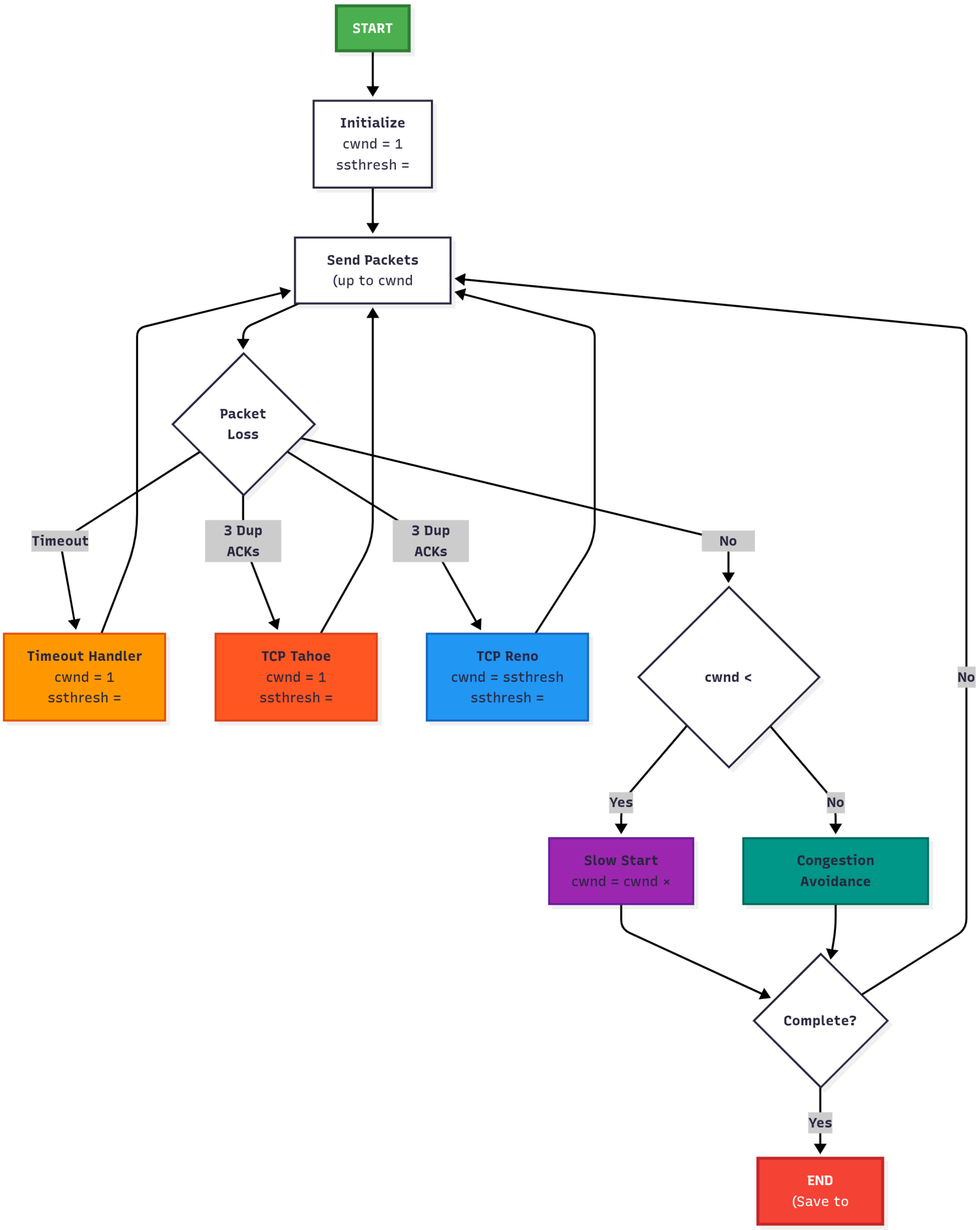


Figure 3.1: Overall Workflow of TCP Tahoe & TCP Reno Implementation

Chapter 4

Result Analysis

4.1 Source Code Used in Experiment

Client.java

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Client {
    static List<DataPoint> tahoeData = new ArrayList<>();
    static List<DataPoint> renoData = new ArrayList<>();

    static class DataPoint {
        int round;
        int cwnd;
        int ssthresh;

        DataPoint(int round, int cwnd, int ssthresh) {
            this.round = round;
            this.cwnd = cwnd;
            this.ssthresh = ssthresh;
        }
    }

    public static void main(String[] args) {
        String host = "127.0.0.1";
        int port = 5000;
        int timeoutMs = 2000;

        try (Socket socket = new Socket(host, port)) {
            socket.setSoTimeout(timeoutMs);
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getIn

            System.out.println("Connected to Server... Waiting for READY signal...");
```

```
String ready = in.readLine();
if (!"READY".equalsIgnoreCase(ready)) {
    System.out.println("Server not ready. Exiting...");
    return;
}

Scanner sc = new Scanner(System.in);
System.out.print("Enter TCP mode (TAHOE / RENO): ");
String mode = sc.nextLine().trim().toUpperCase();
sc.close();
if (!mode.equals("TAHOE") && !mode.equals("RENO")) {
    System.out.println("Invalid mode! Defaulting to TAHOE.");
    mode = "TAHOE";
}
System.out.println("== TCP " + mode + " Mode ==\n");

int cwnd = 1;
int ssthresh = 8;
int dupACKcount = 0;
String lastACK = "";
int nextPacketNum = 1;
int round = 1;
int totalRounds = 15;
Queue<String> packetsToRetransmit = new LinkedList<>();
boolean inFastRecovery = false;

Map<String, Long> sentPackets = new HashMap<>();


while (round <= totalRounds) {
    System.out.println("Round " + round + ": cwnd = " + cwnd + ", ssthres

    if (mode.equals("TAHOE")) {
        tahoeData.add(new DataPoint(round, cwnd, ssthresh));
    } else {
        renoData.add(new DataPoint(round, cwnd, ssthresh));
    }

    List<String> packetsToSend = new ArrayList<>();

    while (!packetsToRetransmit.isEmpty() && packetsToSend.size() < cwnd)
        packetsToSend.add(packetsToRetransmit.poll());
    }

    while (packetsToSend.size() < cwnd && nextPacketNum <= totalRounds *
        packetsToSend.add("pkt" + nextPacketNum);
        nextPacketNum++;
```

```
}

if (packetsToSend.isEmpty()) {
    break;
}

String packetMessage = String.join(",", packetsToSend);
long sendTime = System.currentTimeMillis();
out.println(packetMessage);
System.out.println("Sent packets: " + String.join(", ", packetsToSend

for (String pkt : packetsToSend) {
    sentPackets.put(pkt, sendTime);
}

dupACKcount = 0;
lastACK = "";
boolean fastRetransmitTriggered = false;
boolean timeoutTriggered = false;
Set<String> ackedPackets = new HashSet<>();

int maxACKs = packetsToSend.size() + 2;
int acksReceived = 0;
long roundStartTime = System.currentTimeMillis();

while (acksReceived < maxACKs) {
    String ack = null;
    try {
        ack = in.readLine();
        if (ack == null) {
            break;
        }
    } catch (SocketTimeoutException e) {

        System.out.println("==> Timeout detected: No ACK received wit
        timeoutTriggered = true;


        ssthresh = Math.max(cwnd / 2, 1);


        cwnd = 1;
        inFastRecovery = false;
        System.out.println("Timeout: cwnd -> 1, ssthresh -> " + ssthr

        for (String pkt : packetsToSend) {
            if (!ackedPackets.contains(pkt) && !packetsToRetransmit.c
                packetsToRetransmit.offer(pkt);
        }
    }


    if (mode.equals("TAHOE")) {
        tahoeData.add(new DataPoint(round, cwnd, ssthresh));
    } else {
        renoData.add(new DataPoint(round, cwnd, ssthresh));
    }
    break;
}

if (ack == null) {
    break;
}

acksReceived++;
System.out.println("Received: " + ack);
System.out.flush();

if (ack.startsWith("ACK:")) {
    String ackPkt = ack.substring(4);


    if (!ackPkt.equals("NA") && !ackedPackets.contains(ackPkt)) {

        for (String pkt : packetsToSend) {
            if (ackPkt.equals(pkt)) {
                ackedPackets.add(ackPkt);
                sentPackets.remove(pkt);
                break;
            }
        }
    }


    if (!lastACK.isEmpty() && lastACK.equals(ack)) {
        dupACKcount++;

        if (dupACKcount == 3 && !fastRetransmitTriggered && !time

            System.out.println("==> 3 Duplicate ACKs: Fast Retran
            System.out.flush();
            fastRetransmitTriggered = true;


