COMPUTER NETWORKS LAB ASSIGNMENT-3

Name: Suman Khara

Class: BCSE-III

Section: A2

Roll Number: 002210501098

Assignment Number: 3

Problem Statement: Implement p-persistent CSMA technique with collision detection.

DESIGN

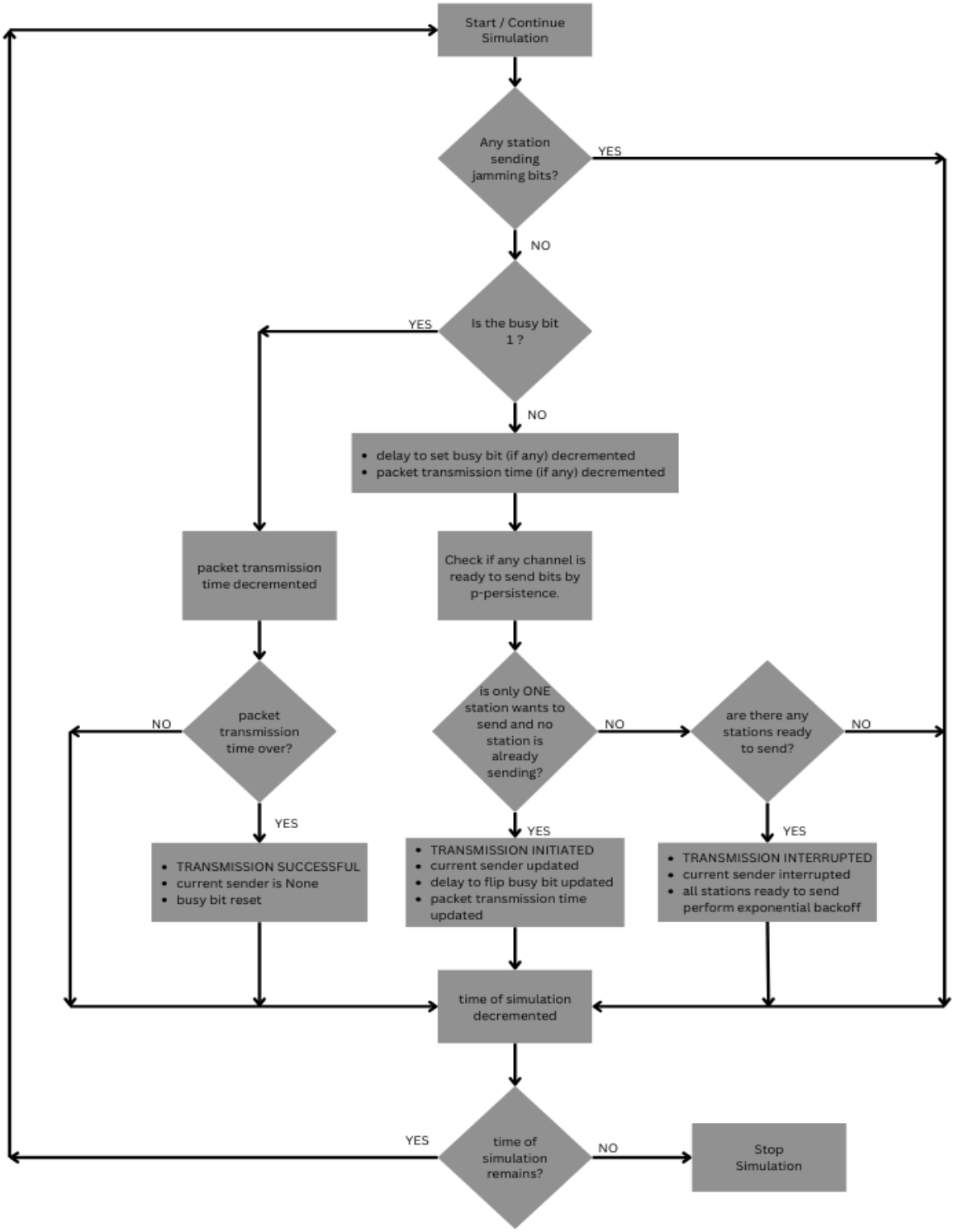

Fig 1: Flowchart of the Simulation

This assignment simulates the p-persistent CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocol, focusing on core aspects of collision detection, backoff mechanisms, and controlled data transmission in a shared communication channel.

