Experiment No: 15	
Name	Suyash Tambe
PRN	22070126117
Date of Performance	18st October 2024
Title	To implement Leaky Bucket traffic Shaping Algorithm
Theory (short)	The Leaky Bucket method is a traffic shaping mechanism in computer networks that regulates the rate at which data packets are sent. The primary purpose of traffic shaping is to reduce network congestion by managing data flow. **Concept:** The program assumes a bucket to store incoming data packets. These packets arrive at different rates, yet they are sent out (or "leak") from the bucket at a consistent pace. If the bucket is full when a new packet comes, it is discarded, mimicking a network that discards surplus traffic to prevent overloading. **Components:** 1. **Bucket size**: Represents the maximum capacity of the bucket (i.e., how many packets can be stored at any time). 2. **Leak rate**: The rate at which the bucket releases packets (usually at a constant, predefined rate). 3. **Incoming packets**: Data arrives into the bucket at irregular intervals or in bursts, simulating real network traffic. **Operation:**

	• When packets arrive, they are placed in the bucket if there is space. • The bucket leaks packets at a steady rate, ensuring the network does not get overloaded. • If the bucket is full and new packets arrive, those packets are discarded to prevent exceeding the system's capacity.
Procedure	**Step 1: Initialization** • **Set the bucket size** BBB (the maximum number of packets the bucket can hold). • **Set the leak rate** LLL (the number of packets sent from the bucket per unit of time). • **Initialize current fill level** of the bucket to 0 (i.e., the bucket is initially empty). • **Set last checked time** to the current time. **Step 2: Packet Arrival**

- When a new packet arrives, check the time elapsed since the last leak was processed.
- Calculate the number of packets that have leaked during this time period based on the leak rate.
- Subtract the leaked packets from the current fill level (ensure the fill level doesn't go below zero).
- Update the current time as the last checked time.

Step 3: Check for Space in the Bucket

- If the size of the incoming packet is larger than the available space in the bucket (i.e., if the packet causes the bucket to overflow), **discard the packet**.
- Otherwise, add the packet to the bucket by increasing the current fill level.

Step 4: Leak Packets

- Packets are "leaked" from the bucket at a constant rate. This simulates the steady transmission of data, independent of the input packet rate.
- The bucket leaks out a fixed number of packets based on the leak rate and the time elapsed since the last leak check.

Step 5: Repeat for Each Incoming Packet

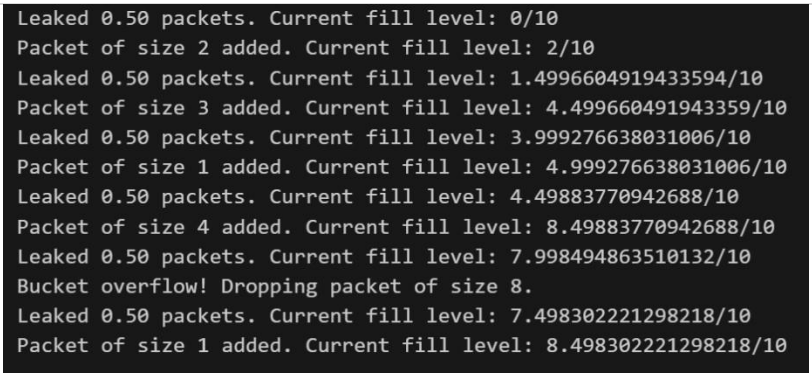
- For every packet that arrives, repeat steps 2 through 4 to decide whether to add the packet to the bucket or discard it based on the available space and leak rate.

Step 6: Handle Overflow

- If the bucket is full and cannot accommodate a new packet, the packet is discarded, which mimics real network behavior where excess traffic is dropped to prevent congestion.

Step 7: Monitor and Adjust

	• Continuously monitor the bucket's fill level to ensure that the incoming traffic is within acceptable limits. • Adjust the bucket size or leak rate based on traffic patterns or network requirements, if necessary.

Output Screenshots	 **Fig 1- Leaky Bucket Algorithm in action**

```python
import tim
class LeakyBucket:
    def __init__(self, bucket_size, leak_rate):
        self.bucket_size = bucket_size
        self.leak_rate = leak_rate
        self.current_fill = 0
        self.last_check = time.time()

    def add_packet(self, packet_size):
        self._leak()
        if packet_size > self.bucket_size:
            print(f"Packet of size {packet_size} is too large to fit in the bucket.")
            return False
        if self.current_fill + packet_size > self.bucket_size:
            print(f"Bucket overflow! Dropping packet of size {packet_size}.")
            return False
        self.current_fill += packet_size
        print(f"Packet of size {packet_size} added. Current fill level: {self.current_fill}/{self.bucket_size}")
        return True
    def _leak(self):
        now = time.time()
        time_elapsed = now - self.last_check
        leaked_amount = time_elapsed * self.leak_rate
        if leaked_amount > 0:
            self.current_fill = max(0, self.current_fill - leaked_amount)
            self.last_check = now
            print(f"Leaked {leaked_amount:.2f} packets. Current fill level: {self.current_fill}/{self.bucket_size}")
# Example usage:
bucket_size = 10  # Bucket can hold 10 packets
leak_rate = 1     # Leaking at a rate of 1 packet per second
bucket = LeakyBucket(bucket_size, leak_rate)
# Simulate incoming packets
packets = [2, 3, 1, 4, 8, 1]  # List of packet sizes
for packet in packets:
    time.sleep(0.5)  # Wait for half a second between packets
    bucket.add_packet(packet)
```

Fig 2- Code for Leaky Bucket Algorithm

Observation	1. **Rate Control**: The algorithm controls the output data flow rate to ensure a constant transmission speed, regardless of how irregular or bursty the incoming traffic is. 2. **Packet Dropping**: When the incoming traffic exceeds the bucket's capacity, excess packets are discarded to prevent congestion, simulating real network behavior. 3. **Traffic Shaping**: The algorithm smooths out traffic bursts by controlling how data is released into the network, ensuring a steady flow rather than a bursty one. 4. **Bucket Overflow**: If incoming packets arrive faster than the rate at which they can be leaked, the bucket can fill up quickly, leading to dropped packets, showing the importance of balancing input rates with the leak rate. 5. **Efficiency**: The algorithm works efficiently for managing and regulating varying traffic loads, allowing a network to handle traffic spikes without overwhelming the system. 6. **Parameter Sensitivity**: Performance depends heavily on the choice of bucket size and leak rate–larger buckets allow handling more bursty traffic, while a higher leak rate allows faster transmission.
Self-assessment Q&A	NA

Conclusion	The Leaky Bucket Algorithm is a useful traffic shaping tool in network systems that controls data flow and prevents congestion. By maintaining a constant rate of packet transmission and rejecting surplus data as needed, the method smoothes out bursty traffic patterns and assures more consistent network performance. It is critical for controlling network resources, ensuring that data flows within set bounds, and preventing network devices from being overloaded. Its simplicity and efficacy make it suitable for a wide range of networking situations, notably rate limitation and congestion management.