            String lostPacket = null;
```

```
if (ackPkt.equals("NA") || ackPkt.equals("pkt0")) {

    lostPacket = packetsToSend.get(0);
} else {

    int ackNum = extractPacketNumber(ackPkt);
    lostPacket = "pkt" + (ackNum + 1);


    boolean found = false;
    for (String pkt : packetsToSend) {
        if (pkt.equals(lostPacket)) {
            found = true;
            break;
        }
    }

    if (!found) {
        for (String pkt : packetsToSend) {
            int pktNum = extractPacketNumber(pkt);
            if (pktNum > ackNum) {
                lostPacket = pkt;
                break;
            }
        }
    }
}


if (lostPacket != null) {
    if (mode.equals("TAHOE")) {

        boolean addRemaining = false;
        for (String pkt : packetsToSend) {
            if (pkt.equals(lostPacket)) {
                addRemaining = true;
            }
            if (addRemaining && !packetsToRetransmit.
                packetsToRetransmit.offer(pkt);
            }
        }
    } else {

        boolean addRemaining = false;
        for (String pkt : packetsToSend) {
            if (pkt.equals(lostPacket)) {
                addRemaining = true;
            }
```

```
                        if (addRemaining && !packetsToRetransmit.
                            packetsToRetransmit.offer(pkt);
                        }
                    }
                }
            }


            ssthresh = Math.max(cwnd / 2, 1);

            if (mode.equals("RENO")) {

                cwnd = ssthresh;
                inFastRecovery = true;
                System.out.println(
                        "TCP RENO Fast Recovery: cwnd -> " + cwnd

                renoData.add(new DataPoint(round, cwnd, ssthresh)
            } else {

                cwnd = 1;
                inFastRecovery = false;
                System.out.println("TCP TAHOE Reset: cwnd -> 1");

                tahoeData.add(new DataPoint(round, cwnd, ssthresh
            }


            dupACKcount = 0;
            lastACK = "";


        }
    } else {

        dupACKcount = 1;
        lastACK = ack;
    }
}


if (fastRetransmitTriggered) {
    break;
}


if (!fastRetransmitTriggered && ackedPackets.size() == packetsToS
                        break;
                    }
                }


                if (timeoutTriggered) {

                } else if (mode.equals("RENO") && inFastRecovery && !fastRetransmitTr

                    cwnd = cwnd + 1;
                    System.out.println("Congestion Avoidance (Fast Recovery): cwnd ->

                    inFastRecovery = false;
                } else if (!fastRetransmitTriggered) {

                    if (cwnd < ssthresh) {

                        cwnd = cwnd * 2;
                        System.out.println("Slow Start: cwnd -> " + cwnd);
                    } else {

                        cwnd = cwnd + 1;
                        System.out.println("Congestion Avoidance: cwnd -> " + cwnd);
                    }
                } else {

                }


                round++;
                System.out.println();
                Thread.sleep(500);
            }


            System.out.println("\nTransmission complete. Disconnecting...");


            saveDataToFile(mode);

            socket.close();

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
    private static void saveDataToFile(String mode) {
        try {
            String filename = mode.toLowerCase() + "_data.csv";
            PrintWriter writer = new PrintWriter(new FileWriter(filename));
            writer.println("Round,cwnd,ssthresh");

            List<DataPoint> data = mode.equals("TAHOE") ? tahoeData : renoData;
            for (DataPoint point : data) {
                writer.println(point.round + "," + point.cwnd + "," + point.ssthresh)
            }

            writer.close();
            System.out.println("Data saved to " + filename);
        } catch (IOException e) {
            System.err.println("Error saving data: " + e.getMessage());
        }
    }

    private static int extractPacketNumber(String packet) {
        try {
            if (packet.startsWith("pkt")) {
                return Integer.parseInt(packet.substring(3));
            }
        } catch (Exception e) {
            return -1;
        }
        return -1;
    }
}
```

Server.java

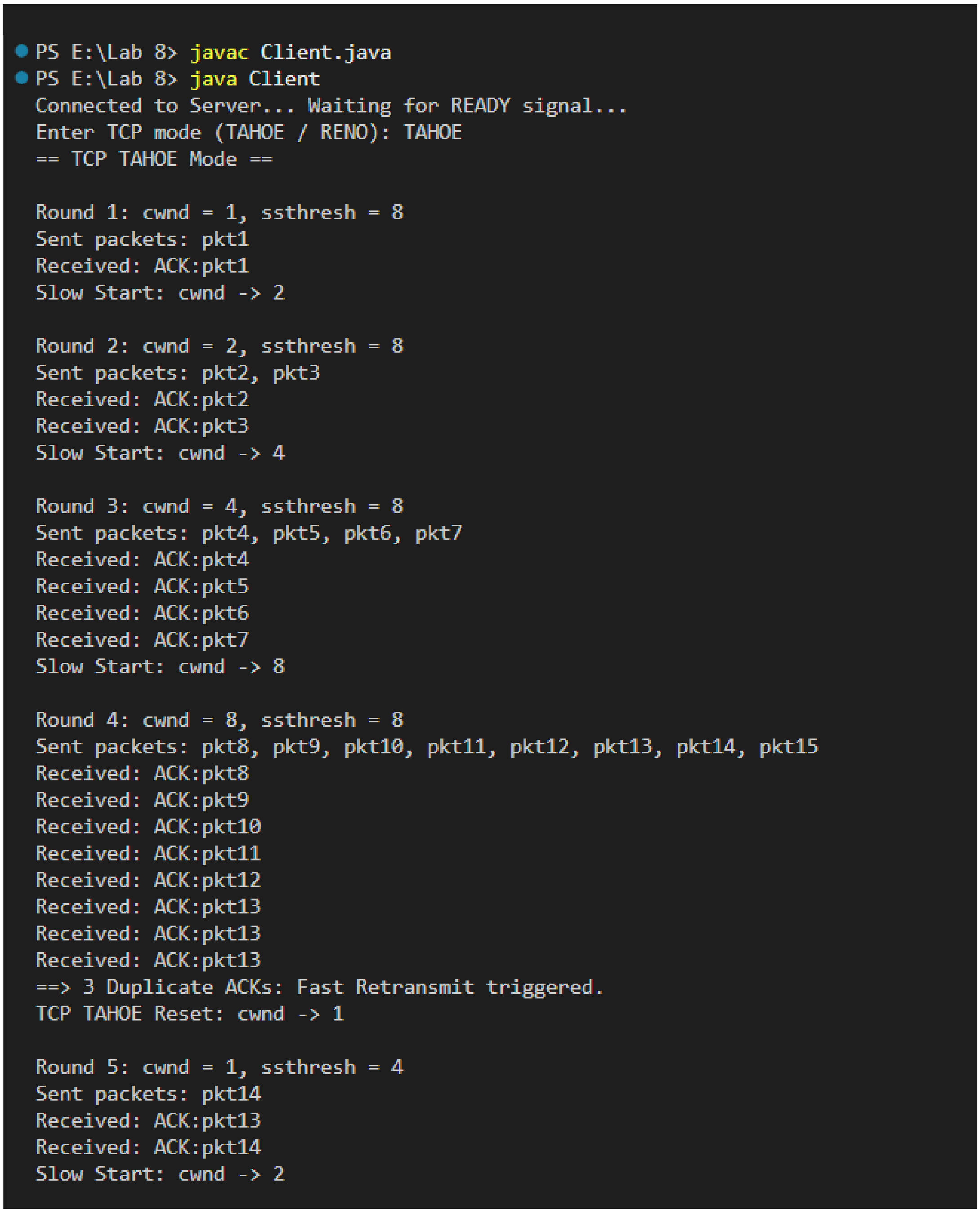
```
import java.io.*;
import java.net.*;
import java.util.*;

public class Server {
    public static void main(String[] args) {
        int port = 5000;
        System.out.println("Server started on port " + port);

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

            Socket socket = serverSocket.accept();
            System.out.println("Client connected: " + socket.getInetAddress());

            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getIn
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            out.println("READY");

            Random rand = new Random();
            String lastSuccessfullyReceivedPacketAcrossRounds = "";
            int roundCount = 0;


            while (true) {

                String message = in.readLine();
                if (message == null || message.isEmpty()) {
                    break;
                }

                roundCount++;


                String[] packets = message.split(",");
                List<String> packetList = new ArrayList<>();
                for (String pkt : packets) {
                    packetList.add(pkt.trim());
                }

                if (packetList.isEmpty()) {
                    continue;
                }


                boolean simulateTimeout = false;
                int lossIndex = -1;


                boolean canSimulateCongestion = false;
                for (String pkt : packetList) {
                    int pktNum = extractPacketNumber(pkt);
                    if (pktNum >= 11) {
                        canSimulateCongestion = true;
                        break;
                    }
                }

                if (canSimulateCongestion && roundCount > 3) {

                    int randomValue = rand.nextInt(100);

                    if (randomValue < 5) {
```

```
        simulateTimeout = true;
        System.out.println("Simulating timeout: Not sending ACKs for
                          String.join(", ", packetList) + ")");

        continue;
    } else if (randomValue < 25) {

        List<Integer> eligibleIndices = new ArrayList<>();
        for (int i = 0; i < packetList.size(); i++) {
            int pktNum = extractPacketNumber(packetList.get(i));
            if (pktNum >= 11) {
                eligibleIndices.add(i);
            }
        }
        if (!eligibleIndices.isEmpty()) {
            lossIndex = eligibleIndices.get(rand.nextInt(eligibleIndi
            System.out.println("Simulating packet loss: Packet " + pa
                              " will be lost (will trigger duplicate
        }
    }