1. Objective

- The goal is to simulate the p-persistent CSMA/CD protocol with multiple stations that transmit data frames over a shared channel. This design aims to observe channel access, collision handling, and the backoff process to resolve contentions.
- Key parameters include the probability of transmission (p), frame sizes for each station, channel bandwidth, and propagation delay (Tp).

2. Components and Classes

Channel Class

- **Purpose:** Represents the shared communication channel. Maintains the channel state and keeps track of busy periods, finish delays, and which station currently occupies the channel.
- **Attributes:**
    - bandwidth: Specifies data transmission rate in bits per millisecond.
    - Tp: Maximum propagation delay, crucial for calculating collision detection times.
    - busy: A boolean flag indicating channel availability.
    - current_station: Tracks which station currently occupies the channel.
    - busy_delay and finish_delay: Timers for channel availability and transmission completion.

Station Class

- **Purpose:** Represents each station participating in the CSMA/CD protocol. Each station has unique attributes like frame size and

probability of transmission, along with internal timers and counters for handling the backoff process.

- **Attributes:**
    - ID: Unique identifier for the station.
    - frame_size: Size of data frame to transmit.
    - p: Probability that the station will transmit in the current slot.
    - waiting_period and drop_period: Timers for transmission cooldown and dropouts after multiple collisions.
    - sending and jamming: Flags for tracking the station's transmission status and collision occurrences.
- **Methods:**
    - p_persistence(): Checks transmission probability and decides if the station attempts transmission.
    - exponential_backoff(): Implements exponential backoff, doubling waiting time with each collision.
    - interrupt(): Called during a collision to halt transmission and initiate backoff.

3. Simulation Workflow

- The simulation loop runs for a defined time in milliseconds, representing discrete time slots. Each iteration corresponds to a 1ms slot.
- **Collision Detection and Handling:**
    - When multiple stations attempt to access the channel simultaneously, a collision occurs. The current station is interrupted, and both stations engage in exponential backoff.

- **Transmission Success:**
    - If only one station attempts transmission while the channel is free, it is granted access. After frame_size/bandwidth milliseconds, transmission completes, and the channel becomes available again.

4. Design Choices

- **1ms Time Slot:** Ensures fine-grained simulation, capturing rapid changes in channel status and accurately modeling delays.
- **p-persistence Model:** Adds randomness, reflecting real-world conditions where stations may not always transmit immediately upon sensing an idle channel.
- **Exponential Backoff Strategy:** Simulates adaptive waiting times after collisions, mimicking real CSMA/CD network behavior and helping to reduce further collisions.
- **Logging and Output:** Events are logged to a text file and printed to the console, providing real-time insights into the channel and station states.

5. Parameterization and Customization

- The simulation is parameterized, allowing easy adjustments of bandwidth, Tp, frame_size, and p. This flexibility allows testing of various network conditions, making the simulation versatile and adaptable.

