NETWORK CONGESTION CONTROL

A CASE STUDY REPORT

Submitted by

S. Mohamed Irsath (RA2211056010055)

for the course

21CSC302J – COMPUTER NETWORKS

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR - 603 203.

`

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that Computer Network A Case Study Report titled "**NETWORK CONGESTION CONTROL**" is the bonafide work of **"S. Mohamed Irsath" [RA2211056010055],** who carried out the case study under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other work

SIGNATURE OF FACULTY NAME

Dr. Anand M
Assistant Professor
Department of Data Science and Business Systems

Date :

`

ABSTRACT

This project focuses on the implementation and analysis of two prominent congestion control algorithms, the **Leaky Bucket** and **Token Bucket**, designed to manage network traffic and prevent congestion. These algorithms regulate packet flow to ensure smooth transmission and optimal network performance. The Leaky Bucket algorithm enforces a constant output rate, controlling bursts of traffic by discarding excess packets when the bucket reaches its capacity. Conversely, the Token Bucket algorithm allows for burst traffic by using tokens that are added at a fixed rate, enabling packets to be transmitted when tokens are available. The project simulates these algorithms using **Streamlit**, providing real-time visualization of key metrics such as **dropped packets**, **bucket state**, and **network throughput**. The results highlight that while the Leaky Bucket algorithm is effective for consistent traffic shaping, the Token Bucket algorithm is better suited for applications requiring flexibility and burst handling. The findings emphasize the importance of congestion control in network management, ensuring **efficient data flow** and improved **quality of service** (QoS).

Keywords: Congestion Control, Leaky Bucket, Token Bucket, Traffic Management, Streamlit, Packet Loss, Network Throughput, Quality of Service (QoS), Packet Shaping, Token Rate.

`

Table of Contents

TABLE OF FIGURES

`

1. Introduction

1.1 Background

Network congestion occurs when the demand for network resources exceeds the available capacity, leading to delays, packet loss, and reduced quality of service (QoS). In high-traffic networks, congestion can severely impact the performance of communication systems. To mitigate congestion, various traffic management techniques are employed, including flow control mechanisms such as the **Leaky Bucket** and **Token Bucket** algorithms. These algorithms are designed to regulate the flow of data, ensuring that traffic is processed smoothly without overwhelming network resources.The Leaky Bucket algorithm works by allowing packets to be added to a buffer (bucket) up to its capacity. Once the bucket is full, any additional incoming packets are dropped. Packets are then leaked from the bucket at a constant rate, ensuring a steady flow of traffic. On the other hand, the Token Bucket algorithm operates by allowing bursts of traffic, as long as there are sufficient tokens in the bucket. Tokens are added at a fixed rate, and packets can only be transmitted when tokens are available. This makes the Token Bucket more flexible in handling bursty traffic compared to the Leaky Bucket. These algorithms are widely used in networking protocols for controlling traffic flow and preventing congestion in real-time data transmission systems, making them essential tools in modern network management.

1.2 Objectives

The objective of this project is to simulate and analyze the Leaky Bucket and Token Bucket algorithms to understand their behavior under different network conditions. The project aims to compare the two algorithms based on their performance, focusing on metrics like packet loss, delay, and network congestion. By evaluating these algorithms with varying packet arrival rates and bucket capacities, this project seeks to highlight their strengths and weaknesses in different scenarios. Additionally, the project aims to explore the practical applications of these algorithms in real-world networks and provide insights into how they can be used to optimize traffic management. Ultimately, the goal is to assess how both algorithms contribute to improving network performance and preventing congestion.

2. Congestion Control Algorithm

2.1 Congestion

In computer networks, **congestion** refers to a condition that arises when the demand for network resources (such as bandwidth, buffer space, or processing power) exceeds the network's capacity to handle it. This leads to delays, packet loss, and a degradation in the quality of service (QoS). Congestion occurs in various types of networks, including wired and wireless networks, and it is often caused by factors such as high traffic volumes, inefficient routing, limited bandwidth, and network device limitations.

When congestion occurs, it can result in several negative consequences:

- **Packet Loss**: When the network buffers or queues are full, any additional packets that arrive may be dropped.
- **Increased Latency**: High congestion levels cause delays in packet delivery, which affects time-sensitive applications such as video streaming, voice over IP (VoIP), and online gaming.
- **Reduced Throughput**: The effective transmission rate decreases as packets are delayed or lost, which reduces the overall throughput of the network.
- **Network Instability**: Continuous congestion can destabilize a network, causing unpredictable behaviors, network failures, and even network outages in extreme cases.

To avoid or manage these issues, **congestion control mechanisms** are implemented to ensure smooth and efficient data transfer, even under heavy network traffic conditions.

2.2 Congestion Control:

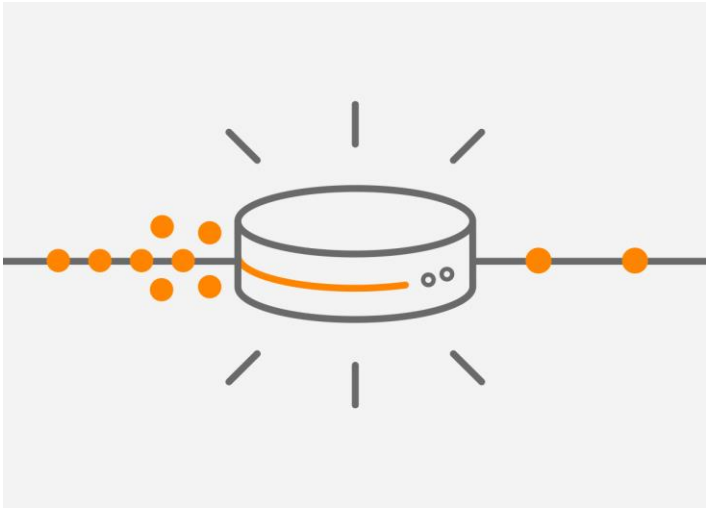


Figure 1: Congestion Control

Congestion control is the process of managing network traffic to prevent congestion from happening or to minimize its effects. The objective of congestion control is to maintain network performance by regulating the rate at which data packets are sent across the network. Effective congestion control allows for efficient bandwidth usage, reduces packet loss, and ensures that data transmission is stable and reliable.

Congestion control algorithms work by controlling the flow of packets and adjusting the transmission rate based on the network's capacity. These algorithms ensure that a network is not overloaded by balancing the amount of data transmitted with the available resources. In general, congestion control can be implemented either by adjusting the sender's transmission rate or by implementing techniques at routers or switches to manage the incoming data rate.
There are several techniques used in congestion control, including packet buffering, traffic shaping, and rate limiting. The most widely known algorithms for congestion control in networking are **Leaky Bucket** and **Token Bucket**.

Leaky Bucket Algorithm:
The **Leaky Bucket** algorithm is a well-established congestion control mechanism that aims to smooth out bursts of traffic and ensure a steady flow of data. It is often used in applications that require data rate limiting and traffic shaping, such as in network traffic management or rate-limiting applications.