}


String lastSuccessfullyReceivedPacket = lastSuccessfullyReceivedPacke

for (int i = 0; i < packetList.size(); i++) {
    String packet = packetList.get(i);

    if (i == lossIndex) {

        if (i == 0 && lastSuccessfullyReceivedPacket.isEmpty()) {

            String ackToSend = "ACK:NA";
            System.out.println("Packet lost: " + packet + " - Sending
            out.println(ackToSend);
            out.println(ackToSend);
            out.println(ackToSend);
        } else {

            String ackToSend = lastSuccessfullyReceivedPacket.isEmpty
                    : "ACK:" + lastSuccessfullyReceivedPacket;
            System.out.println("Packet lost: " + packet + " - Sending
                    + lastSuccessfullyReceivedPacket
                    + " (3 times)");
            out.println(ackToSend);
            out.println(ackToSend);
```

```
                            out.println(ackToSend);
                        }

                        break;
                    } else {

                        System.out.println("Received: " + packet + " - Sending ACK:" 
                        out.println("ACK:" + packet);
                        lastSuccessfullyReceivedPacket = packet;
                        lastSuccessfullyReceivedPacketAcrossRounds = packet;
                    }
                }


                Thread.sleep(100);
            }


            System.out.println("\nClient disconnected.");
            socket.close();

        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static int extractPacketNumber(String packet) {
        try {
            if (packet.startsWith("pkt")) {
                return Integer.parseInt(packet.substring(3));
            }
        } catch (Exception e) {
            return -1;
        }
        return -1;
    }
}
```

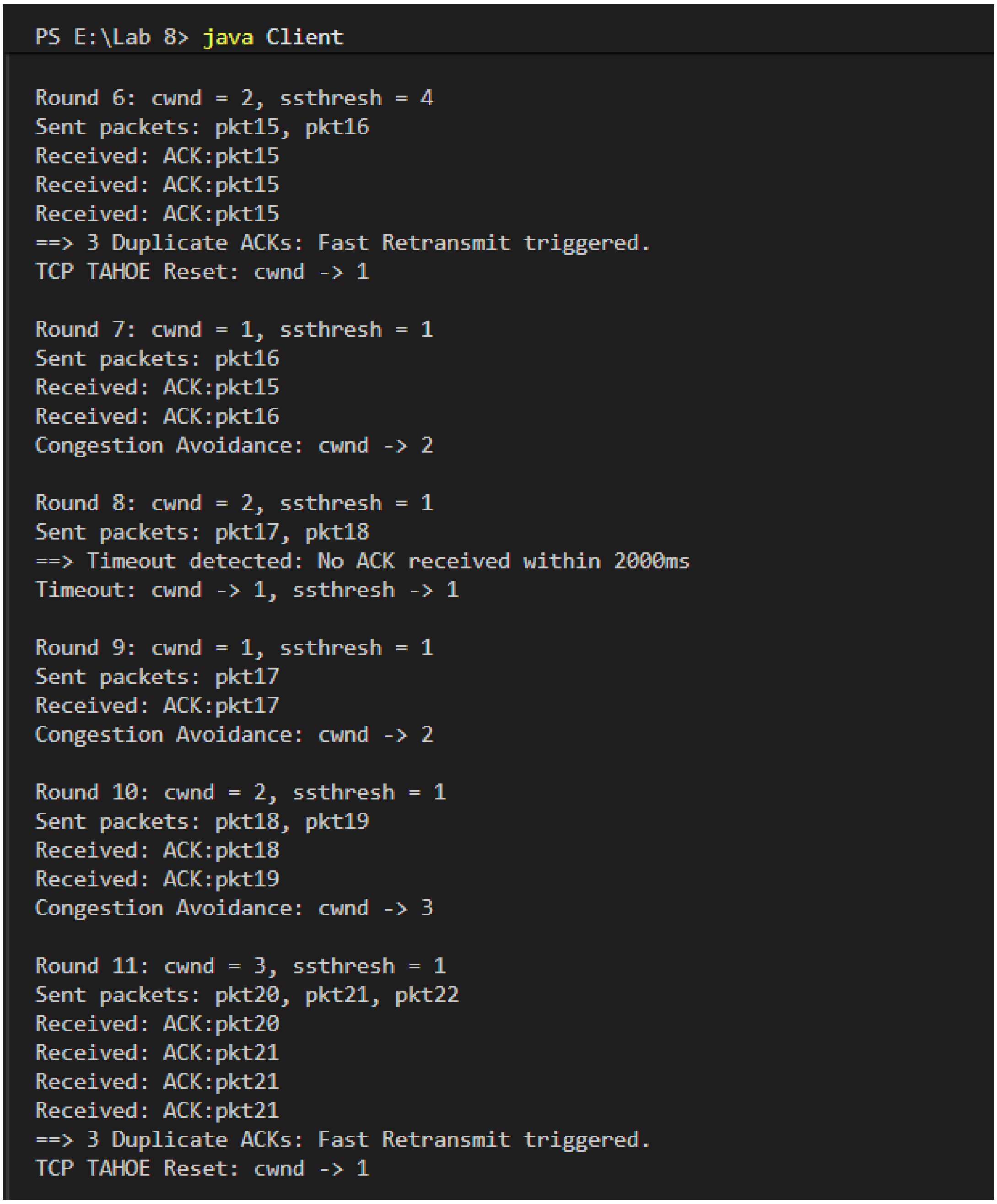
4.2 TCP Tahoe Output

TCP Tahoe — Client Output



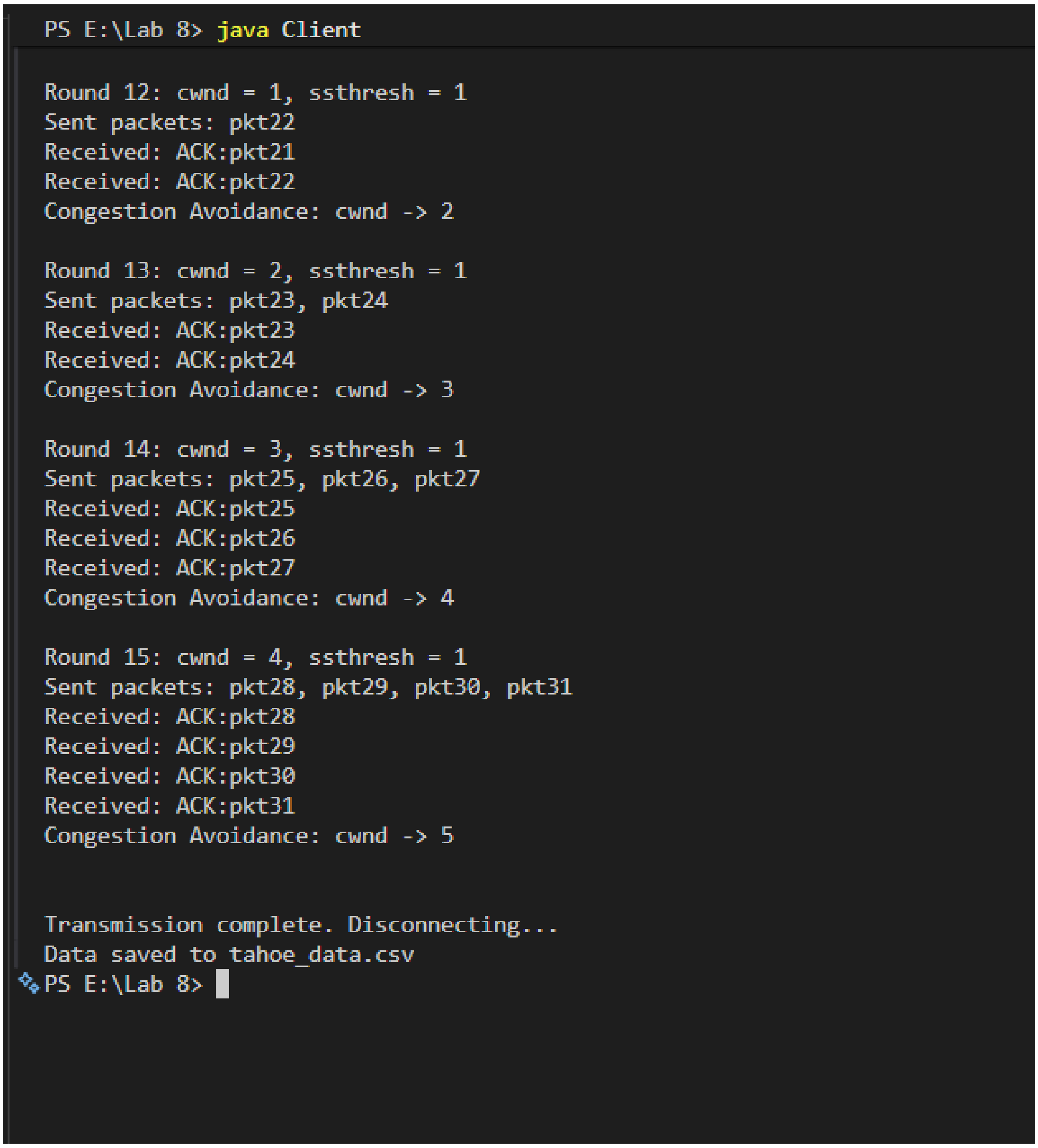
```
PS E:\Lab 8> javac Client.java
PS E:\Lab 8> java Client
Connected to Server... Waiting for READY signal...
Enter TCP mode (TAHOE / RENO): TAHOE
== TCP TAHOE Mode ==