IMPLEMENTATION

```
import random


class Channel:
    def __init__(self, bandwidth, Tp):
        self.bandwidth = bandwidth
        self.Tp = Tp
        self.busy = False
        self.current_station = None
        self.busy_delay = 0
        self.finish_delay = 0


class Station:
    def __init__(self, ID, frame_size, channel, p):
        self.ID = ID
        self.frame_size = frame_size
        self.channel = channel
        self.p = p
        self.ready = True
```

```
        self.waiting_period = 0
        self.drop_period = 0
        self.k = 0
        self.jamming = False
        self.count = 0
        self.interrupted = 0
        self.dropped = 0
        self.sending = False

    def exponential_backoff(self):
        if self.k < 15:
            self.k += 1
            self.waiting_period = random.randint(0, 2 ** self.k - 1)
        else:
            self.k = 0
            self.drop_period = 5
            self.dropped += 1

    def p_persistence(self):
        if self.sending:
            return False
        elif self.drop_period > 0:
            self.drop_period -= 1
            return False
        elif self.waiting_period == 0 and not self.channel.busy:
            if random.random() < self.p:
                return True
            else:
                return False
        elif self.waiting_period > 0:
            self.waiting_period -= 1
            return False
        else:
            self.exponential_backoff()

    def interrupt(self):
        self.channel.busy_delay = 0
        self.channel.current_station = None
        self.jamming = True
        self.sending = False
        self.exponential_backoff()
        self.interrupted += 1


def simulate(stations, channel, time):
    time_slots = time * 1000

    with open('log.txt', 'w') as log:
        while time_slots > 0:
```

```
            jamming = False
            for _, station in stations.items():
                if station.jamming:
                    jamming = True
                    log.write(f"Jamming by {station.ID}\n")
                    station.jamming = False
                    break

            if jamming:
                time_slots -= 1
                continue

            if not channel.busy:
                if channel.busy_delay > 0:
                    channel.busy_delay -= 1
                if channel.finish_delay > 0:
                    channel.finish_delay -= 1

                if channel.busy_delay == 0 and channel.current_station:
                    channel.busy = True
                    log.write("Busy bit enabled")

                ready = []
                for _, station in stations.items():
                    if station.p_persistence():
                        ready.append(station)

                if channel.current_station is None and len(ready) == 1:
                    channel.busy_delay = channel.Tp
                    channel.finish_delay = ready[0].frame_size //
channel.bandwidth
                    channel.current_station = ready[0].ID
                    ready[0].sending = True
                    log.write(f"{ready[0].ID} started sending\n")

                elif ready:
                    if channel.current_station:
                        log.write(f"{channel.current_station} interrupted. ")
                        stations[channel.current_station].interrupt()
                    log.write("Exponential backoffs performed by: ")
                    for station in ready:
                        station.exponential_backoff()
                        log.write(f"{station.ID} ")
                    log.write("\n")
                else:
                    log.write(".\n")
            else:
                if channel.finish_delay > 0:
```

```
                    channel.finish_delay -= 1

                if channel.finish_delay == 0:
                    stations[channel.current_station].count += 1
                    stations[channel.current_station].sending = False
                    stations[channel.current_station].drop_period = 1
                    channel.current_station = None
                    channel.busy = False
                    log.write("Transmission successful. Busy bit disabled\n")
                else:
                    log.write(".\n")

            time_slots -= 1


channel = Channel(bandwidth=1000, Tp=5)
stations = {
    "A": Station(ID="A", frame_size=12000, channel=channel, p=0.33),
    "B": Station(ID="B", frame_size=16000, channel=channel, p=0.33),
    "C": Station(ID="C", frame_size=11000, channel=channel, p=0.33)
}

simulate(stations, channel, time=50)

for id, station in stations.items():
    print(f'{id} successful: {station.count}, interrupted: {
          station.interrupted}, dropped: {station.dropped}')
```

Output:

A successful: 371, interrupted: 5, dropped: 0

B successful: 858, interrupted: 2, dropped: 1

C successful: 781, interrupted: 6, dropped: 0

Observation:

The observation reflects the CSMA/CD protocol's impact on transmission success, interruptions, and packet drops over a shared channel among three stations with equal probabilities (`p=0.33`). Station B achieved the highest successful transmissions (858), followed by Station C (781) and Station A (371). Interruptions were relatively infrequent, with only a few collisions leading to backoff, as seen in each station's low interruption count. Notably, only Station

B experienced a dropped transmission, due to repeated collisions reaching the backoff limit. This indicates a well-functioning p-persistent CSMA/CD setup with minimal collisions and efficient channel sharing among stations.

