

Experiment No: 15	
Name	Vaibhav Sharma
PRN	22070126125
Date of Performance	16 st October 2024
Title	To implement Leaky Bucket traffic Shaping Algorithm
Theory (short)	<p>The Leaky Bucket algorithm is a traffic shaping mechanism used in computer networks to control the rate at which data packets are transmitted. The main goal of traffic shaping is to prevent network congestion by regulating the data flow.</p> <p>Concept:</p> <p>The algorithm envisions a bucket that holds incoming data packets. These packets arrive at varying rates, but they are sent out (or "leak") from the bucket at a constant rate. If the bucket is full when a new packet arrives, the packet is dropped, simulating a network that discards excess traffic to prevent overloading.</p> <p>Components:</p> <ol style="list-style-type: none"> 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. <p>Operation:</p> <ul style="list-style-type: none"> • 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	<p>Step 1: Initialization</p> <ul style="list-style-type: none"> • 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. <p>Step 2: Packet Arrival</p>

- 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.

```
import time

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:
```

```

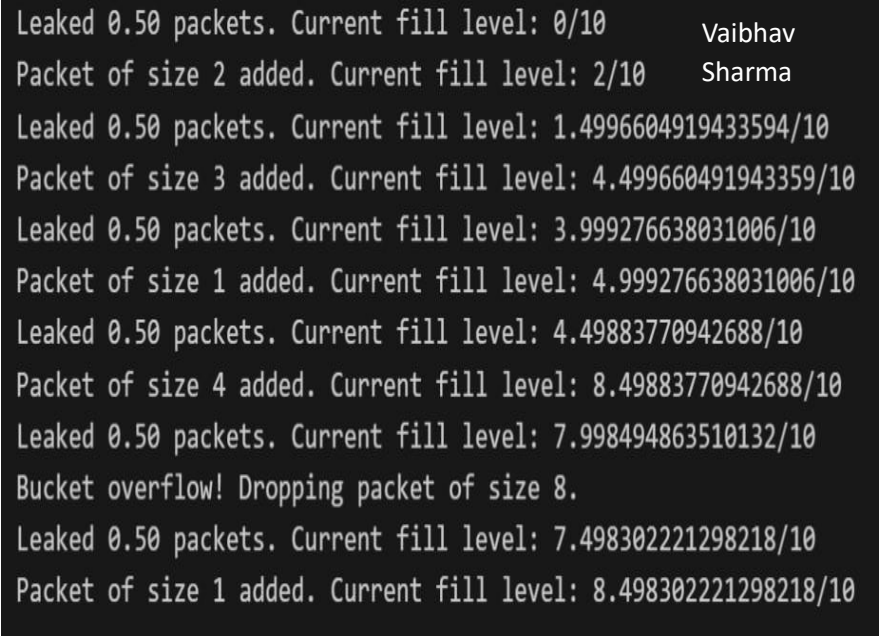
        print(f"Packet of size {packet_size} is too large to
fit in the bucket.")
        return False
    if self.current_fill + packet_size > self.bucket_size:
        print(f"Bucket overflow! Dropping packet of size
{packet_size}.")
        return False
    self.current_fill += packet_size
    print(f"Packet 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, 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)

```

Output Screenshots	
---------------------------	--

Observation	<ol style="list-style-type: none"> 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.
--------------------	--

<p>Self-assessment Q&A</p>	<p>Q: What is the primary purpose of the Leaky Bucket algorithm in networking?</p> <p>Ans: The Leaky Bucket algorithm controls the data flow rate to prevent network congestion by releasing packets at a constant rate.</p> <p>Q: What happens when the bucket is full and new packets arrive in the Leaky Bucket algorithm?</p> <p>Ans: When the bucket is full, any new packets are dropped to prevent overloading the network.</p> <p>Q: How does the leak rate affect the data transmission in the Leaky Bucket algorithm?</p> <p>Ans: The leak rate controls how quickly packets are transmitted, ensuring a steady, regulated flow of data, regardless of how irregularly packets arrive.</p>
<p>Conclusion</p>	<p>The Leaky Bucket Algorithm is an effective traffic shaping mechanism used in network systems to control data flow and prevent congestion. By maintaining a steady rate of packet transmission and discarding excess traffic when necessary, the algorithm smooths out bursty traffic patterns and ensures a more consistent network performance. It plays a crucial role in managing network resources, ensuring that data flows within predefined limits, and helping to prevent overloading of network devices. Its simplicity and effectiveness make it widely applicable in various networking environments, particularly for rate limiting and congestion control.</p>