Round 1: cwnd = 1, ssthresh = 8
Sent packets: pkt1
Received: ACK:pkt1
Slow Start: cwnd -> 2

Round 2: cwnd = 2, ssthresh = 8
Sent packets: pkt2, pkt3
Received: ACK:pkt2
Received: ACK:pkt3
Slow Start: cwnd -> 4

Round 3: cwnd = 4, ssthresh = 8
Sent packets: pkt4, pkt5, pkt6, pkt7
Received: ACK:pkt4
Received: ACK:pkt5
Received: ACK:pkt6
Received: ACK:pkt7
Slow Start: cwnd -> 8

Round 4: cwnd = 8, ssthresh = 8
Sent packets: pkt8, pkt9, pkt10, pkt11, pkt12, pkt13, pkt14, pkt15
Received: ACK:pkt8
Received: ACK:pkt9
Received: ACK:pkt10
Received: ACK:pkt11
Received: ACK:pkt12
Received: ACK:pkt13
Received: ACK:pkt13
Received: ACK:pkt13
==> 3 Duplicate ACKs: Fast Retransmit triggered.
TCP TAHOE Reset: cwnd -> 1

Round 5: cwnd = 1, ssthresh = 4
Sent packets: pkt14
Received: ACK:pkt13
Received: ACK:pkt14
Slow Start: cwnd -> 2
```

Figure 4.1: TCP Tahoe Client Output – Screenshot 1



```
PS E:\Lab 8> java Client

Round 6: cwnd = 2, ssthresh = 4
Sent packets: pkt15, pkt16
Received: ACK:pkt15
Received: ACK:pkt15
Received: ACK:pkt15
==> 3 Duplicate ACKs: Fast Retransmit triggered.
TCP TAHOE Reset: cwnd -> 1

Round 7: cwnd = 1, ssthresh = 1
Sent packets: pkt16
Received: ACK:pkt15
Received: ACK:pkt16
Congestion Avoidance: cwnd -> 2

Round 8: cwnd = 2, ssthresh = 1
Sent packets: pkt17, pkt18
==> Timeout detected: No ACK received within 2000ms
Timeout: cwnd -> 1, ssthresh -> 1

Round 9: cwnd = 1, ssthresh = 1
Sent packets: pkt17
Received: ACK:pkt17
Congestion Avoidance: cwnd -> 2

Round 10: cwnd = 2, ssthresh = 1
Sent packets: pkt18, pkt19
Received: ACK:pkt18
Received: ACK:pkt19
Congestion Avoidance: cwnd -> 3

Round 11: cwnd = 3, ssthresh = 1
Sent packets: pkt20, pkt21, pkt22
Received: ACK:pkt20
Received: ACK:pkt21
Received: ACK:pkt21
Received: ACK:pkt21
==> 3 Duplicate ACKs: Fast Retransmit triggered.
TCP TAHOE Reset: cwnd -> 1
```

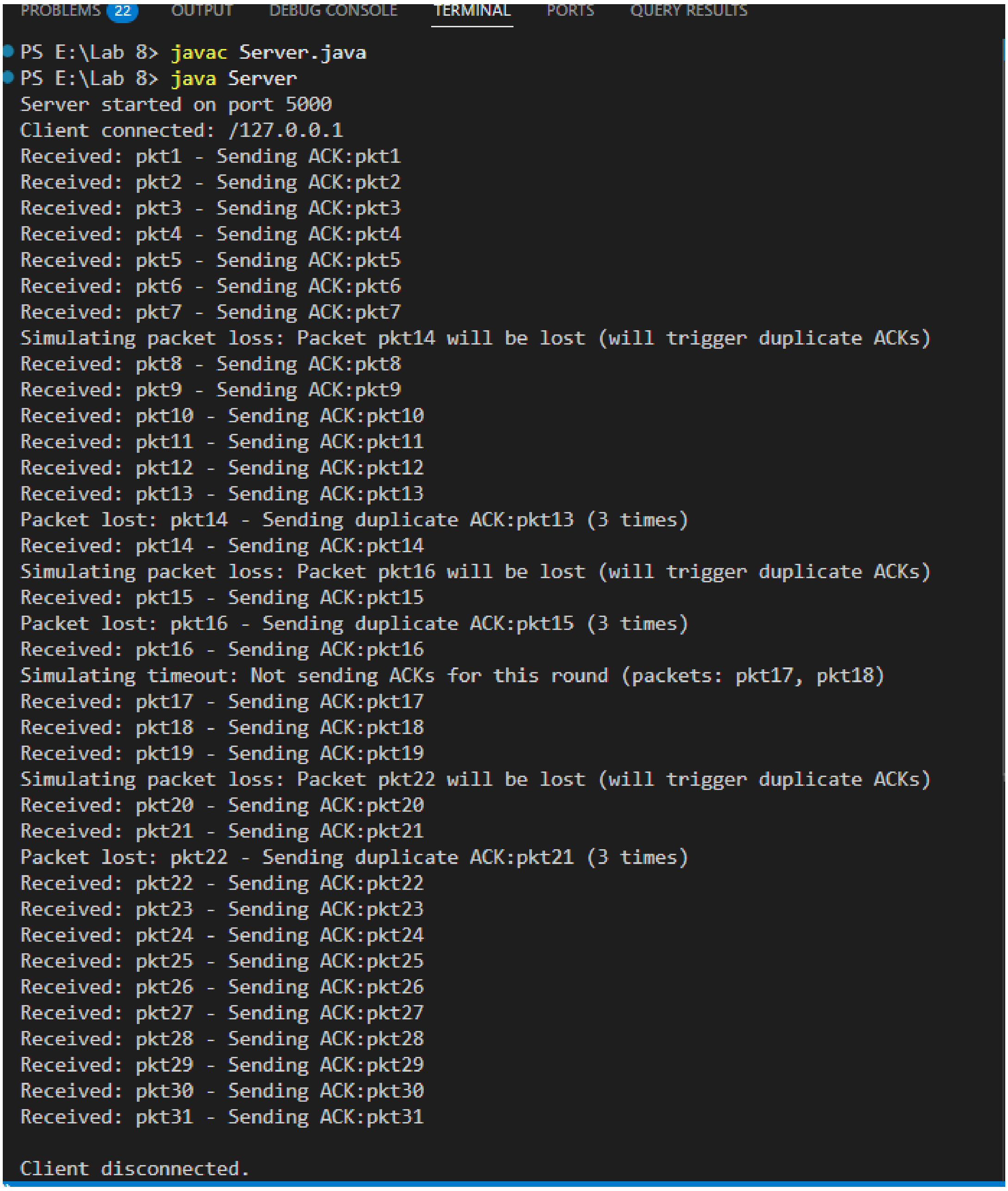
Figure 4.2: TCP Tahoe Client Output – Screenshot 2



```
PS E:\Lab 8> java Client

Round 12: cwnd = 1, ssthresh = 1
Sent packets: pkt22
Received: ACK:pkt21
Received: ACK:pkt22
Congestion Avoidance: cwnd -> 2

Round 13: cwnd = 2, ssthresh = 1
Sent packets: pkt23, pkt24
Received: ACK:pkt23
Received: ACK:pkt24
Congestion Avoidance: cwnd -> 3

Round 14: cwnd = 3, ssthresh = 1
Sent packets: pkt25, pkt26, pkt27
Received: ACK:pkt25
Received: ACK:pkt26
Received: ACK:pkt27
Congestion Avoidance: cwnd -> 4

Round 15: cwnd = 4, ssthresh = 1
Sent packets: pkt28, pkt29, pkt30, pkt31
Received: ACK:pkt28
Received: ACK:pkt29
Received: ACK:pkt30
Received: ACK:pkt31
Congestion Avoidance: cwnd -> 5