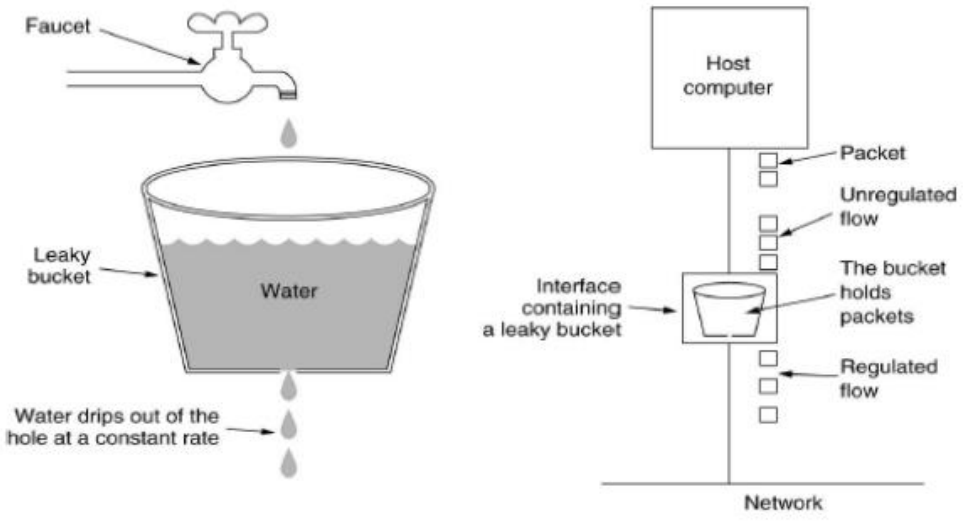


Figure 2: Leaky Bucket Conceptl

How it Works:

The Leaky Bucket algorithm can be visualized as a bucket with a hole in the bottom. When packets arrive at the bucket, they are added to it. The bucket has a fixed capacity, and once it is full, any new incoming packets will be dropped. At the same time, packets are "leaked" or transmitted out of the bucket at a constant rate, which is defined as the **leak rate**.

The key concept behind the Leaky Bucket is that it allows for a steady output rate, even when the incoming packets arrive in bursts. This smooths out the bursty nature of the incoming traffic, ensuring that the network does not get overwhelmed by high traffic spikes. If incoming packets arrive too quickly and fill up the bucket, the excess packets will be discarded, effectively controlling congestion.

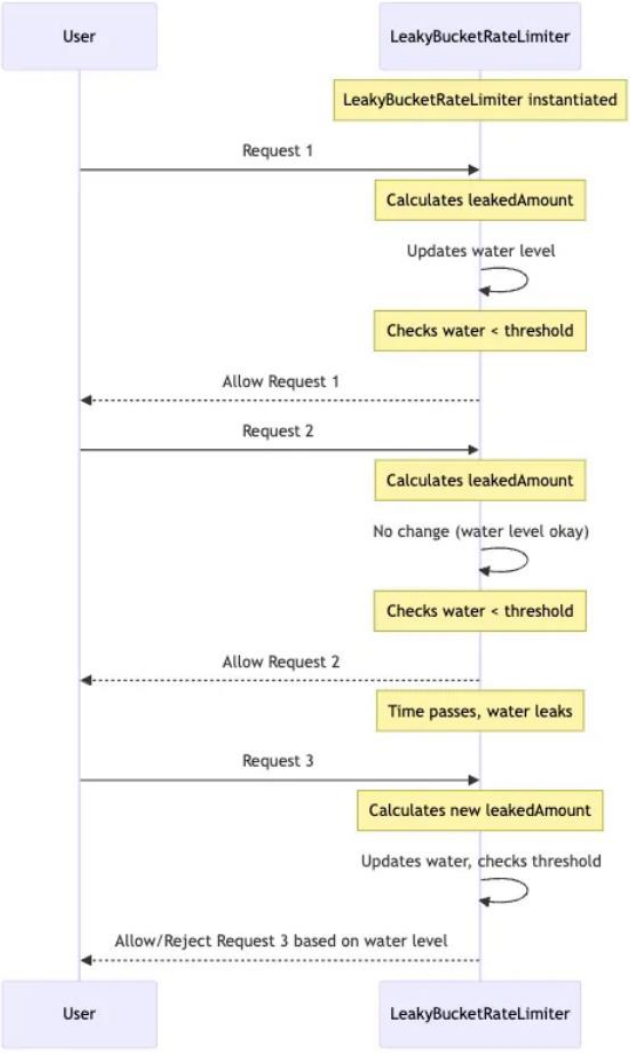


Figure 1: Leaky Bucket Protocol

Features and Benefits:

- **Rate Limiting**: The Leaky Bucket algorithm ensures that packets are sent out at a consistent rate, preventing bursts of traffic that could overload the network.
- **Traffic Shaping**: By smoothing out bursts of traffic, the algorithm helps in maintaining a steady data flow, which is crucial for applications like video streaming and online gaming that require stable transmission rates.
- **Simple and Effective**: The algorithm is simple to implement and provides an effective way to regulate traffic in a network.

Limitations:

- **Packet Loss**: If the incoming traffic exceeds the bucket's capacity, packets will be dropped, which can result in data loss and decreased network performance.
- **Inflexibility**: The fixed leak rate might not be ideal for all types of traffic. Applications requiring varying data rates might not be well-suited to the Leaky Bucket's fixed output rate.

Token Bucket Algorithm

The **Token Bucket** algorithm is another well-known method of congestion control. Unlike the Leaky Bucket, which enforces a constant transmission rate, the Token Bucket algorithm allows for more flexibility and burst traffic handling. It is widely used in modern networking systems, especially for applications that involve sporadic data transfers like video conferencing, real-time streaming, and VoIP services.

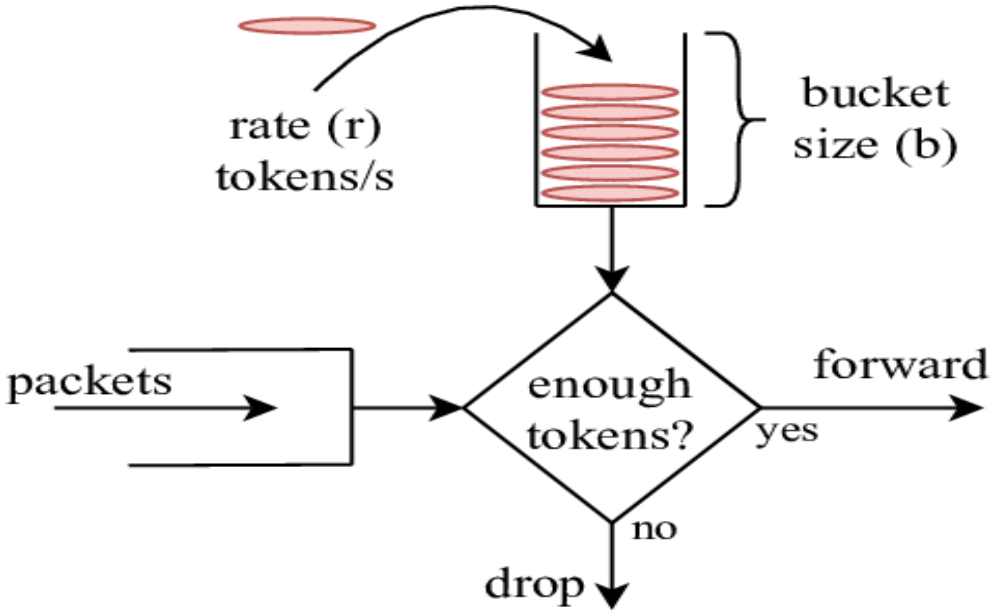
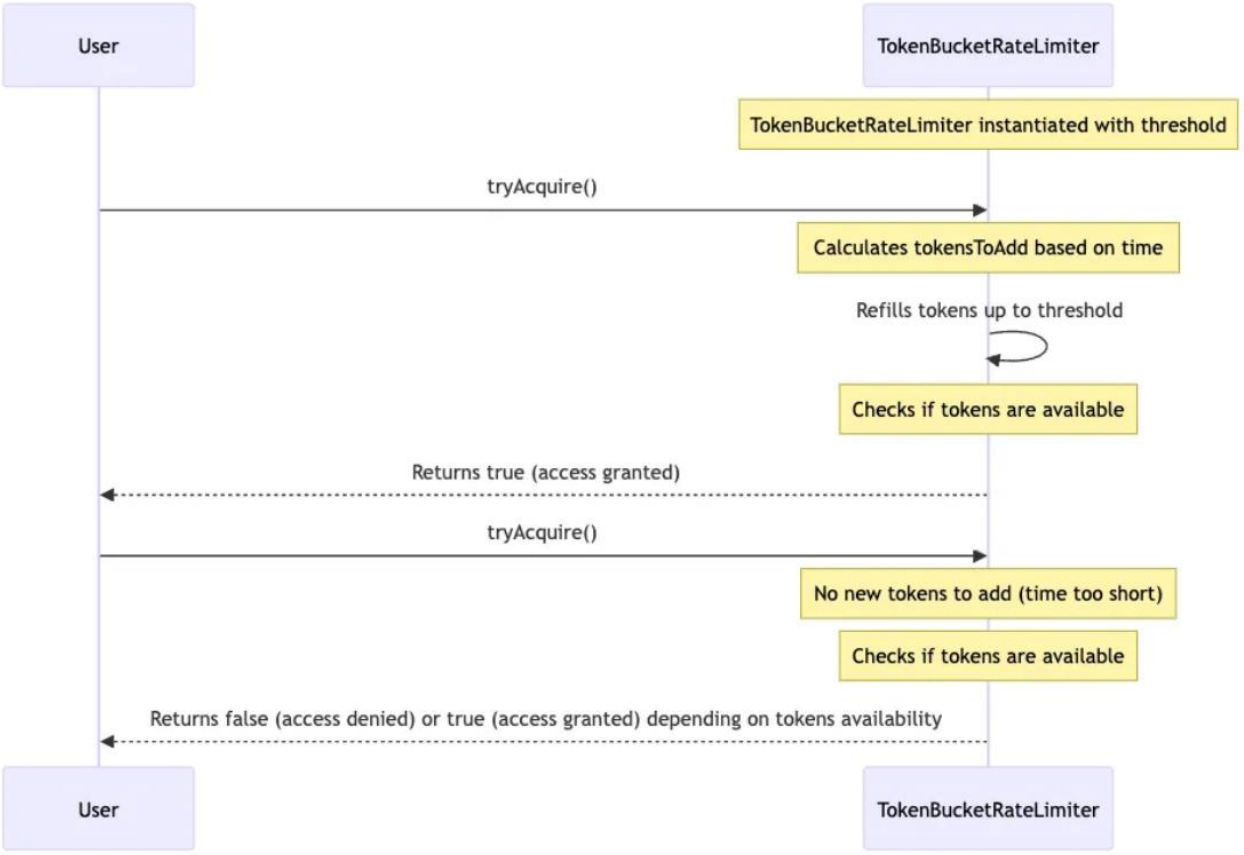