COMPARISON GRAPHS AND ANALYSIS

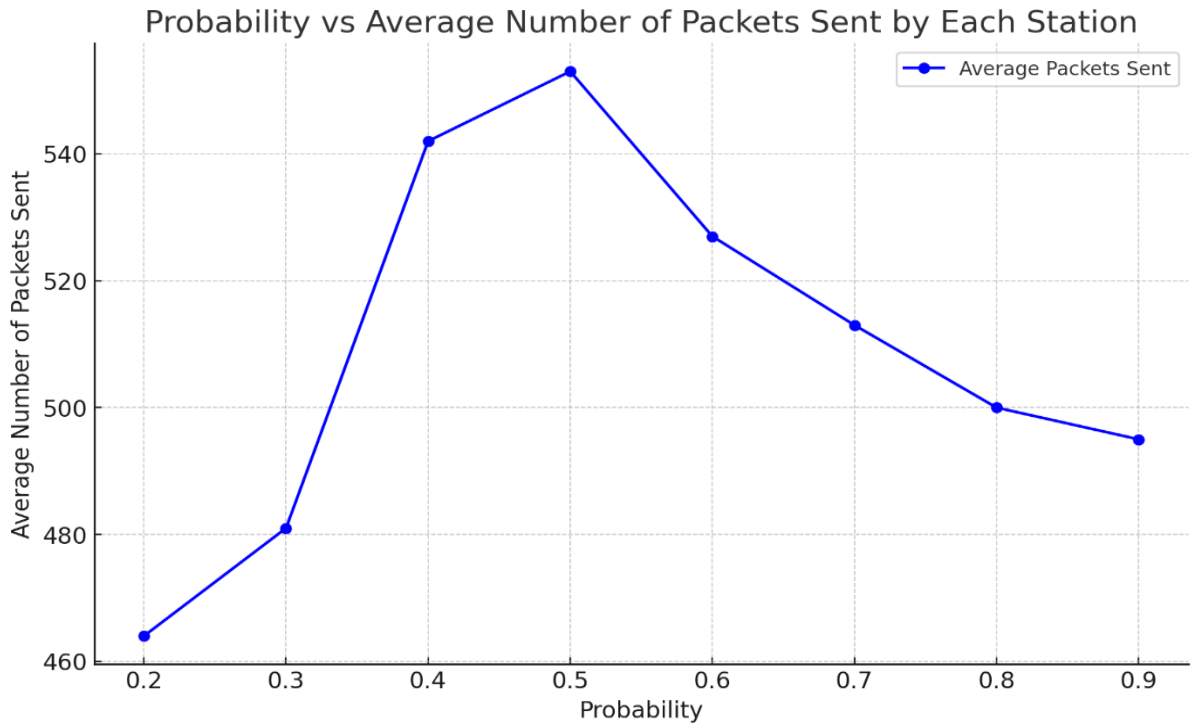

Fig 2: Probability vs Average number of packets sent by each station. Number of stations=5. Time=50 seconds. Time Slot=1ms. Frame size=12kb. Bandwidth=1kbps.

Probability vs. Average Number of Packets Sent by Each Station

In this plot, we observe the relationship between the probability of successful transmission attempts and the average number of packets sent by each station. This relationship typically shows that:

- **Low Probabilities**: At low transmission probabilities, fewer stations attempt to transmit at any given time, resulting in a lower average number of packets sent due to underutilization of the channel. Fewer attempts reduce the chances of collisions, which leads to fewer retries but also reduces the network throughput.

- **Optimal Probability Range**: As the probability increases, stations begin to transmit more packets, increasing the average number of packets sent. In this range, the channel utilization improves, and more packets are successfully sent by each station. This is the region where the network operates most efficiently, balancing throughput and collision avoidance.
- **High Probabilities**: At very high transmission probabilities, all stations frequently attempt to transmit, which leads to an increase in collisions. Consequently, each station's effective packet transmission rate starts to drop due to excessive retransmissions, resulting in a decline in the average packets sent per station.

This graph highlights the need for an optimal probability setting to maximize throughput while minimizing collisions.