Transmission complete. Disconnecting...
Data saved to tahoe_data.csv
PS E:\Lab 8>
```

Figure 4.3: TCP Tahoe Client Output – Screenshot 3

TCP Tahoe — Server Output

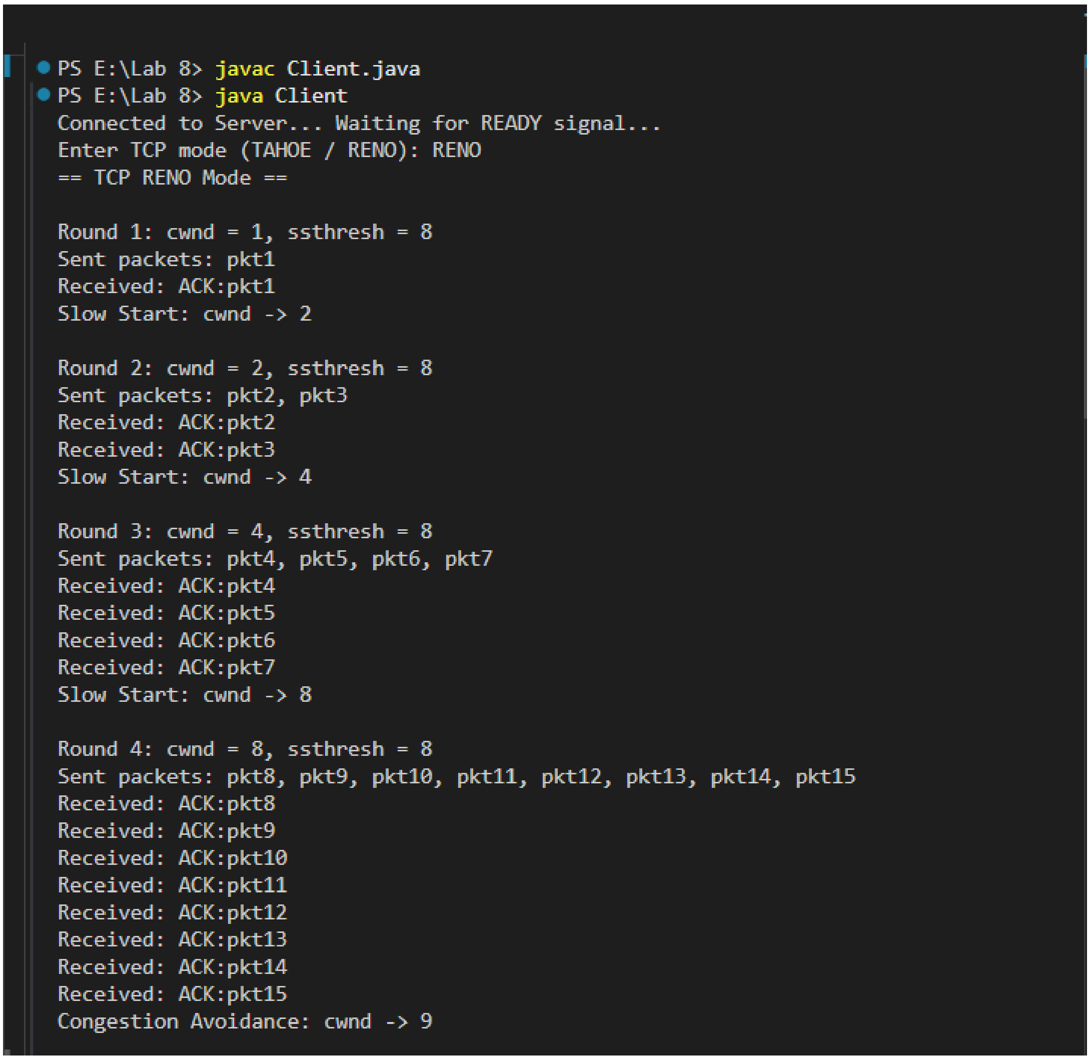


```
PS E:\Lab 8> javac Server.java
PS E:\Lab 8> java Server
Server started on port 5000
Client connected: /127.0.0.1
Received: pkt1 - Sending ACK:pkt1
Received: pkt2 - Sending ACK:pkt2
Received: pkt3 - Sending ACK:pkt3
Received: pkt4 - Sending ACK:pkt4
Received: pkt5 - Sending ACK:pkt5
Received: pkt6 - Sending ACK:pkt6
Received: pkt7 - Sending ACK:pkt7
Simulating packet loss: Packet pkt14 will be lost (will trigger duplicate ACKs)
Received: pkt8 - Sending ACK:pkt8
Received: pkt9 - Sending ACK:pkt9
Received: pkt10 - Sending ACK:pkt10
Received: pkt11 - Sending ACK:pkt11
Received: pkt12 - Sending ACK:pkt12
Received: pkt13 - Sending ACK:pkt13
Packet lost: pkt14 - Sending duplicate ACK:pkt13 (3 times)
Received: pkt14 - Sending ACK:pkt14
Simulating packet loss: Packet pkt16 will be lost (will trigger duplicate ACKs)
Received: pkt15 - Sending ACK:pkt15
Packet lost: pkt16 - Sending duplicate ACK:pkt15 (3 times)
Received: pkt16 - Sending ACK:pkt16
Simulating timeout: Not sending ACKs for this round (packets: pkt17, pkt18)
Received: pkt17 - Sending ACK:pkt17
Received: pkt18 - Sending ACK:pkt18
Received: pkt19 - Sending ACK:pkt19
Simulating packet loss: Packet pkt22 will be lost (will trigger duplicate ACKs)
Received: pkt20 - Sending ACK:pkt20
Received: pkt21 - Sending ACK:pkt21
Packet lost: pkt22 - Sending duplicate ACK:pkt21 (3 times)
Received: pkt22 - Sending ACK:pkt22
Received: pkt23 - Sending ACK:pkt23
Received: pkt24 - Sending ACK:pkt24
Received: pkt25 - Sending ACK:pkt25
Received: pkt26 - Sending ACK:pkt26
Received: pkt27 - Sending ACK:pkt27
Received: pkt28 - Sending ACK:pkt28
Received: pkt29 - Sending ACK:pkt29
Received: pkt30 - Sending ACK:pkt30
Received: pkt31 - Sending ACK:pkt31

