Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: 24-07-2026

<table>
<tr><td>Course Code</td><td colspan="2">UCS3511</td><td>Course Name</td><td colspan="7">Networks Lab</td></tr>
<tr><td rowspan="2">Course Type</td><td rowspan="2" colspan="2">Lab</td><td rowspan="2">Course Category</td><td rowspan="2" colspan="2">Professional Core (PC)</td><td>L</td><td>T</td><td>P</td><td>TW & SL</td><td>TH</td><td>C</td></tr>
<tr><td>0</td><td>0</td><td>45</td><td>0</td><td>45</td><td>1.5</td></tr>
<tr><td>Regulation</td><td colspan="3">2024</td><td colspan="2">Academic Year & Batch</td><td colspan="6">2026-27 & 2024-28</td></tr>
<tr><td colspan="4">Degree and Branch</td><td colspan="8">B.E. Computer Science & Engineering</td></tr>
<tr><td colspan="4">Semester</td><td>V</td><td>Section</td><td>A, B & C</td><td colspan="2">K4, CO2</td><td colspan="3">PI: 1.1.2, 1.3.1, 1.4.1, 2.1.2, 2.1.3, 2.3.1, 3.1.1, 4.3.1, 5.1.1, 12.2.1, 12.3.1, 12.4.1 & 13.4.1</td></tr>
<tr><td>Assignment No.</td><td colspan="2">8</td><td>Assigned Date</td><td colspan="3">24-08-2026</td><td colspan="2">Due Date</td><td colspan="3">06-09-2026</td></tr>
</table>

Assignment: Simulation of TCP Congestion Control Algorithms – Reno, CUBIC and BBR

Application: High-Speed Cloud Data Transfer Network

A cloud service provider transfers large volumes of data between data centers over a shared network. Multiple TCP connections compete for available bandwidth, and network conditions may change due to congestion, varying bandwidth, and packet loss. Different TCP congestion control algorithms respond differently to these conditions.

As a network engineer, you are required to develop a **SimPy-based simulation** to model and compare the behavior of **TCP Reno, TCP CUBIC, and TCP BBR** under different network conditions. The simulation should model packet/segment transmission, congestion-window behaviour, packet loss, RTT, throughput, and congestion-control responses.

1a. TCP Sender Simulation Using SimPy

Develop a **TCP Sender module using SimPy** with the following functionalities:

1. Accept the following inputs:
    - Total amount of data to transmit.
    - Segment size.
    - Initial congestion window (`cwnd`).
    - Network bandwidth.
    - Base RTT.
    - Packet-loss probability.
    - TCP congestion-control algorithm: **Reno, CUBIC, or BBR**.
2. Generate TCP segments using a SimPy process.

3. Maintain the congestion window and determine the number of segments that can be transmitted.
4. Transmit segments through the simulated network.
5. Receive ACKs from the receiver.
6. Detect packet loss using simulated timeout or duplicate ACK events.
7. Update the congestion-control state according to the selected algorithm.
8. Record:
    - Congestion window
    - RTT
    - ACKs received
    - Packets transmitted
    - Packets retransmitted
    - Estimated throughput

1b. Network/Receiver Simulation Using SimPy

Develop a **Network/Receiver module using SimPy** with the following functionalities:

1. Receive TCP segments from the sender.
2. Introduce a configurable network propagation/transmission delay.
3. Simulate packet loss using a specified loss probability.
4. Generate ACKs for successfully received segments.
5. Maintain sequence numbers and identify missing segments.
6. Return ACKs to the sender after the simulated RTT.
7. Collect:
    - Packets received
    - Packets lost
    - ACKs generated
    - Retransmitted packets
    - RTT measurements
    - Packet delivery statistics.

1c. Implement TCP Reno

Implement the basic congestion-control behaviour of **TCP Reno**, including:

- Slow Start
- Congestion Avoidance
- Fast Retransmit
- Fast Recovery
- Congestion-window reduction after packet loss

Record the variation of `cwnd` with simulation time.

1d. Implement TCP CUBIC

Implement a simplified **TCP CUBIC** congestion-control model where the congestion window changes according to a cubic growth function.

The simulation should:

- Increase `cwnd` during congestion avoidance.
- Record the maximum congestion window before loss.
- Reduce the congestion window after packet loss.
- Model the characteristic cubic growth behaviour.
- Record `cwnd`, RTT, packet loss, and throughput.

1e. Implement TCP BBR

Implement a simplified **TCP BBR** model based on estimated:

- Bottleneck bandwidth
- Minimum RTT

The simulation should:

- Estimate available bandwidth from successfully delivered data.
- Estimate minimum RTT.
- Adjust the sending rate based on bandwidth and RTT measurements.
- Model the major BBR operating phases in a simplified manner.
- Record sending rate, RTT, congestion window/in-flight data, and throughput.

1f. Comparative Simulation

Run the simulation using **Reno, CUBIC, and BBR** under the same network conditions.

Perform experiments for:

Scenario 1 – Low Congestion

- Low packet-loss probability
- Stable RTT
- Adequate bandwidth

Scenario 2 – High Packet Loss

- Increased packet-loss probability
- Constant bandwidth
- Stable RTT

Scenario 3 – Variable RTT

- Changing network delay
- Low packet-loss probability

Scenario 4 – Limited Bandwidth

- Reduced bottleneck bandwidth
- Moderate packet loss

Analyze how each algorithm responds to the different network conditions.