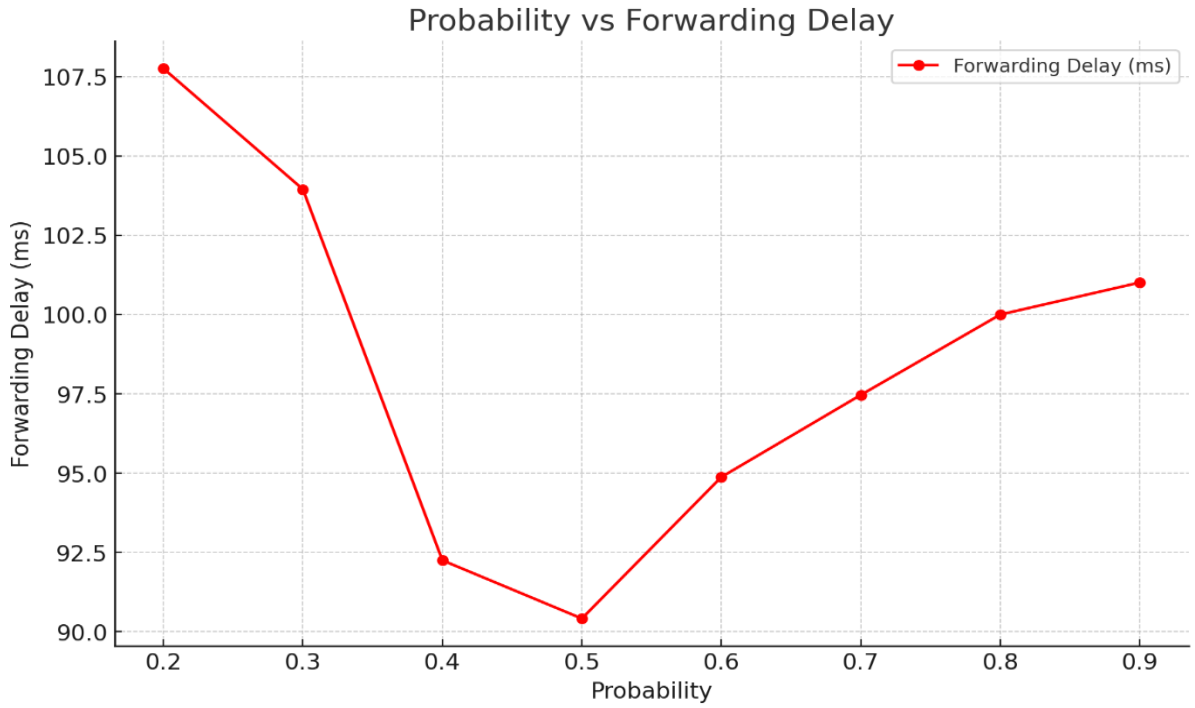

Fig 3: Probability vs Average Forwarding Delay. Number of stations=5. Time=50 seconds. Time Slot=1ms. Frame size=12kb. Bandwidth=1kbps.

Probability vs. Average Forwarding Delay

This plot illustrates how the probability of transmission attempts affects the forwarding delay, which represents the time taken for a packet to be successfully transmitted from one station to another.

- **Low to Moderate Probabilities**: Initially, as probability increases, there is an incremental rise in forwarding delay. This is because with low traffic, fewer collisions occur, allowing each packet to have a lower waiting time before successful transmission.
- **Moderate to High Probabilities**: With higher probabilities, stations attempt to transmit more frequently, leading to an increase in the likelihood of collisions. This results in multiple retransmissions, elevating the forwarding delay significantly as packets wait longer for an available transmission slot.
- **Very High Probabilities**: When probabilities are extremely high, the delay reaches a peak due to saturation of the network. Collisions occur frequently, and each packet experiences substantial delay before it can be forwarded successfully.

This plot demonstrates that, as transmission probability increases, the forwarding delay initially remains manageable but eventually escalates sharply due to increased collisions, suggesting that network performance is optimized at intermediate probabilities.