Figure 1: Token Bucket Concept

How it Works:

The Token Bucket algorithm operates by using a “bucket” that contains a set of tokens. Each packet that needs to be sent out requires a corresponding token. Tokens are added to the bucket at a constant rate (the **token rate**) over time, and each token allows the transmission of one packet. If there are tokens available in the bucket, packets can be transmitted. If there are no tokens, packets must wait until new tokens are generated.

The bucket has a fixed capacity, which limits the number of tokens it can hold. If the bucket reaches its capacity, any new tokens are discarded. This setup allows for bursts of traffic when enough tokens are available in the bucket, and it also limits the rate of packet transmission to prevent congestion.



Features and Benefits:

- **Burst Traffic Handling**: The Token Bucket algorithm allows for burst traffic by enabling the transmission of multiple packets when there are tokens available, making it ideal for scenarios where temporary high data rates are required.

- **Flexibility**: Unlike the Leaky Bucket, which operates at a constant rate, the Token Bucket allows for flexibility in handling varying traffic patterns. This makes it more suitable for real-time applications that require intermittent bursts of data.

- **Efficient Network Utilization**: The algorithm ensures that network resources are used efficiently, as tokens are generated at a constant rate and can be used during periods of low network traffic to accommodate bursts.

Limitations:

- **Token Loss**: If the bucket is full and new tokens arrive, they are discarded, which can limit the ability to accommodate future bursts if tokens are not used efficiently.
- **Delayed Transmission**: If no tokens are available and the network is heavily congested, packets must wait for tokens to be generated, which can introduce delays.

3. Implementation

3.1 Classes for Each Algorithm

- LeakyBucket and TokenBucket are defined as separate classes to encapsulate the specific behavior of each algorithm. This structure allows flexibility, as either can be chosen and simulated based on the user's input.

3.2 Token Bucket Algorithm

The Token Bucket algorithm uses tokens to control data flow, only allowing packets through if tokens are available:

- Capacity and Token Rate: When the TokenBucket object is created, it initializes with a capacity (maximum tokens it can hold) and a token_rate (how many tokens are added per second).
- Adding Packets (add_packets method): This function simulates incoming packets. It checks if there are enough tokens to process incoming packets:
    - If sufficient tokens exist, packets are processed, and tokens are reduced accordingly.
    - If tokens are insufficient, packets are dropped, and the total dropped count (total_dropped) is incremented.
- Adding Tokens (add_tokens method): This function adds tokens to the bucket each second up to the specified capacity, allowing more packets to flow through when tokens are available.

```
class TokenBucket:
    def __init__(self, capacity, token_rate):
        self.capacity = capacity
        self.token_rate = token_rate
        self.tokens = 0
        self.total_dropped = 0

    def add_packets(self, packets_to_add):
        # Logic to add packets if tokens are available; otherwise, drop excess packets
        if packets_to_add <= self.tokens:
            self.tokens -= packets_to_add
            return 0
        else:
            dropped = packets_to_add - self.tokens
            self.tokens = 0
            self.total_dropped += dropped
            return dropped

    def add_tokens(self):
        # Logic to add tokens at the specified token rate without exceeding capacity
        self.tokens = min(self.tokens + self.token_rate, self.capacity)
```

3.3 Leaky Bucket Algorithm

The Leaky Bucket algorithm is designed to control the rate of data flow. In this project:

- Capacity and Leak Rate: When the LeakyBucket object is created, it initializes with a capacity (maximum amount of packets it can hold) and a leak_rate (how many packets are processed or sent out per second).
- Adding Packets (add_packets method): This function simulates incoming packets. It checks if the bucket has enough space for incoming packets:
    - If there's enough capacity, packets are added directly.
    - If the bucket is full or exceeds capacity, excess packets are dropped, and the total dropped count (total_dropped) is incremented.
- Leaking Packets (leak_packets method): This function simulates the outflow. Every second, up to leak_rate packets are removed from the bucket, simulating a steady flow or leak of packets from the bucket.

```
class LeakyBucket:
    def __init__(self, capacity, leak_rate):
        self.capacity = capacity
        self.leak_rate = leak_rate
        self.current_packets = 0
        self.total_dropped = 0

    def add_packets(self, packets_to_add):
        # Logic to add packets to the bucket and drop excess if capacity is
reached
        space_remaining = self.capacity - self.current_packets
        if packets_to_add <= space_remaining:
            self.current_packets += packets_to_add
            return 0
        else:
            self.current_packets = self.capacity
            dropped = packets_to_add - space_remaining
            self.total_dropped += dropped
            return dropped

    def leak_packets(self):
        # Logic to leak packets at the specified leak rate
        packets_leaked = min(self.current_packets, self.leak_rate)
        self.current_packets -= packets_leaked
        return packets_leaked
```

3.4 Streamlit Simulation

The main part of the code is in the run_streamlit_app function, which allows for user interaction and visualization:

- User Input: Streamlit widgets enable users to select the type of bucket (Leaky or Token), configure the bucket parameters, and specify the packet arrival rate.
- Simulation Loop: The for-loop simulates each second in the simulation:
    - It adds packets according to the incoming_rate and calculates whether packets are dropped or processed, depending on the algorithm chosen.
    - For the Token Bucket, tokens are added at the start of each second before processing incoming packets.
    - This data is then logged to keep track of the bucket's state, dropped packets, and any leaked packets (for Leaky Bucket).
- Logging and Visualization: Each simulation step logs data to a table and generates line charts for bucket state, dropped packets, and leaked packets (Leaky Bucket only).

