Name-Soumyadeep Singh

Class-BCSE 3

Roll-002210501018

Section-A1

Computer Networks Lab

Assignment 3

Implement p-persistent CSMA technique with collision detection.

Detailed Simulation Analysis of p-Persistent CSMA/CD Protocol

This simulation models the essential principles of the p-persistent CSMA/CD (Carrier Sense Multiple Access with Collision Detection) protocol. The focus is on observing collision detection, the backoff mechanism, and data transmission control within a shared communication channel, reflecting key aspects of network communication.

---

1. Objectives of the Simulation:

- **Primary Goal**: To create a simulation of the p-persistent CSMA/CD protocol, representing multiple stations that share a single channel. Each station attempts to transmit data frames while competing for channel access. Key objectives include understanding how the protocol manages access, resolves collisions, and maintains communication efficiency.
- **Parameters in Focus**: Essential parameters include:
    - **Transmission Probability (p)**: Determines the likelihood of a station attempting to send data in any given slot.
    - **Frame Size**: Varies per station and impacts transmission duration.
    - **Channel Bandwidth**: Controls data transfer rate.
    - **Propagation Delay (Tp)**: Accounts for signal travel time, crucial in detecting and responding to collisions.

---

2. Components and Key Classes:

- **Channel Class**:
    - **Purpose**: Represents the shared channel, maintaining channel availability, managing data transmission times, and coordinating access between stations.
    - **Attributes**:
        - **bandwidth**: Defines the data rate in bits per millisecond, dictating how quickly a station can send data.

        - **Tp**: Maximum propagation delay, essential for collision detection, representing signal travel time across the channel.
        - **busy**: Boolean flag indicating whether the channel is currently occupied.
        - **current_station**: Tracks the station currently using the channel.
        - **busy_delay and finish_delay**: Timers controlling the channel's busy status and monitoring when it becomes available for the next transmission.
    - **Function**: The channel class plays a central role, handling time-sensitive attributes that govern how stations contend for access and respond to collisions.
- **Station Class**:
    - **Purpose**: Models each individual station in the network, including parameters for transmission behavior, frame characteristics, and collision response.
    - **Attributes**:
        - **ID**: Unique identifier for each station, aiding in tracking transmission status.
        - **frame_size**: Specifies the size of the data frame being sent by the station.
        - **p**: Probability of transmission, used in the p-persistence model to introduce randomness in channel access.
        - **waiting_period and drop_period**: Internal timers managing the station's wait time after collisions and dropout intervals.
        - **sending and jamming**: Flags tracking whether a station is transmitting or experiencing a collision.
    - **Key Methods**:
        - **p_persistence()**: Checks if the station should attempt transmission based on probability p, introducing a randomized access approach.
        - **exponential_backoff()**: Implements exponential backoff after a collision, doubling the waiting time for each consecutive collision to help reduce further contentions.

        - **interrupt()**: Activated upon a collision, immediately stopping transmission and triggering the backoff process to prevent channel overcrowding.

---

3. Simulation Workflow and Processes:

- **Time-Based Iteration**:
    - The simulation runs over a defined time frame, with each millisecond acting as a discrete time slot. This approach enables precise monitoring of rapid status changes and helps simulate realistic delays and access patterns.
- **Collision Detection and Response**:
    - When multiple stations simultaneously attempt to use the channel, a collision occurs. The affected station's transmission is interrupted, and each colliding station initiates the backoff process.
- **Successful Transmission**:
    - If only one station attempts access while the channel is idle, it secures the channel. The transmission completes after a time based on frame size and channel bandwidth, after which the channel is released.

---

4. Design Considerations and Methodology:

- **Fine-Grained 1ms Time Slot:**
    - By using a 1-millisecond slot duration, the simulation can capture high-frequency events, including rapid channel status changes and propagation delays, allowing a closer approximation of real-world conditions.
- **p-Persistence Model for Randomized Access**:
    - By integrating the probability parameter p, each station's attempt to transmit is probabilistic, simulating a more realistic network where stations may not transmit the moment the channel appears free.
- **Exponential Backoff Strategy**:

    - The backoff mechanism models how real CSMA/CD networks adjust waiting times after collisions, doubling each station’s wait time with each successive collision. This method helps spread out retries, minimizing the chances of repeated collisions.
- **Real-Time Event Logging**:
    - Events are both printed in real-time and logged to a file. This output structure supports both on-the-spot monitoring and subsequent analysis, providing insights into how the protocol manages collisions and channel occupation.