Client disconnected.
```

Figure 4.4: TCP Tahoe Server Output

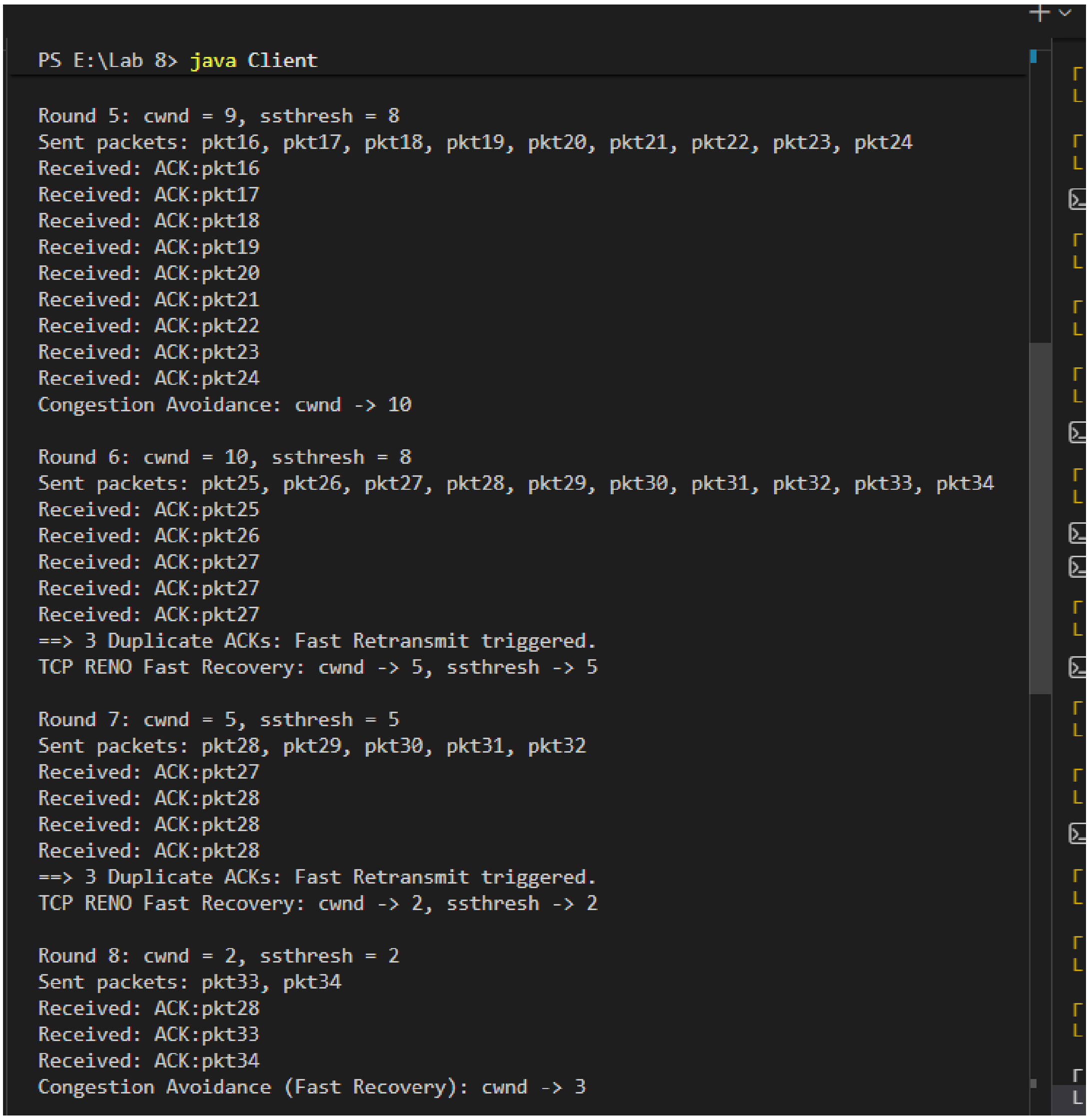
4.3 TCP Reno Output

TCP Reno — Client Output



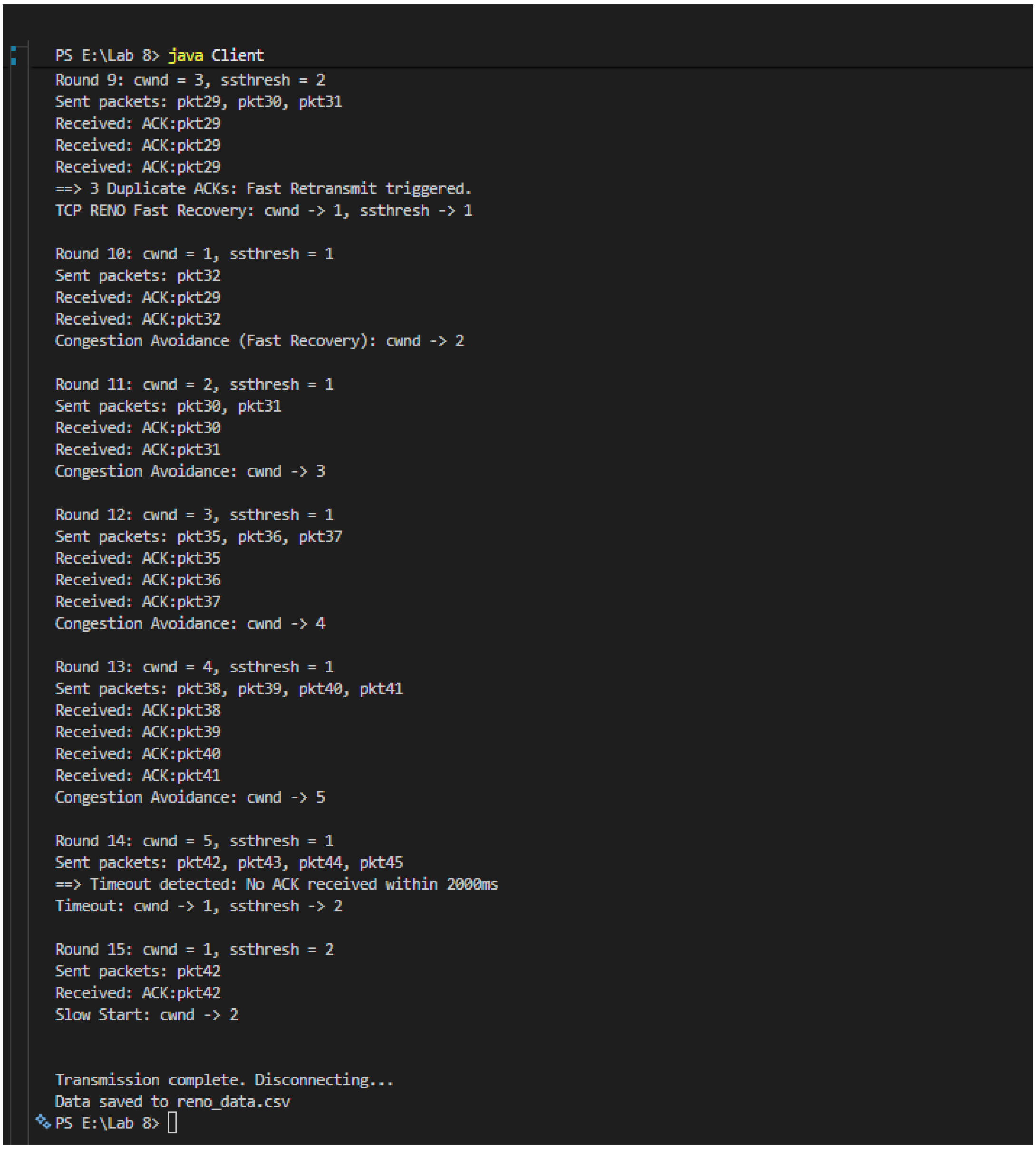
```
PS E:\Lab 8> javac Client.java
PS E:\Lab 8> java Client
Connected to Server... Waiting for READY signal...
Enter TCP mode (TAHOE / RENO): RENO
== TCP RENO Mode ==

Round 1: cwnd = 1, ssthresh = 8
Sent packets: pkt1
Received: ACK:pkt1
Slow Start: cwnd -> 2

Round 2: cwnd = 2, ssthresh = 8
Sent packets: pkt2, pkt3
Received: ACK:pkt2
Received: ACK:pkt3
Slow Start: cwnd -> 4

Round 3: cwnd = 4, ssthresh = 8
Sent packets: pkt4, pkt5, pkt6, pkt7
Received: ACK:pkt4
Received: ACK:pkt5
Received: ACK:pkt6
Received: ACK:pkt7
Slow Start: cwnd -> 8

Round 4: cwnd = 8, ssthresh = 8
Sent packets: pkt8, pkt9, pkt10, pkt11, pkt12, pkt13, pkt14, pkt15
Received: ACK:pkt8
Received: ACK:pkt9
Received: ACK:pkt10
Received: ACK:pkt11
Received: ACK:pkt12
Received: ACK:pkt13
Received: ACK:pkt14
Received: ACK:pkt15
Congestion Avoidance: cwnd -> 9
```

Figure 4.5: TCP Reno Client Output – Screenshot 1



```
PS E:\Lab 8> java Client

Round 5: cwnd = 9, ssthresh = 8
Sent packets: pkt16, pkt17, pkt18, pkt19, pkt20, pkt21, pkt22, pkt23, pkt24
Received: ACK:pkt16
Received: ACK:pkt17
Received: ACK:pkt18
Received: ACK:pkt19
Received: ACK:pkt20
Received: ACK:pkt21
Received: ACK:pkt22
Received: ACK:pkt23
Received: ACK:pkt24
Congestion Avoidance: cwnd -> 10

Round 6: cwnd = 10, ssthresh = 8
Sent packets: pkt25, pkt26, pkt27, pkt28, pkt29, pkt30, pkt31, pkt32, pkt33, pkt34
Received: ACK:pkt25
Received: ACK:pkt26
Received: ACK:pkt27
Received: ACK:pkt27
Received: ACK:pkt27
==> 3 Duplicate ACKs: Fast Retransmit triggered.
TCP RENO Fast Recovery: cwnd -> 5, ssthresh -> 5

Round 7: cwnd = 5, ssthresh = 5
Sent packets: pkt28, pkt29, pkt30, pkt31, pkt32
Received: ACK:pkt27
Received: ACK:pkt28
Received: ACK:pkt28
Received: ACK:pkt28
==> 3 Duplicate ACKs: Fast Retransmit triggered.
TCP RENO Fast Recovery: cwnd -> 2, ssthresh -> 2