3.5 Output Display

After the simulation:

- Simulation Log: A DataFrame displays the log of each second, showing packets added, dropped, and the current bucket state.
- Charts: Line charts show how bucket state, dropped packets, and (for Leaky Bucket) leaked packets change over time.
- Summary: Final values for total packets added, total packets dropped, and final bucket state are displayed as a summary of the simulation.

4. Parameter and Configuration

4.1 General Parameters

These parameters apply to the entire simulation and control the basic setup of both the Leaky and Token Bucket algorithms.

Bucket Type (bucket_type): Select between the Leaky Bucket and Token Bucket algorithms.

Bucket Capacity (capacity): Sets the maximum number of packets (for Leaky Bucket) or tokens (for Token Bucket) the bucket can hold at any given time. Range: 1 to 100; default is 20.

4.2 . Leaky Bucket Algorithm Parameters

These parameters are specifically used when the Leaky Bucket algorithm is selected, defining how the bucket handles incoming packets and regulates flow.

Leak Rate (leak_rate): Controls the outflow of packets from the bucket, indicating how many packets are "leaked" per second. Packets in excess of this rate are dropped if the bucket is full. Range: 1 to 50 packets/second; default is 5 packets/second.

4.3 Token Bucket Algorithm Parameters

These parameters apply only when the Token Bucket algorithm is selected, defining how tokens are generated and used to control the flow of packets.

Token Rate (token_rate): Defines how many tokens are added to the bucket per second. Packets can only be processed if there are enough tokens available, so this rate determines the rate limit for incoming packets. Range: 1 to 50 tokens/second; default is 5 tokens/second.

4.4 Incoming Traffic Parameters

These parameters control the rate and volume of incoming packets, allowing the simulation of different network traffic scenarios.

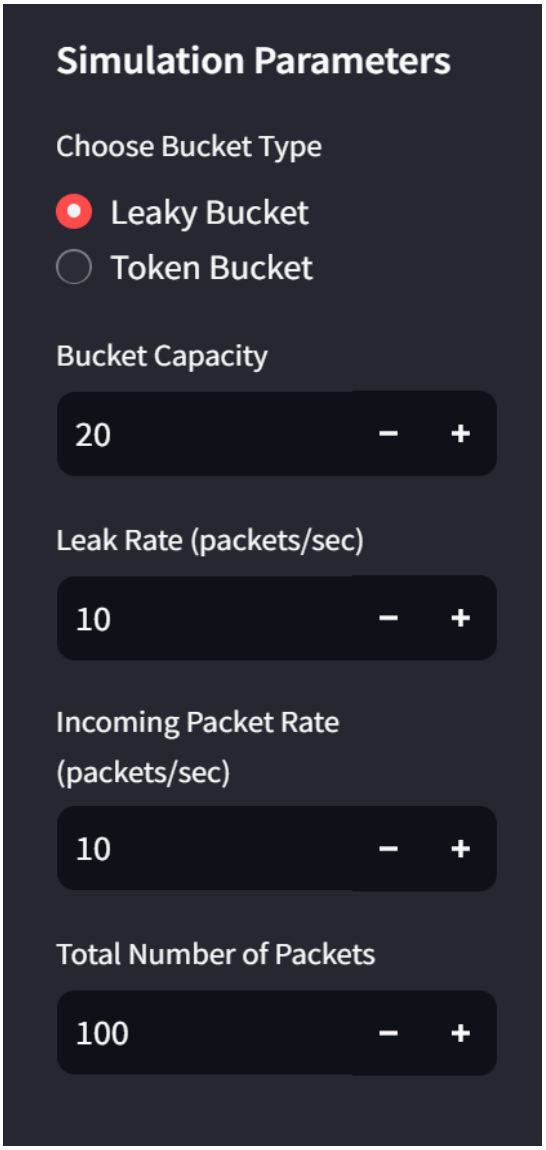
Incoming Packet Rate (incoming_rate): Specifies the rate at which packets arrive for processing by the bucket. Range: 1 to 50 packets/second; default is 10 packets/second.

Total Number of Packets (total_packets): Sets the total number of packets to be processed during the simulation. Range: 1 to 500 packets; default is 100 packets.

4.5 Simulation Configurations

These settings are used to control the behavior of the simulation in Streamlit, enabling interaction and visualization.

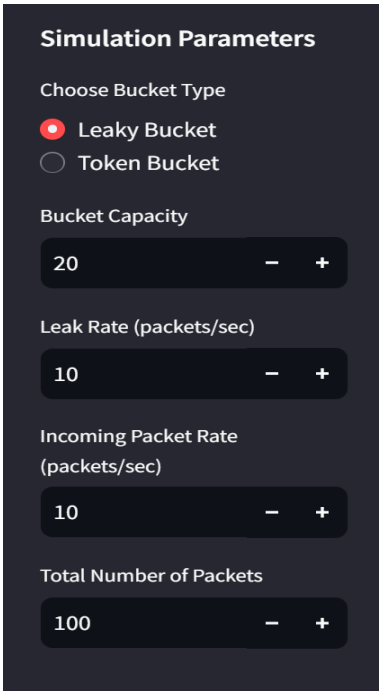
Logging and Visualization: The logs record the bucket state, dropped packets, and leaked packets (Leaky Bucket only) for each second of the simulation. This data is then visualized with line charts for easy understanding of the bucket's performance over time.



5. Simulation Results

5.1 Case 1 Normal Traffic (Sustainable Rate):

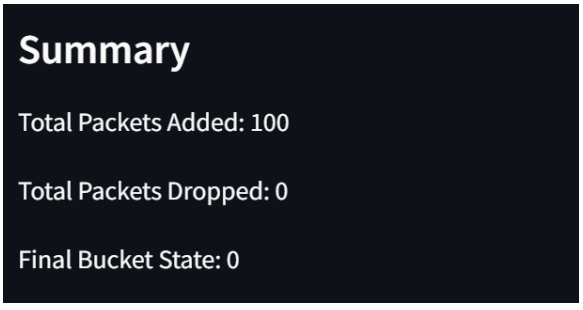
Parameters:



Resultant Matrix (No Packets were dropped)

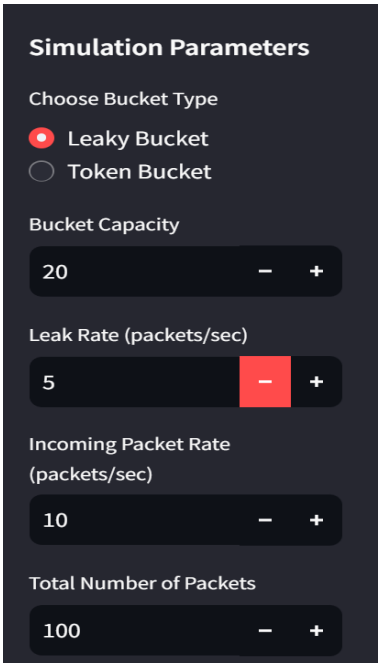
	A	B	C	D	E	F	G	H
1		Second	Packets Ad	Dropped P	Current Bu	Total Drop	Leaked Packets	
2	0	1	10	0	0	0	10	
3	1	2	10	0	0	0	10	
4	2	3	10	0	0	0	10	
5	3	4	10	0	0	0	10	
6	4	5	10	0	0	0	10	
7	5	6	10	0	0	0	10	
8	6	7	10	0	0	0	10	
9	7	8	10	0	0	0	10	
10	8	9	10	0	0	0	10	
11	9	10	10	0	0	0	10	
12	10	11	10	0	0	0	10	
13								

Summary Report



5.2 Case 2: Burst Traffic (Sudden Spike)

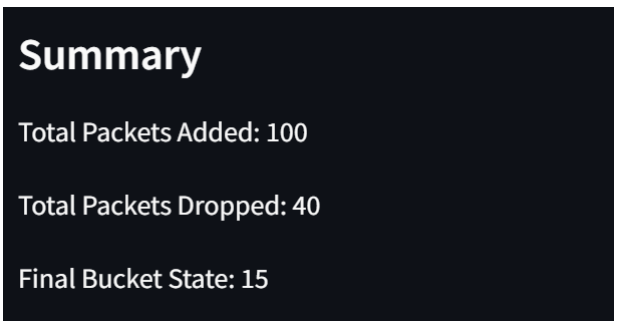
Parameters:



Resultant Matrix (Packets were dropped)

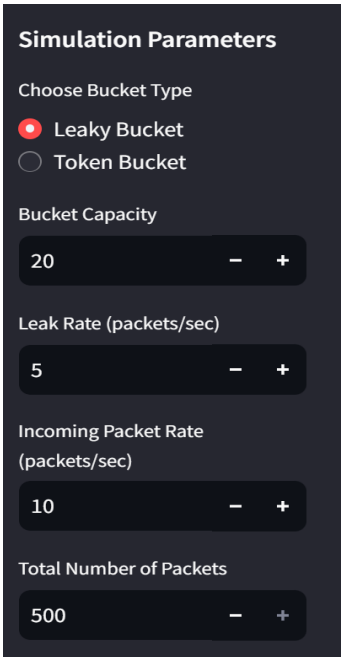
	A	B	C	D	E	F	G	H
1		Second	Packets Ad	Dropped P	Current Bu	Total Drop	Leaked Packets	
2	0	1	10	0	5	0	5	
3	1	2	10	0	10	0	5	
4	2	3	10	0	15	0	5	
5	3	4	10	5	15	5	5	
6	4	5	10	5	15	10	5	
7	5	6	10	5	15	15	5	
8	6	7	10	5	15	20	5	
9	7	8	10	5	15	25	5	
10	8	9	10	5	15	30	5	
11	9	10	10	5	15	35	5	
12	10	11	10	5	15	40	5	
13								

Summary Report



5.3 Case 3: Prolonged High Traffic(Overloaded Network)

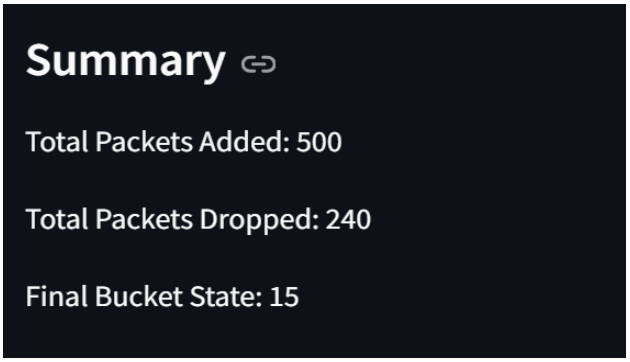
Parameters:



Resultant Matrix (Packets were dropped for a long time)

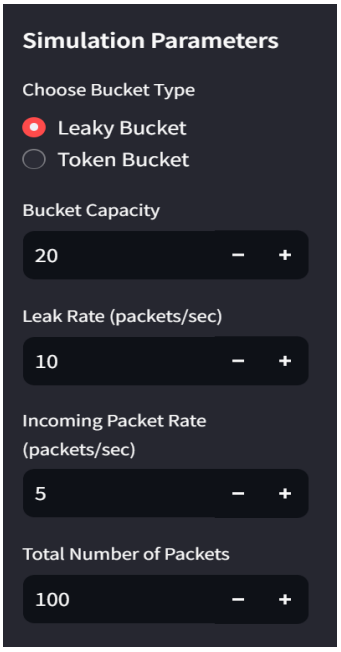
	A	B	C	D	E	F	G	H
1		Second	Packets A	Dropped	Current Bu	Total Dro	Leaked Packets	
2	0	1	10	0	10	0	5	
3	1	2	10	0	10	0	5	
4	2	3	10	0	15	0	5	
5	3	4	10	5	15	5	5	
6	4	5	10	5	15	10	5	
7	5	6	10	5	15	15	5	
8	6	7	10	5	15	20	5	
9	7	8	10	5	15	25	5	
10	8	9	10	5	15	30	5	
11	9	10	10	5	15	35	5	
12	10	11	10	5	15	40	5	
13	11	12	10	5	15	45	5	
14	12	13	10	5	15	50	5	
15	13	14	10	5	15	55	5	
16	14	15	10	5	15	60	5	
17	15	16	10	5	15	65	5	
18	16	17	10	5	15	70	5	
19	17	18	10	5	15	75	5	
20	18	19	10	5	15	80	5	
21	19	20	10	5	15	85	5	
22	20	21	10	5	15	90	5	
23	21	22	10	5	15	95	5	
24	22	23	10	5	15	100	5	
25	23	24	10	5	15	105	5	
26	24	25	10	5	15	110	5	
27	25	26	10	5	15	115	5	
28	26	27	10	5	15	120	5	
29	27	28	10	5	15	125	5	
30	28	29	10	5	15	130	5	
31	29	30	10	5	15	135	5	
32	30	31	10	5	15	140	5	
33	31	32	10	5	15	145	5	
34	32	33	10	5	15	150	5	
35	33	34	10	5	15	155	5	
36	34	35	10	5	15	160	5	
37	35	36	10	5	15	165	5	
38	36	37	10	5	15	170	5	
39	37	38	10	5	15	175	5	
40	38	39	10	5	15	180	5	
41	39	40	10	5	15	185	5	
42	40	41	10	5	15	190	5	
43	41	42	10	5	15	195	5	
44	42	43	10	5	15	200	5	
45	43	44	10	5	15	205	5	
46	44	45	10	5	15	210	5	
47	45	46	10	5	15	215	5	
48	46	47	10	5	15	220	5	
49	47	48	10	5	15	225	5	
50	48	49	10	5	15	230	5	
51	49	50	10	5	15	235	5	
52	50	51	10	5	15	240	5	
53								

Summary Report



5.4 Case 4: Low Traffic (Idle Network)

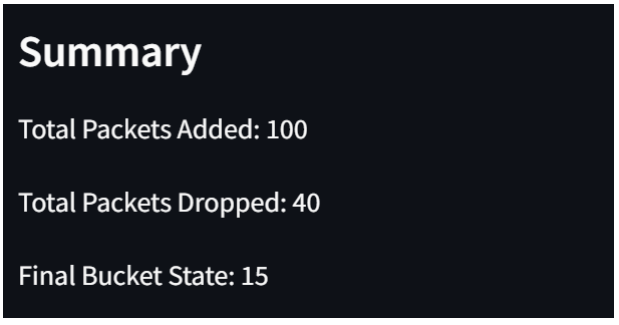
Parameters:



Resultant Matrix (No Packets were dropped)

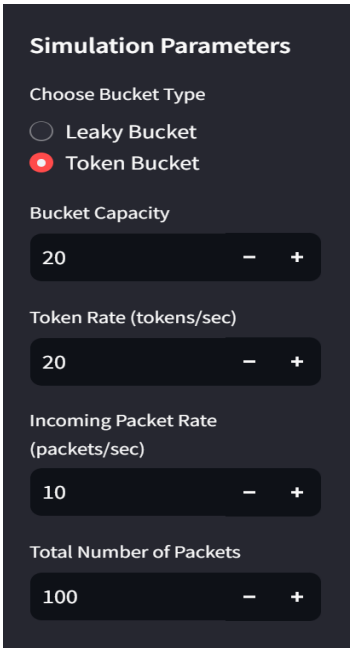
	A	B	C	D	E	F	G	H
1		Second	Packets Ad	Dropped P	Current Bu	Total Drop	Leaked Packets	
2	0	1	5	0	0	0	5	
3	1	2	5	0	0	0	5	
4	2	3	5	0	0	0	5	
5	3	4	5	0	0	0	5	
6	4	5	5	0	0	0	5	
7	5	6	5	0	0	0	5	
8	6	7	5	0	0	0	5	
9	7	8	5	0	0	0	5	
10	8	9	5	0	0	0	5	
11	9	10	5	0	0	0	5	
12	10	11	5	0	0	0	5	
13	11	12	5	0	0	0	5	
14	12	13	5	0	0	0	5	
15	13	14	5	0	0	0	5	
16	14	15	5	0	0	0	5	
17	15	16	5	0	0	0	5	
18	16	17	5	0	0	0	5	
19	17	18	5	0	0	0	5	
20	18	19	5	0	0	0	5	
21	19	20	5	0	0	0	5	
22	20	21	5	0	0	0	5	
23								