---

5. Parameterization and Flexibility:

- **Adjustable Parameters**:
    - This simulation allows easy customization of variables like **bandwidth**, **Tp**, **frame size**, and **p**. This flexibility supports a wide range of test scenarios, making the simulation highly adaptable for observing performance under various network conditions.
- **Scalability for Real-World Applications**:
    - With parameter adjustments, the simulation can mimic environments of varying density and activity, from sparse to highly active networks, highlighting the protocol's scalability.

Commentary on the Simulation and Potential Extensions

This p-persistent CSMA/CD simulation serves as an insightful model of core network behavior in a multi-station communication system, specifically demonstrating collision management, controlled channel access, and adaptive backoff techniques. Although it omits certain real-world complexities, such as noise interference, advanced error correction, and dynamic user arrival, the model still captures the fundamental dynamics of the CSMA/CD protocol.

The project underscores how probabilistic access (p-persistence) and exponential backoff combine to facilitate fair and efficient channel sharing. The detailed structure of the Channel and Station classes, paired with a discrete time-slot system, provides a foundation that can easily be expanded to include real-world elements like noise simulation, adaptive error handling, and even dynamic code assignments. These enhancements would create a more robust model, bringing the simulation closer to the conditions and challenges found in practical network environments.

Ultimately, this project not only demonstrates the basics of p-persistent CSMA/CD but also emphasizes the importance of access control and collision handling in multi-user networks. With further extensions, it could evolve into a comprehensive tool for analyzing CDMA protocols in modern communication networks, offering valuable insights into real-world applications and optimizations.

Implementation

```
import random

class Channel:

    def __init__(self, bandwidth, propagation_delay):

        self.bandwidth = bandwidth

        self.propagation_delay = propagation_delay

        self.is_busy = False

        self.active_station = None

        self.busy_timer = 0

        self.finish_timer = 0




class Station:
    def __init__(self, station_id, frame_size, channel,
persistence_probability):

        self.station_id = station_id

        self.frame_size = frame_size

        self.channel = channel

        self.persistence_probability = persistence_probability

        self.transmission_ready = True

        self.backoff_timer = 0

        self.recovery_timer = 0

        self.backoff_stage = 0

        self.is_jamming = False

        self.transmission_attempts = 0

        self.interruptions = 0

        self.dropped_frames = 0

        self.is_transmitting = False


    def calculate_backoff(self):

        if self.backoff_stage < 15:

            self.backoff_stage += 1

            self.backoff_timer = random.randint(0, 2 **
self.backoff_stage - 1)

        else:

            self.backoff_stage = 0

            self.recovery_timer = 5

            self.dropped_frames += 1
```

```python
def check_persistence(self):

    if self.is_transmitting:

        return False

    elif self.recovery_timer > 0:

        self.recovery_timer -= 1

        return False

    elif self.backoff_timer == 0 and not self.channel.is_busy:

        return random.random() < self.persistence_probability

    elif self.backoff_timer > 0:

        self.backoff_timer -= 1

        return False

    else:

        self.calculate_backoff()


def interrupt_transmission(self):

    self.channel.busy_timer = 0

    self.channel.active_station = None

    self.is_jamming = True

    self.is_transmitting = False

    self.calculate_backoff()

    self.interruptions += 1
```

```
def simulate_transmission(stations, channel, duration):
    time_steps = int(duration * 1000)

    with open('log.txt', 'w') as log_file:
        while time_steps > 0:
            jamming_detected = False
            for station in stations.values():
                if station.is_jamming:
                    jamming_detected = True
                    log_file.write(f"Jamming detected by Station {station.station_id}\n")
                    station.is_jamming = False
                    break

            if jamming_detected:
                time_steps -= 1
                continue

            if not channel.is_busy:
                manage_channel_timers(channel)
                ready_stations = [s for s in stations.values() if s.check_persistence()]

                if channel.active_station is None and len(ready_stations) == 1:
                    initiate_transmission(channel, 
ready_stations[0], log_file)

                elif ready_stations:

                    handle_collision(channel, ready_stations, 
stations, log_file)

                else:

                    log_file.write(".\n")

            else:

                handle_active_transmission(channel, stations, 
log_file)



            time_steps -= 1




def manage_channel_timers(channel):

    if channel.busy_timer > 0:

        channel.busy_timer -= 1

    if channel.finish_timer > 0:

        channel.finish_timer -= 1

    if channel.busy_timer == 0 and channel.active_station:

        channel.is_busy = True




def initiate_transmission(channel, station, log_file):

    channel.busy_timer = channel.propagation_delay
```

```python
    channel.finish_timer = station.frame_size // channel.bandwidth

    channel.active_station = station.station_id

    station.is_transmitting = True

    log_file.write(f"Station {station.station_id} began
transmission\n")



def handle_collision(channel, ready_stations, stations, log_file):

    if channel.active_station:

        log_file.write(f"Station {channel.active_station} was
interrupted. ")

        stations[channel.active_station].interrupt_transmission()

    log_file.write("Stations performing backoff: ")

    for station in ready_stations:

        station.calculate_backoff()

        log_file.write(f"{station.station_id} ")

    log_file.write("\n")



def handle_active_transmission(channel, stations, log_file):

    if channel.finish_timer > 0:

        channel.finish_timer -= 1

    if channel.finish_timer == 0:

        complete_transmission(channel, stations, log_file)

    else:
        log_file.write(".\n")



def complete_transmission(channel, stations, log_file):

    stations[channel.active_station].transmission_attempts += 1

    stations[channel.active_station].is_transmitting = False

    stations[channel.active_station].recovery_timer = 1

    channel.active_station = None

    channel.is_busy = False

    log_file.write("Transmission completed. Channel is now
free\n")



# Channel and Stations Setup

network_channel = Channel(bandwidth=1000, propagation_delay=5)

persistence_prob = 0.33

network_stations = {

    "A": Station(station_id="A", frame_size=12000,
channel=network_channel,
persistence_probability=persistence_prob),

    "B": Station(station_id="B", frame_size=12000,
channel=network_channel,
persistence_probability=persistence_prob),

    "C": Station(station_id="C", frame_size=12000,
channel=network_channel, persistence_probability=persistence_prob)

}

# Simulation Data Collection
```

```
times = []

average_transmissions = []


for i in range(1, 150, 1):

    current_time = i / 10

    simulate_transmission(network_stations, network_channel,
duration=current_time)

    avg_count = sum(station.transmission_attempts for station in
network_stations.values())

    times.append(current_time)

    average_transmissions.append(avg_count / (current_time * 3))
```