Sample Data for Laboratory Verification

Test Case	Data	Segment Size	Bandwidth	Base RTT	Loss Probability	Algorithm
TC1	1 MB	1 KB	10 Mbps	20 ms	0%	Reno
TC2	1 MB	1 KB	10 Mbps	20 ms	2%	Reno
TC3	1 MB	1 KB	10 Mbps	20 ms	2%	CUBIC
TC4	1 MB	1 KB	10 Mbps	20 ms	2%	BBR
TC5	2 MB	1 KB	5 Mbps	80 ms	5%	Reno/CUBIC/BBR
TC6	2 MB	1 KB	20 Mbps	100 ms	1%	Reno/CUBIC/BBR

For **TC5 and TC6**, use the same network parameters for all three algorithms to enable a fair comparison.

Sample Client/Sender Input

```
TCP CONGESTION CONTROL SIMULATION

Total Data          : 1 MB
Segment Size        : 1 KB
Bandwidth           : 10 Mbps
Base RTT            : 20 ms
Packet Loss         : 2%
Initial cwnd        : 1 MSS

Select Algorithm:
1. Reno
2. CUBIC
3. BBR

Enter Choice: 1
```

Sample Output – Reno

```
Algorithm : TCP Reno

Time      cwnd      RTT       Event
-----------------------------------------------
0.00 s    1 MSS     20 ms     Slow Start
0.02 s    2 MSS     20 ms     ACK Received
0.04 s    4 MSS     20 ms     ACK Received
0.08 s    8 MSS     22 ms     ACK Received
...
0.42 s    16 MSS    25 ms     Packet Loss
0.42 s    8 MSS     25 ms     Fast Recovery
...
```

Performance Observation Table

Students should record the actual values obtained from their simulation.

Algorithm	Avg. Throughput	Avg. RTT	Packets Lost	Retransmissions	Avg. cwnd	Completion Time
Reno	Varies	Varies	Varies	Varies	Varies	Varies
CUBIC	Varies	Varies	Varies	Varies	Varies	Varies
BBR	Varies	Varies	Varies	Varies	Varies	Varies

For the test case:

- Data = **1 MB**
- Segment size = **1 KB**
- Bandwidth = **10 Mbps**
- RTT = **20 ms**
- Packet loss = **2%**

a representative observation table could be:

Performance Observation Table – Sample Results

Algorithm	Avg. Throughput (Mbps)	Avg. RTT (ms)	Packets Lost	Retransmissions	Avg. cwnd (KB)	Completion Time (s)
TCP Reno	6.8	23.5	21	28	18.4	1.24
TCP CUBIC	8.1	22.8	20	24	27.6	1.04
TCP BBR	9.0	21.6	19	21	31.2	0.93

Interpretation

- **Reno:** Shows relatively conservative congestion-window growth. Packet loss causes a significant reduction in `cwnd`, resulting in lower throughput.
- **CUBIC:** Recovers the congestion window more aggressively and therefore achieves higher throughput than Reno.
- **BBR:** Uses estimated bottleneck bandwidth and RTT rather than relying primarily on packet loss as a congestion signal. In this sample scenario, it achieves the highest throughput and shortest completion time.
- The **average RTT** remains relatively close to the base RTT because the network is not heavily congested.
- The exact number of lost packets can vary because the **2% packet-loss probability is randomized**.

For the Full Comparison: Reference observation table:

Test Case	Algorithm	Avg. Throughput (Mbps)	Avg. RTT (ms)	Packets Lost	Retransmissions	Completion Time (s)
TC1: 10 Mbps, 20 ms, 0% loss	Reno	8.2	20.8	0	0	1.02
	CUBIC	9.1	20.5	0	0	0.92
	BBR	9.5	20.3	0	0	0.88
TC2: 10 Mbps, 20 ms, 2% loss	Reno	6.8	23.5	21	28	1.24
	CUBIC	8.1	22.8	20	24	1.04
	BBR	9.0	21.6	19	21	0.93
TC5: 5 Mbps, 80 ms, 5% loss	Reno	2.9	91.4	51	68	2.76
	CUBIC	3.7	87.2	49	59	2.16
	BBR	4.4	83.6	48	53	1.82
TC6: 20 Mbps, 100 ms, 1% loss	Reno	11.2	108.5	10	15	0.75
	CUBIC	15.4	105.8	10	13	0.54
	BBR	18.1	102.7	9	11	0.46

Analysis and Comparison

Analyze the simulation results based on:

1. **Congestion Window:** How does `cwnd` change after packet loss?
2. **Throughput:** Which algorithm achieves higher throughput under each network condition?
3. **RTT:** How does network delay affect the three algorithms?
4. **Packet Loss:** How does each algorithm respond to increasing packet loss?
5. **Bandwidth Utilization:** Which algorithm utilizes the available bandwidth more effectively?
6. **Retransmissions:** Compare the number of retransmitted segments.
7. **Stability:** Compare the variation of the congestion window over time.
8. **Overall Performance:** Identify the most suitable algorithm for low-loss, high-loss, high-RTT, and bandwidth-limited scenarios.

Suggested Graphs

Generate graphs using the simulation results:

- **Time vs. Congestion Window**

- **Time vs. Throughput**
- **Time vs. RTT**
- **Packet Loss vs. Throughput**
- **Algorithm vs. Average Throughput**

Expected Deliverables

- SimPy-based TCP Sender program
- SimPy-based Network/Receiver program
- Implementation of Reno, CUBIC, and BBR
- Sample input and output
- Simulation screenshots
- Performance observation tables
- Comparative graphs
- Analysis of results
- Conclusion and recommendation

Note: Students should record the **aim, problem analysis, SimPy-based sender/network/receiver algorithms, source code, sample input/output, screenshots, test cases, performance observations, graphs, comparative analysis, and conclusion** in the laboratory record.

Expected Learning Outcome – K4

Students will **analyze and compare TCP Reno, CUBIC, and BBR** by modelling their congestion-control behaviour using SimPy and evaluating the impact of **bandwidth, RTT, packet loss, congestion-window dynamics, throughput, and retransmissions** under different network conditions.