Round 8: cwnd = 2, ssthresh = 2
Sent packets: pkt33, pkt34
Received: ACK:pkt28
Received: ACK:pkt33
Received: ACK:pkt34
Congestion Avoidance (Fast Recovery): cwnd -> 3
```

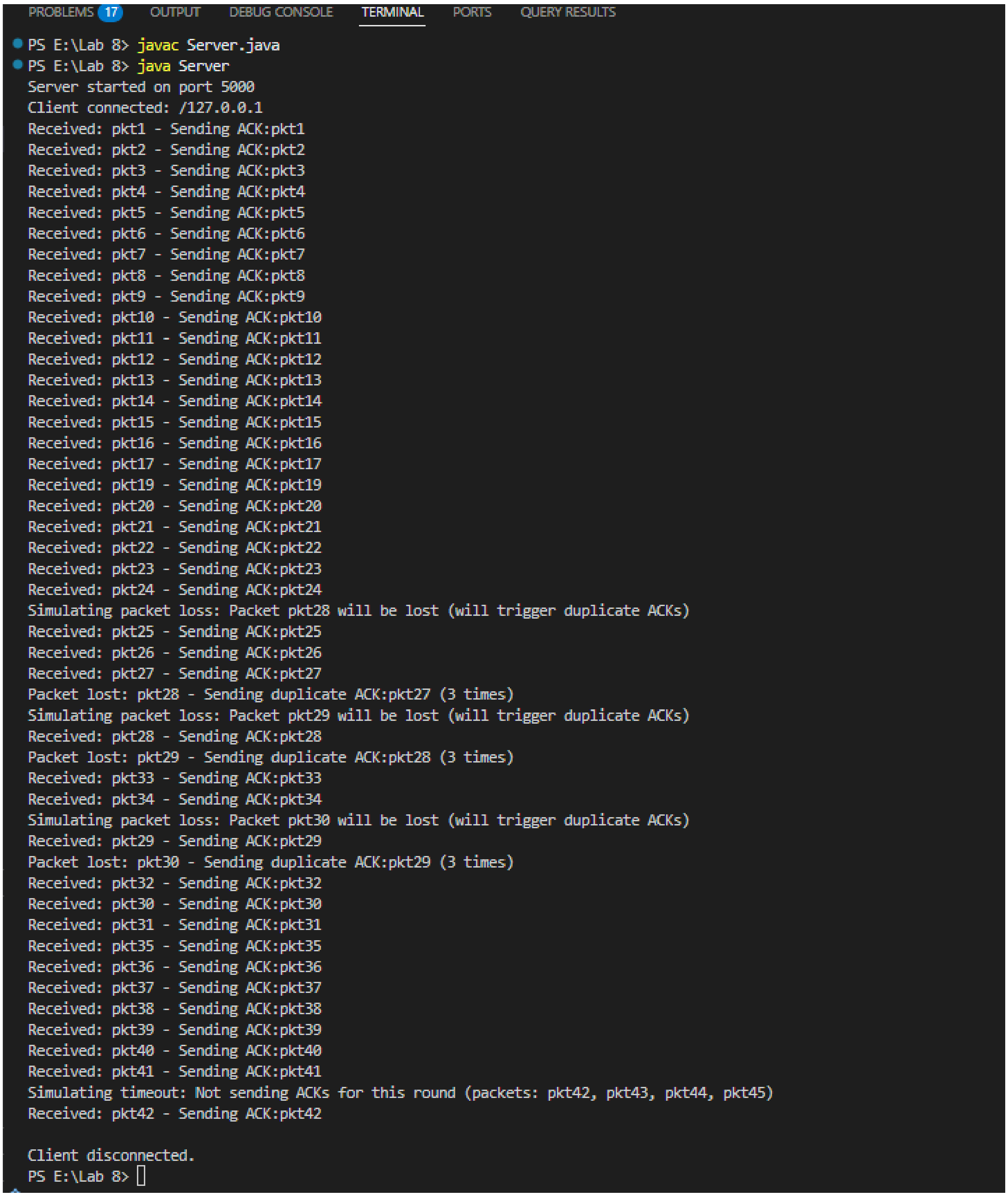
Figure 4.6: TCP Reno Client Output – Screenshot 2



```
PS E:\Lab 8> java Client
Round 9: cwnd = 3, ssthresh = 2
Sent packets: pkt29, pkt30, pkt31
Received: ACK:pkt29
Received: ACK:pkt29
Received: ACK:pkt29
==> 3 Duplicate ACKs: Fast Retransmit triggered.
TCP RENO Fast Recovery: cwnd -> 1, ssthresh -> 1

Round 10: cwnd = 1, ssthresh = 1
Sent packets: pkt32
Received: ACK:pkt29
Received: ACK:pkt32
Congestion Avoidance (Fast Recovery): cwnd -> 2

Round 11: cwnd = 2, ssthresh = 1
Sent packets: pkt30, pkt31
Received: ACK:pkt30
Received: ACK:pkt31
Congestion Avoidance: cwnd -> 3

Round 12: cwnd = 3, ssthresh = 1
Sent packets: pkt35, pkt36, pkt37
Received: ACK:pkt35
Received: ACK:pkt36
Received: ACK:pkt37
Congestion Avoidance: cwnd -> 4

Round 13: cwnd = 4, ssthresh = 1
Sent packets: pkt38, pkt39, pkt40, pkt41
Received: ACK:pkt38
Received: ACK:pkt39
Received: ACK:pkt40
Received: ACK:pkt41
Congestion Avoidance: cwnd -> 5

Round 14: cwnd = 5, ssthresh = 1
Sent packets: pkt42, pkt43, pkt44, pkt45
==> Timeout detected: No ACK received within 2000ms
Timeout: cwnd -> 1, ssthresh -> 2

Round 15: cwnd = 1, ssthresh = 2
Sent packets: pkt42
Received: ACK:pkt42
Slow Start: cwnd -> 2


Transmission complete. Disconnecting...
Data saved to reno_data.csv
PS E:\Lab 8>
```

Figure 4.7: TCP Reno Client Output – Screenshot 3

TCP Reno — Server Output



```
PS E:\Lab 8> javac Server.java
PS E:\Lab 8> java Server
Server started on port 5000
Client connected: /127.0.0.1
Received: pkt1 - Sending ACK:pkt1
Received: pkt2 - Sending ACK:pkt2
Received: pkt3 - Sending ACK:pkt3
Received: pkt4 - Sending ACK:pkt4
Received: pkt5 - Sending ACK:pkt5
Received: pkt6 - Sending ACK:pkt6
Received: pkt7 - Sending ACK:pkt7
Received: pkt8 - Sending ACK:pkt8
Received: pkt9 - Sending ACK:pkt9
Received: pkt10 - Sending ACK:pkt10
Received: pkt11 - Sending ACK:pkt11
Received: pkt12 - Sending ACK:pkt12
Received: pkt13 - Sending ACK:pkt13
Received: pkt14 - Sending ACK:pkt14
Received: pkt15 - Sending ACK:pkt15
Received: pkt16 - Sending ACK:pkt16
Received: pkt17 - Sending ACK:pkt17
Received: pkt19 - Sending ACK:pkt19
Received: pkt20 - Sending ACK:pkt20
Received: pkt21 - Sending ACK:pkt21
Received: pkt22 - Sending ACK:pkt22
Received: pkt23 - Sending ACK:pkt23
Received: pkt24 - Sending ACK:pkt24
Simulating packet loss: Packet pkt28 will be lost (will trigger duplicate ACKs)
Received: pkt25 - Sending ACK:pkt25
Received: pkt26 - Sending ACK:pkt26
Received: pkt27 - Sending ACK:pkt27
Packet lost: pkt28 - Sending duplicate ACK:pkt27 (3 times)
Simulating packet loss: Packet pkt29 will be lost (will trigger duplicate ACKs)
Received: pkt28 - Sending ACK:pkt28
Packet lost: pkt29 - Sending duplicate ACK:pkt28 (3 times)
Received: pkt33 - Sending ACK:pkt33
Received: pkt34 - Sending ACK:pkt34
Simulating packet loss: Packet pkt30 will be lost (will trigger duplicate ACKs)
Received: pkt29 - Sending ACK:pkt29
Packet lost: pkt30 - Sending duplicate ACK:pkt29 (3 times)
Received: pkt32 - Sending ACK:pkt32
Received: pkt30 - Sending ACK:pkt30
Received: pkt31 - Sending ACK:pkt31
Received: pkt35 - Sending ACK:pkt35
Received: pkt36 - Sending ACK:pkt36
Received: pkt37 - Sending ACK:pkt37
Received: pkt38 - Sending ACK:pkt38
Received: pkt39 - Sending ACK:pkt39
Received: pkt40 - Sending ACK:pkt40
Received: pkt41 - Sending ACK:pkt41
Simulating timeout: Not sending ACKs for this round (packets: pkt42, pkt43, pkt44, pkt45)
Received: pkt42 - Sending ACK:pkt42