Comparison graphs and analysis

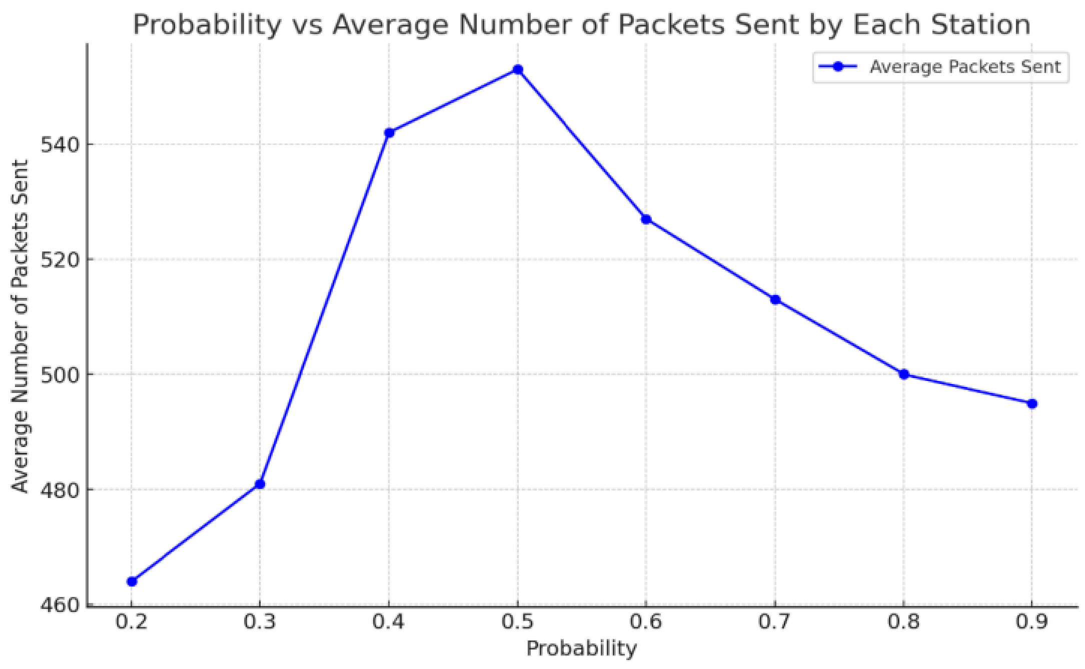

Fig 2: Probability vs Average number of packets sent by each station. Number of stations=5. Time=50 seconds. Time Slot=1ms. Frame size=12kb. Bandwidth=1kbps.