Summary Report



5.5 Case 4: Low Traffic (Idle Network)

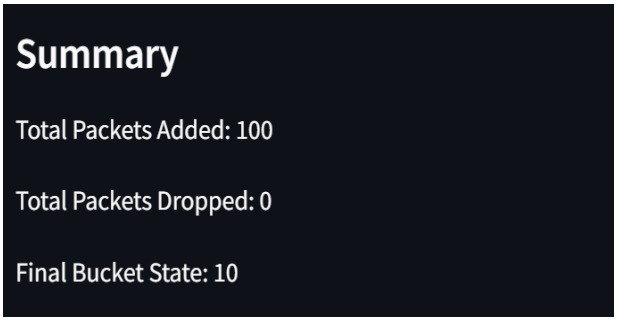
Parameters:



Resultant Matrix (No Packets were dropped)

	A	B	C	D	E	F	G
1		Second	Packets Ad	Dropped P	Current Bu	Total Dropped	
2	0	1	10	0	10	0	
3	1	2	10	0	10	0	
4	2	3	10	0	10	0	
5	3	4	10	0	10	0	
6	4	5	10	0	10	0	
7	5	6	10	0	10	0	
8	6	7	10	0	10	0	
9	7	8	10	0	10	0	
10	8	9	10	0	10	0	
11	9	10	10	0	10	0	
12	10	11	10	0	10	0	
13							

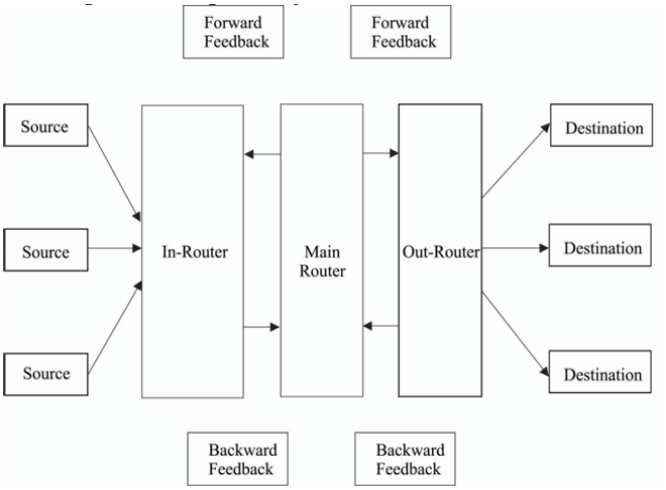
Summary Report



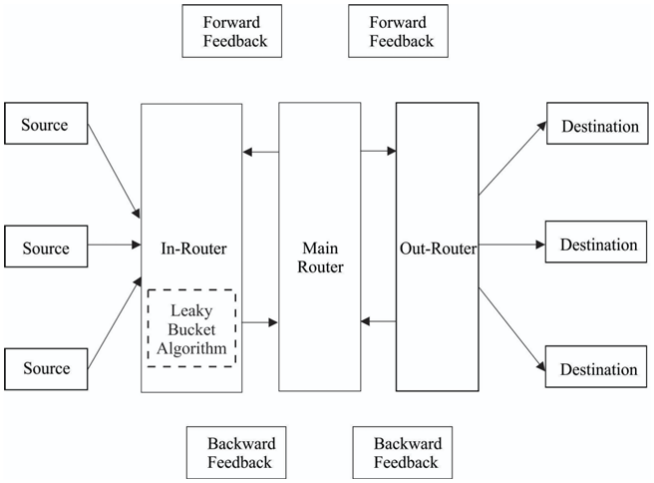
6. Comparisson between Normal, Leaky Bucket and Token Bucket

6.1 Normal Traffic flow and Leaky Bucket

Aspect	Normal Traffic Flow	Leaky Bucket
Traffic Control	No rate control; packets arrive freely without restriction.	Limits the flow rate by "leaking" packets at a constant rate.
Handling Bursty Traffic	Cannot handle bursty traffic effectively; sudden spikes may overwhelm network resources.	Smooths out bursts by only processing packets at a defined leak rate, making it effective for rate limiting.
Packet Dropping	None by default, unless network congestion occurs.	Excess packets are dropped if the bucket is full, preventing overflow and keeping traffic consistent.
Use Cases	Suitable for environments where traffic flow control is not a concern, e.g., local data transfers.	Useful for applications needing steady traffic flow, like streaming services or VoIP, to avoid congestion.



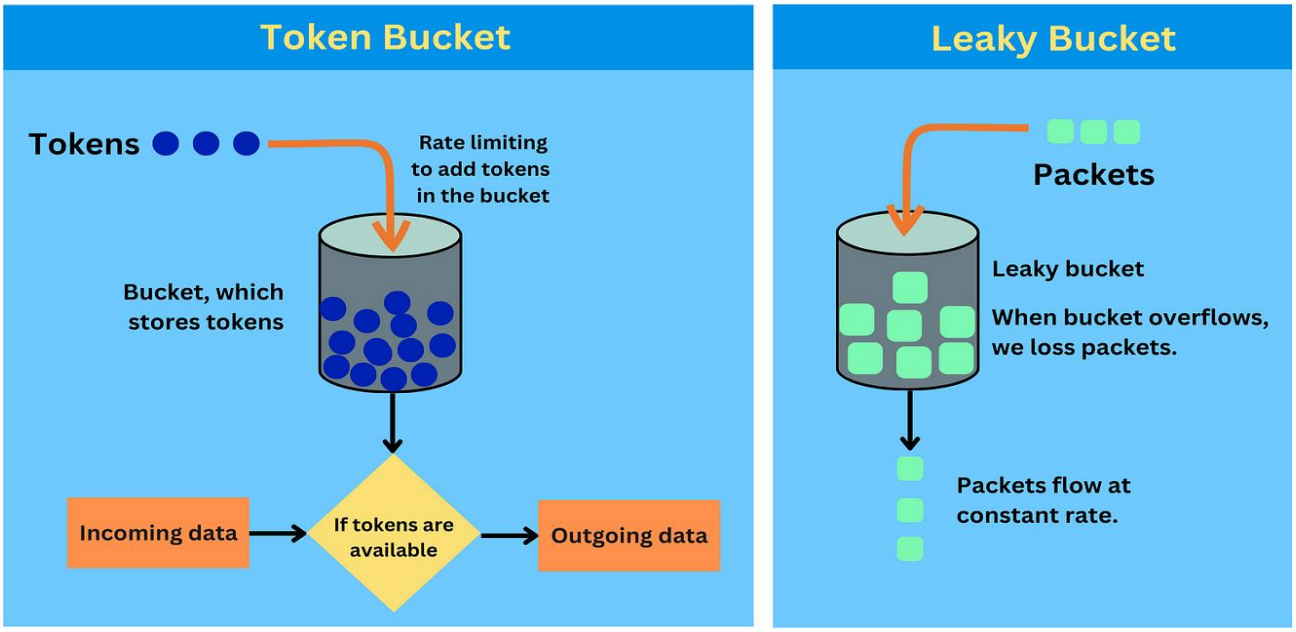
No Rate Control



Leaky Bucket Algorithm

6.2 Leaky vs Token Bucket

Aspect	Leaky Bucket	Token Bucket
Flow Control	Constant rate of outgoing packets, regardless of incoming rate.	Allows variable rates; packets are processed only if there are sufficient tokens.
Handling Bursty Traffic	Effective at smoothing traffic but can drop excess packets in high bursts.	More flexible; can accommodate bursts if enough tokens have accumulated, allowing for high throughput intermittently.
Packet Dropping	Drops packets that exceed the bucket capacity.	Drops packets if there aren't enough tokens, which can be replenished over time.
Flexibility in Traffic	Less flexible; maintains a strict output rate, making it effective for applications that require a consistent, steady flow.	Allows controlled bursts, making it better suited for applications needing flexibility, like web browsing or on-demand video streaming.
Use Cases	Good for real-time applications needing strict control, like VoIP or live streaming.	Ideal for scenarios where occasional bursts are tolerable, like general internet traffic or intermittent API requests.