Client disconnected.
PS E:\Lab 8> 
```

Figure 4.8: TCP Reno Server Output

4.4 Congestion Window (cwnd) Graphs

TCP Tahoe cwnd Graph

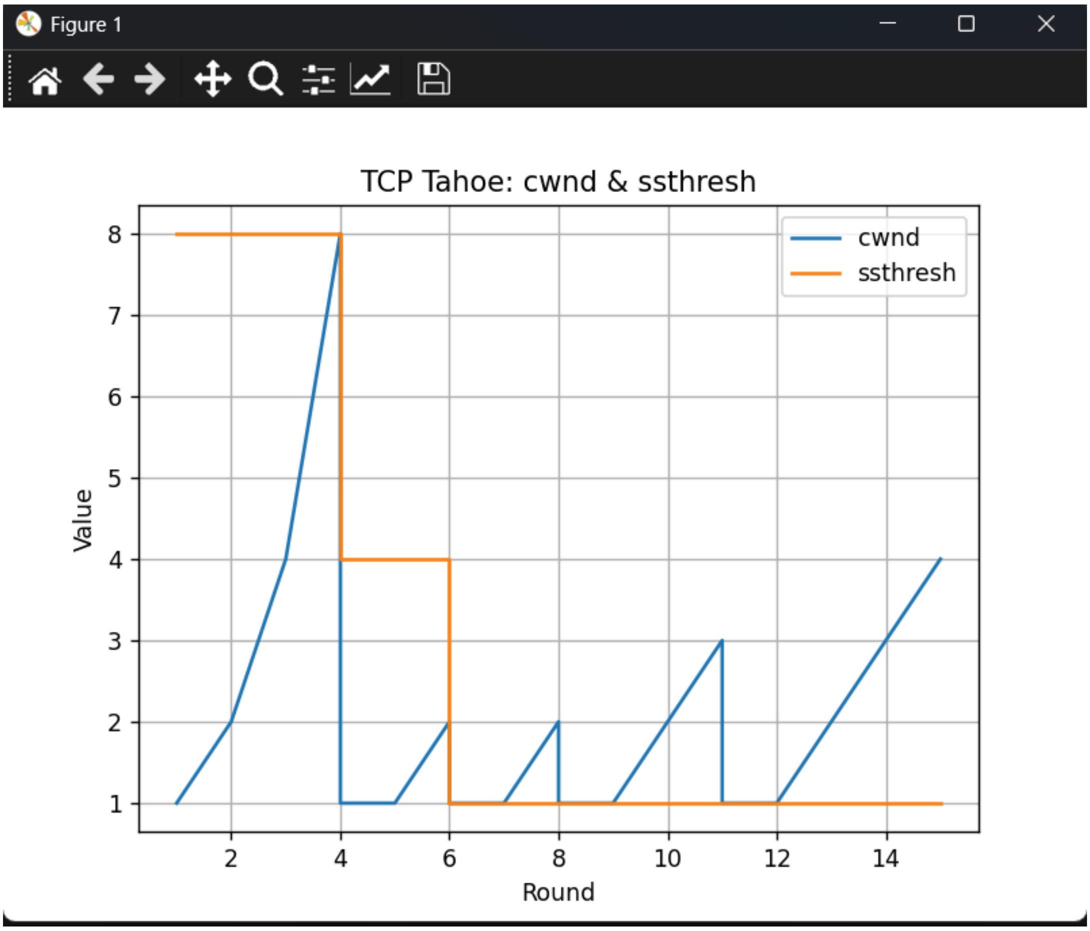


Figure 4.9: TCP Tahoe: Congestion Window vs Transmission Round

TCP Reno cwnd Graph

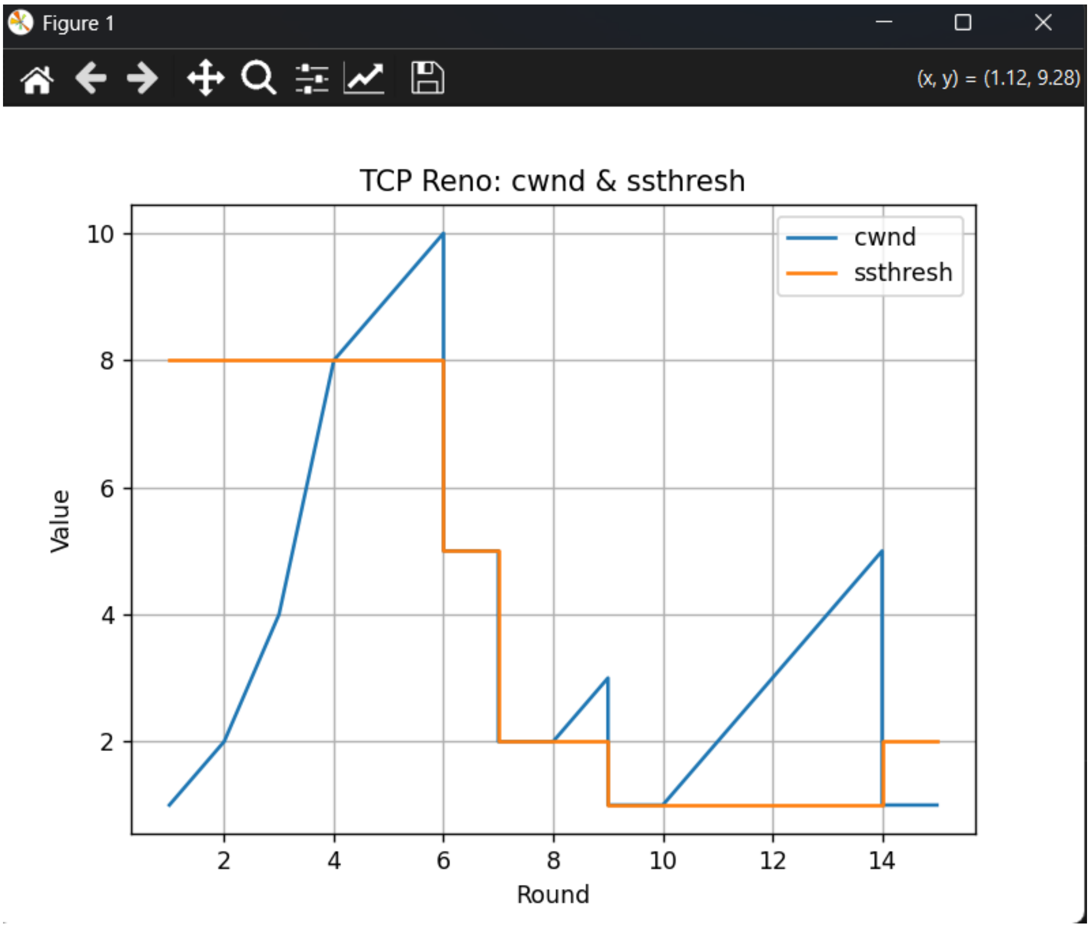


Figure 4.10: TCP Reno: Congestion Window vs Transmission Round

4.5 Performance Metrics

Packet Loss Rate

Each sudden drop in cwnd indicates a loss event.

$$\text{Packet Loss Rate}_{Tahoe} = \frac{4}{15} \approx 26.7\%$$

$$\text{Packet Loss Rate}_{Reno} = \frac{4}{15} \approx 26.7\%$$

Throughput Estimation

$$\text{Throughput} \approx \frac{\text{Average cwnd}}{\text{RTT}}$$

Average cwnd (Approximation)

$$\text{TCP Tahoe cwnd values: } 1, 2, 4, 8, 1, 1, 2, 1, 1, 2, 3, 1, 2, 3, 4$$

$$\text{Average cwnd}_{Tahoe} \approx 2.3 \text{ MSS}$$

$$\text{TCP Reno cwnd values: } 1, 2, 4, 8, 9, 10, 5, 2, 3, 1, 2, 3, 4, 5, 1$$

$$\text{Average cwnd}_{Reno} \approx 3.6 \text{ MSS}$$

Thus,

$$\text{Throughput}_{Reno} \approx 1.56 \times \text{Throughput}_{Tahoe}$$

4.6 RTT Behavior Analysis

A timeout of **2000 ms** was used.

TCP Tahoe RTT

$$RTT_{spike} \approx 2000 \text{ ms}$$

Frequent timeouts → high RTT variance.

TCP Reno RTT

$$RTT_{Reno} \approx 600\text{–}1000 \text{ ms}$$

Fewer timeouts → smoother RTT.

Summary

- Tahoe → frequent 2000 ms spikes
- Reno → smoother RTT due to fast recovery

4.7 Comparison Between TCP Tahoe and TCP Reno

Parameter	**TCP Tahoe**	**TCP Reno**
Congestion Handling	Resets cwnd to 1 MSS	Halves cwnd, Fast Recovery
Fast Retransmit	Yes	Yes
Fast Recovery	No	Yes
Throughput	Lower	Higher
RTT Stability	More fluctuating	More stable
Packet Loss Reaction	Strict	More tolerant
Best Use Case	Unstable networks	Moderate-loss networks

Table 4.1: Comparison Table: TCP Tahoe vs TCP Reno

Chapter 5

Discussion

- Reno achieves higher throughput due to Fast Recovery
- Tahoe resets cwnd to 1 → slower recovery
- RTT stability is better in Reno

What We Learned

- How congestion control impacts transfer time
- How packet loss affects cwnd and RTT
- How Tahoe and Reno differ in behavior

Difficulties Faced

- Synchronizing logs
- Handling random loss
- Ensuring correct cwnd visualization