Probability vs. Average Number of Packets Sent by Each Station

In this analysis, we examine the impact of transmission probability on the average number of packets sent by each station, observing three main trends:

1. **Low Probabilities:** When transmission probabilities are low, fewer stations attempt to send packets at the same time. As a result, the channel is underutilised, and the average number of packets sent is lower. This configuration reduces the likelihood of collisions and retransmissions, but it also limits overall network throughput.
2. **Optimal Probability Range:** As the probability of transmission increases, the channel begins to see more activity, leading to a rise in the average packets sent by each station. In this range, the network reaches optimal utilization: more packets are successfully transmitted without an excessive increase in collisions. This range is ideal, achieving an effective balance between throughput and collision management.
3. **High Probabilities:** When the probability of transmission is very high, the network becomes saturated, and many stations attempt to send packets simultaneously, increasing the collision rate. Consequently, stations experience more retransmissions and delays, leading to a decline in the effective packet transmission rate.

This trend emphasizes the importance of setting an optimal transmission probability to maximize throughput while minimizing collisions.

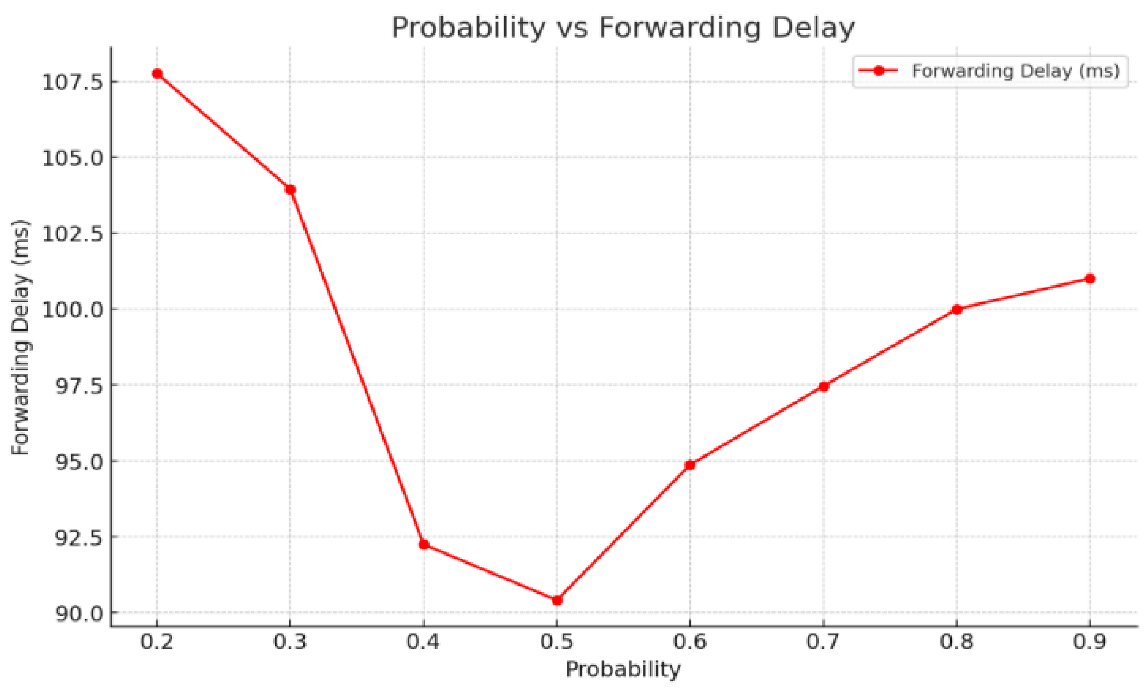

Fig 3: Probability vs Average Forwarding Delay. Number of stations=5. Time=50 seconds. Time Slot=1ms. Frame size=12kb. Bandwidth=1kbps.

Probability vs. Average Forwarding Delay

This analysis explores the effect of transmission probability on forwarding delay, the time taken for a packet to successfully complete its transmission from one station to another.

1. **Low to Moderate Probabilities:** Initially, as transmission probability increases, there's a gradual increase in forwarding delay. With lower traffic, fewer collisions occur, resulting in shorter waiting times for successful packet transmissions.
2. **Moderate to High Probabilities:** At higher probabilities, more frequent transmission attempts increase the collision likelihood. This leads to a greater number of retransmissions and elevates the forwarding delay, as packets need to wait longer for an available transmission slot.
3. **Very High Probabilities:** With extremely high probabilities, the forwarding delay peaks. The network becomes saturated, collisions are rampant, and each packet experiences significant delay before being transmitted successfully.