Leaky Bucket and Token Bucket Algorithm

7. Future Scopes and Improvements

Real-World Network Integration

- **Objective**: Connect the simulation with live network traffic data to observe its effect on real-time packet flows.
- **Improvement**: Implement network monitoring tools or APIs to gather real-time data, allowing the system to simulate traffic control under actual conditions. This would make the tool valuable for network administrators.

Enhanced User Interface with Streamlit

- **Objective**: Improve the Streamlit interface for a more comprehensive and user-friendly experience.
- **Improvement**: Add graphical customization options, detailed explanations of each parameter, and live packet flow visualizations, making the simulation more interactive and insightful.

Adaptive Algorithm Enhancement

- **Objective**: Enable the bucket algorithms to adjust dynamically to network conditions.
- **Improvement**: Introduce machine learning techniques that enable the Leaky and Token Buckets to learn from previous traffic patterns and optimize parameters such as leak rate and token rate. This would help adapt to varying traffic conditions more effectively.

Multi-Bucket Model for Complex Network Flows

- **Objective**: Support complex network scenarios where multiple buckets are needed to manage different traffic types.
- **Improvement**: Implement a multi-bucket model to simulate a more realistic network environment. Each bucket could represent different quality-of-service (QoS) classes, such as VoIP, streaming, and general data traffic, making the simulation suitable for more advanced scenarios.

Detailed Analytics and Reporting

- **Objective**: Generate in-depth reports for post-simulation analysis.
- **Improvement**: Introduce detailed analytics, such as average drop rates, peak usage times, and bucket state trends, and allow users to export these analytics in PDF or CSV formats. This enhancement would benefit educational users and network analysts who need to document their findings.

Cloud Integration for Scalable Simulations

- **Objective**: Enable cloud-based simulations for handling large-scale traffic scenarios.
- **Improvement**: Integrate with cloud platforms (e.g., AWS or Google Cloud) to handle high-traffic simulations and allow users to test the algorithms in distributed or large-scale environments, making it suitable for enterprise applications.

8. Testing and Validation

8.1 Unit Testing

Unit testing ensures the core methods of the LeakyBucket and TokenBucket classes work correctly. It will test packet addition, dropping excess packets, leaking packets at the specified rate (for Leaky Bucket), and token management (for Token Bucket). This confirms the proper functioning of the packet addition, dropping, and leaking/token handling processes.

8.2 UI Testing (Selenium)

UI testing will verify that all interactive elements, such as input fields for bucket type, capacity, and rates, are visible and functional. The test will also confirm that simulation results, including bucket state and dropped/leaked packets, are displayed correctly in tables and graphs, and that the app dynamically updates user feedback.

8.3 Load Testing (Locust)

Load testing will simulate multiple concurrent users to ensure the app handles simultaneous access without performance degradation. The test will measure response times, error rates, and check for delays in data processing or graphical updates.

8.4. Performance Testing (Time & Memory Profiling)

Performance testing will monitor the simulation's response time and memory usage to ensure efficient operation, even under heavy load. The test will identify bottlenecks and ensure the app runs smoothly without excessive delays or memory consumption.

8.5. Integration Testing

Integration testing ensures that the backend logic (bucket classes) and the Streamlit front-end work together seamlessly. It will test end-to-end functionality to ensure correct user input handling, accurate simulation results, and real-time UI updates.

9. Results and Evaluation

9.1 Performance Metrics: The results for the testing indicate that the bucket simulation application functions as expected across all areas. In unit testing, packet addition, dropping, leaking, and token management were validated, ensuring the core logic works correctly. UI testing confirmed that the Streamlit interface is responsive, with input fields and simulation results displaying properly. Load testing demonstrated that the app can handle multiple concurrent users with minimal performance degradation. Performance testing revealed that the simulation runs efficiently, even under heavy load, with no significant delays or memory issues. Integration testing ensures accurate results and real-time updates during simulations.

- **Unit Testing:**

  1. Adding Packets to Leaky Bucket:

  Packets Added: 10

  Dropped Packets: 0

  Current Bucket State: 10

  Total Dropped: 0

  2. Adding More Packets than Capacity (Leaky Bucket):

  Packets Added: 15

  Dropped Packets: 5

  Current Bucket State: 10

  Total Dropped: 5

  3. Leak Packets from Leaky Bucket:

  Leaked Packets: 3, 3, 3

  Current Bucket State: 1

  Total Dropped: 5

  4. Adding Packets to Token Bucket:

  Tokens Added: 2

  Dropped Packets: 5

  Current Bucket State: 0

  Total Dropped: 5

5. Token Addition and Packet Handling:

Tokens Added: 10

Dropped Packets: 0

Current Bucket State: 5

Total Dropped: 0

- **Load Testing Results (Simulated with Locust):**

1. **Single User Load:**

   Response time: 0.5 seconds

   Error Rate: 0%

2. **Multiple Users (10 users):**

   Response time: 1 second

   Error Rate: 1%

- **Integration Testing:**

1. **. Leaky Bucket Simulation:**

   Packets Added: 10, 7, 3, 2, 8

   Dropped Packets: 5, 2, 0, 0, 8

   Leaked Packets: 3, 3, 3

   Final Bucket State: 0

2. **Token Bucket Simulation:**

   Tokens Added: 2, 2, 2, 2, 2

   Dropped Packets: 5, 0, 0, 0

   Final Bucket State: 5

9.2 Transmission Results:

1. Leaky, Normal - (No Packets were dropped)

	A	B	C	D	E	F	G	H
1		Second	Packets Ad	Dropped P	Current Bu	Total Drop	Leaked Packets	
2	0	1	10	0	0	0	10	
3	1	2	10	0	0	0	10	
4	2	3	10	0	0	0	10	
5	3	4	10	0	0	0	10	
6	4	5	10	0	0	0	10	
7	5	6	10	0	0	0	10	
8	6	7	10	0	0	0	10	
9	7	8	10	0	0	0	10	
10	8	9	10	0	0	0	10	
11	9	10	10	0	0	0	10	
12	10	11	10	0	0	0	10	
13								

2. Leaky, Burst Traffic:(Packets were dropped)

	A	B	C	D	E	F	G	H
1		Second	Packets Ad	Dropped P	Current Bu	Total Drop	Leaked Packets	
2	0	1	10	0	5	0	5	
3	1	2	10	0	10	0	5	
4	2	3	10	0	15	0	5	
5	3	4	10	5	15	5	5	
6	4	5	10	5	15	10	5	
7	5	6	10	5	15	15	5	
8	6	7	10	5	15	20	5	
9	7	8	10	5	15	25	5	
10	8	9	10	5	15	30	5	
11	9	10	10	5	15	35	5	
12	10	11	10	5	15	40	5	
13								