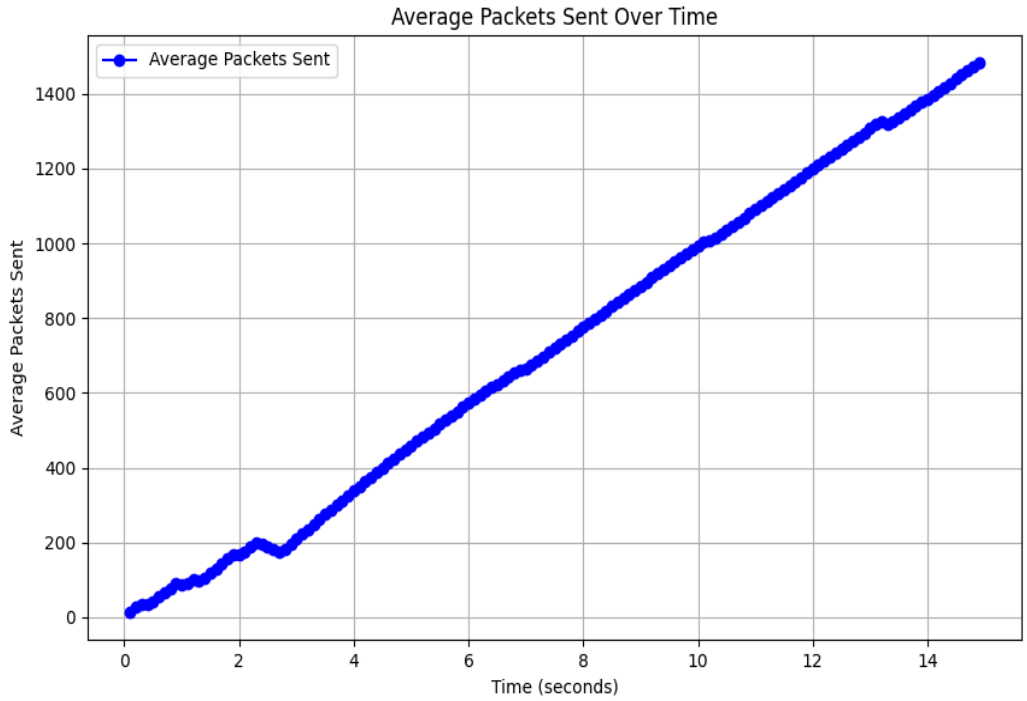

Fig 4: Time vs Average Packets sent by each station per second. Number of stations=3. P=0.33. Time Slot=1ms. Frame size=12kb. Bandwidth=1kbps.

Time vs. Average Packets Sent per Station per Second

This plot shows the average number of packets sent by each station over time, providing insight into how the network's performance evolves as more packets are sent.

- **Linear Growth Pattern**: The plot displays a near-linear increase in the average packets sent over time. This suggests that the network is handling the load effectively, with each station maintaining a stable rate of successful packet transmissions. The linear growth indicates a balanced throughput, where each station is able to send packets consistently without overwhelming the network.
- **Absence of Saturation**: The lack of plateauing or dips suggests that the network is not experiencing significant congestion or collisions even as time progresses. In this scenario, the network may be lightly loaded or configured with an optimal transmission probability, allowing stations to avoid excessive retransmissions and maintain a smooth data flow.
- **Sustained Throughput**: The steady rise in packets sent indicates that each station is able to consistently utilize the available channel capacity without notable interruptions. This consistency points to an effective protocol configuration where stations do not frequently collide or interfere with each other's transmissions, enabling a continuous, high-throughput performance over time.

COMMENTS

This CSMA/CD (Carrier Sense Multiple Access with Collision Detection) simulation assignment provides valuable insights into network performance under a probabilistic transmission protocol. Through the implementation of CSMA/CD, we observe how varying transmission probabilities and time intervals impact crucial metrics such as the average number of packets sent, forwarding delay, and throughput over time. The analysis of these metrics reveals essential aspects of network behavior under collision-detection conditions, making it possible to evaluate the efficiency and stability of the protocol. This assignment reinforces fundamental networking concepts, particularly collision management in shared-channel protocols. Through hands-on simulation, it becomes clear how critical parameters like transmission

probability impact network performance, and we gain a deeper appreciation for the complexities involved in designing robust network protocols.

In summary, simulating CSMA/CD offered an invaluable perspective on network protocol dynamics and demonstrated how careful configuration can lead to efficient and reliable communication in shared-channel networks.