This plot illustrates that forwarding delay remains manageable up to a point, after which it sharply increases with transmission probability due to heightened collisions. This highlights an optimal probability range for efficient network performance.

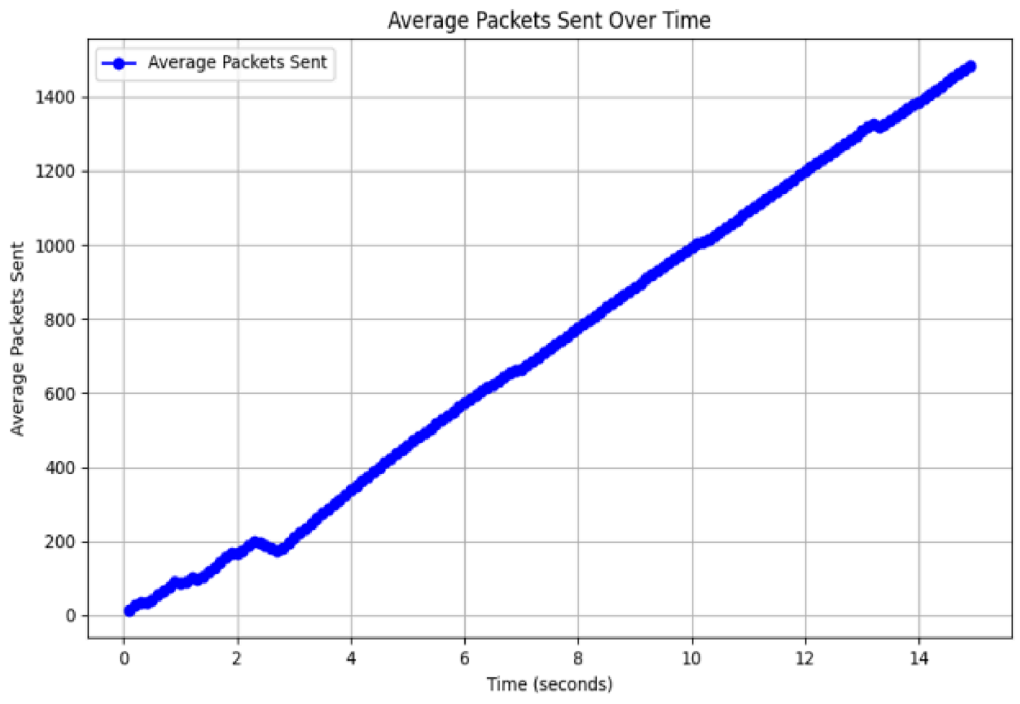

Fig 4: Time vs Average Packets sent by each station per second. Number of stations=3. P=0.33. Time Slot=1ms. Frame size=12kb. Bandwidth=1kbps.

Time vs. Average Packets Sent per Station per Second

This analysis presents the average number of packets each station sends per second over time, showing network performance as more packets are transmitted.

1. **Linear Growth Pattern:** The plot demonstrates a near-linear increase in average packets sent over time, suggesting that the network manages the load effectively, with each station consistently achieving successful packet transmissions. The linear rise indicates balanced throughput, where each station can transmit packets steadily without overwhelming the network.

2. **Absence of Saturation:** The lack of plateaus or drops in the graph indicates minimal congestion or collision, even as time progresses. This stability suggests the network is configured with an optimal transmission probability, allowing stations to avoid frequent retransmissions and maintain smooth data flow.
3. **Sustained Throughput:** The steady increase in packet transmissions indicates each station's ability to consistently utilize available channel capacity. This shows that stations experience minimal interference, ensuring a continuous, high-throughput performance over time.

Comments

This CSMA/CD (Carrier Sense Multiple Access with Collision Detection) simulation assignment offers valuable insights into the behavior of a shared-channel network under a probabilistic transmission protocol. By simulating CSMA/CD, we can observe how different transmission probabilities affect critical metrics, including the average number of packets sent, forwarding delay, and throughput. The results demonstrate the network's response to varying probabilities, revealing key insights into the efficiency and stability of the protocol.

Through this simulation, we deepen our understanding of core networking principles, particularly regarding collision management and the trade-offs involved in protocol configuration for shared-channel environments. By adjusting key parameters, such as transmission probability, we can achieve an optimized network that balances high throughput with minimal delays. Overall, this assignment underscores the importance of effective network design in maintaining efficient and reliable communication on shared channels.