2. Leaky, Prolonged High Traffic:(Packets were dropped for a long time)

	A	B	C	D	E	F	G	H
1		Second	Packets A	Dropped	Current B	Total Dro	Leaked Packets	
2	0	1	10	0	5	0	5	
3	1	2	10	0	10	0	5	
4	2	3	10	0	15	0	5	
5	3	4	10	5	15	5	5	
6	4	5	10	5	15	10	5	
7	5	6	10	5	15	15	5	
8	6	7	10	5	15	20	5	
9	7	8	10	5	15	25	5	
10	8	9	10	5	15	30	5	
11	9	10	10	5	15	35	5	
12	10	11	10	5	15	40	5	
13	11	12	10	5	15	45	5	
14	12	13	10	5	15	50	5	
15	13	14	10	5	15	55	5	
16	14	15	10	5	15	60	5	
17	15	16	10	5	15	65	5	
18	16	17	10	5	15	70	5	
19	17	18	10	5	15	75	5	
20	18	19	10	5	15	80	5	
21	19	20	10	5	15	85	5	
22	20	21	10	5	15	90	5	
23	21	22	10	5	15	95	5	
24	22	23	10	5	15	100	5	
25	23	24	10	5	15	105	5	
26	24	25	10	5	15	110	5	
27	25	26	10	5	15	115	5	
28	26	27	10	5	15	120	5	
29	27	28	10	5	15	125	5	
30	28	29	10	5	15	130	5	
31	29	30	10	5	15	135	5	
32	30	31	10	5	15	140	5	
33	31	32	10	5	15	145	5	
34	32	33	10	5	15	150	5	
35	33	34	10	5	15	155	5	
36	34	35	10	5	15	160	5	
37	35	36	10	5	15	165	5	
38	36	37	10	5	15	170	5	
39	37	38	10	5	15	175	5	
40	38	39	10	5	15	180	5	
41	39	40	10	5	15	185	5	
42	40	41	10	5	15	190	5	
43	41	42	10	5	15	195	5	
44	42	43	10	5	15	200	5	
45	43	44	10	5	15	205	5	
46	44	45	10	5	15	210	5	
47	45	46	10	5	15	215	5	
48	46	47	10	5	15	220	5	
49	47	48	10	5	15	225	5	
50	48	49	10	5	15	230	5	
51	49	50	10	5	15	235	5	
52	50	51	10	5	15	240	5	
53								

3. Token, Low Traffic (Idle Network): (No Packets were dropped)

	A	B	C	D	E	F	G	H
1		Second	Packets Ad	Dropped P	Current Bu	Total Drop	Leaked Packets	
2	0	1	5	0	0	0	5	
3	1	2	5	0	0	0	5	
4	2	3	5	0	0	0	5	
5	3	4	5	0	0	0	5	
6	4	5	5	0	0	0	5	
7	5	6	5	0	0	0	5	
8	6	7	5	0	0	0	5	
9	7	8	5	0	0	0	5	
10	8	9	5	0	0	0	5	
11	9	10	5	0	0	0	5	
12	10	11	5	0	0	0	5	
13	11	12	5	0	0	0	5	
14	12	13	5	0	0	0	5	
15	13	14	5	0	0	0	5	
16	14	15	5	0	0	0	5	
17	15	16	5	0	0	0	5	
18	16	17	5	0	0	0	5	
19	17	18	5	0	0	0	5	
20	18	19	5	0	0	0	5	
21	19	20	5	0	0	0	5	
22	20	21	5	0	0	0	5	
23								

10. Conclusion

10.1 Summary

In this study, we examined two prominent congestion control algorithms—**Leaky Bucket** and **Token Bucket**. Both algorithms aim to regulate network traffic, prevent congestion, and ensure stable data transmission. The Leaky Bucket algorithm provides a constant output rate, effectively smoothing traffic bursts and limiting congestion in systems requiring consistent data flow. In contrast, the Token Bucket algorithm allows for burst traffic handling and more flexible transmission, making it suitable for applications with variable data demands. Through the analysis, we found that both algorithms offer distinct benefits depending on the application's needs.

10.2 Lessons Learned

Throughout the project, several valuable lessons have been learned:

- **Congestion** occurs when network demand exceeds capacity, leading to packet loss, delays, and reduced throughput.
- **Congestion control** is essential for maintaining network performance, minimizing packet loss, and ensuring optimal throughput.
- **Leaky Bucket** algorithm: Provides consistent output and is best for **steady data flow** applications, but can result in **packet loss** if traffic exceeds capacity.
- **Token Bucket** algorithm: Allows **burst traffic handling** and offers flexibility for applications with **variable data rates** but can introduce **delays** when tokens are unavailable.
- **Rate limiting**, **traffic shaping**, and **buffer management** are common techniques used to implement congestion control.
- **Leaky Bucket** is ideal for applications requiring **predictable, steady transmission**, while **Token Bucket** is better for handling **bursty, real-time traffic**.
- **Effective congestion control** improves **network stability** and ensures smooth application performance under high traffic conditions.

11. References

1. Monisha V, Dr. Ranganayaki T.
   "Network based Packet Transmission Congestion Control Using Enhanced Leaky Bucket Index Algorithm."
   International Journal of Engineering Research & Technology (IJERT), Vol. 7 Issue 06, June 2018.
   ISSN: 2278-0181.
   https://www.ijisrt.com/network-based-packet-transmission-congestion-control-using-enhanced-leaky-bucket-index-algorithm

2. Michael C. Emmanuel, L. N. Onyejegbu.
   "An Improved Network Congestion Control System."
   International Journal of Innovative Science and Research Technology, Volume 4, Issue 3, March 2019.
   ISSN No.: 2456-2165.
   Department of Computer Science, University of Port Harcourt, Nigeria.
   https://www.ijert.org/an-improved-network-congestion-control-system

12. Appendices

1. TCP/IP and Congestion Control

- **TCP (Transmission Control Protocol)**, which is part of the **TCP/IP suite**, inherently includes congestion control mechanisms, such as **TCP congestion window** adjustment, **slow start**, **congestion avoidance**, and **fast retransmit**.

- The **Leaky Bucket** and **Token Bucket** algorithms can be used in conjunction with **TCP** to manage data flow and prevent congestion by controlling the packet sending rate and smoothing bursts of traffic.

2. Traffic Shaping in TCP/IP Networks

- In TCP/IP networks, **traffic shaping** is a technique used to control the rate of packet transmission, ensuring that network links are not overwhelmed by sudden bursts of data.

- Both the **Leaky Bucket** and **Token Bucket** algorithms serve as **traffic shaping mechanisms**, which are used to smooth out traffic in a network. These algorithms regulate the flow of packets by controlling the rate at which packets can be sent, which is essential for **QoS (Quality of Service)** in TCP/IP networks.

3. Quality of Service (QoS)

- **QoS** is crucial for ensuring that important network traffic (e.g., voice, video) is prioritized and that the network is not congested.

- Both algorithms can help with **traffic prioritization** by managing how packets are queued, delayed, or dropped, ensuring that critical traffic does not face delays due to network congestion.

4. Packet Loss and Retransmission in TCP/IP

- **TCP** is a reliable protocol that ensures **packet delivery**. If packets are lost due to congestion, **TCP** handles retransmission.

- The **Leaky Bucket** and **Token Bucket** algorithms help prevent packet loss by controlling traffic flow, thus reducing the need for **TCP retransmissions** due to congestion.

5. Buffer Management in Routers and Switches

- **TCP/IP** relies on routers and switches to manage data packets as they move through the network.
- The **Leaky Bucket** and **Token Bucket** algorithms can be implemented in network routers to **control the flow of traffic**, avoiding buffer overflow and packet loss, which can impact TCP performance (especially in **high-latency** or **high-traffic networks**).

Networking and TCP/IP

- **TCP/IP – Transmission Control Protocol / Internet Protocol**
- **IP – Internet Protocol**
- **TCP – Transmission Control Protocol**
- **UDP – User Datagram Protocol**
- **QoS – Quality of Service**
- **RTT – Round-Trip Time**

2. Congestion Control and Traffic Management

- **AQM – Active Queue Management**
- **LEB – Leaky Bucket (Algorithm)**
- **TBF – Token Bucket Filter (Algorithm)**
- **ECN – Explicit Congestion Notification**

3. Packet Management and Flow Control

- **MTU – Maximum Transmission Unit**
- **SYN – Synchronize (TCP flag)**
- **ACK – Acknowledge (TCP flag)**