VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI, KARNATAKA, INDIA

NITTE EDUCATION TRUST | NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

An Autonomous Institution Approved by UGC, AICTE, and the Government of Karnataka. Accredited with NAAC 'A+' Grade. Affiliated to Visvesvaraya Technological University, Belagavi. Located at Yelahanka, Bengaluru – 560064, Karnataka, India.

Report On

Congestion Control Using Leaky Bucket Algorithm

BACHELOR OF ENGINEERING

In

ELECTRONICS AND COMMUNICATION ENGINEERING

2023-2024

Submitted By

Manav Manoj 1NT22EC091

YELAHANKA, BENGALURU- 560064

NITTE EDUCATION TRUST | NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

NITTE EDUCATION TRUST | NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

Department Electronics and Communication Engineering

Certificate

This is to certify that the Learning Activity (Experiential Learning) work entitled ***"Congestion Control using Leaky Bucket Algorithm"*** has been carried out by ***Manav Manoj (1NT22EC091)***, bonafide students of ***Nitte Meenakshi Institute of Technology***, in partial fulfillment of the requirements for the award of the degree of **Bachelor of Engineering** in the **Department of Electronics and Communication Engineering,** under **Visvesvaraya Technological University**, Belagavi, during the academic year **2024–2025**.

The report has been examined and approved as it meets the academic requirements specified under the autonomous scheme of **Nitte Meenakshi Institute of Technology** for the said degree

Signature of the Guide

Ms. Divya G
Associate Professor
Dept. of Electronics and Communication Engg
Nitte Meenakshi Institute of Technology,
Yelahanka, Bengaluru-560064.

Signature of the Guide

Ms. Prajna K
Assistant Professor
Dept. of Electronics and Communication Engg
Nitte Meenakshi Institute of Technology,
Yelahanka, Bengaluru-560064.

Acknowledgment

The satisfaction and the euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible. The constant guidance of these persons and encouragement provided, crowned our efforts with success and glory. Although it is not possible to thank all the members who helped for the completion of the seminar work individually, I take this opportunity to express my gratitude to one and all.

I am grateful to the management and our institute **Nitte Meenakshi Institute of Technology** with its very ideals and inspiration for having provided me with the facilities which made this work a success.

I express my sincere gratitude to the Principal, **Dr. Nagraju** for the support and encouragement.

I wish to place on record, my grateful thanks to **Dr. Parameshachari**, Head, Department of ECE, Nitte Meenakshi Institute of Technology, for the constant encouragement provided to me.

I am indebted with a deep sense of gratitude for the constant inspiration, encouragement, timely guidance and valid suggestion given to me by my guides **Ms. Divya G** and **Ms. Prajna**, Assistant Professors, Department of ECE, Nitte Meenakshi Institute of Technology.

I am thankful to all the staff members of the department for providing relevant information and helped in different capacities in carrying out this academic work.

Last, but not least, I owe my debts to my parents, friends and also those who directly or indirectly have helped me to make the seminar a success.

Abstract

In modern computer networks, effective congestion control is essential to maintaining Quality of Service (QoS) and preventing performance degradation. This report investigates the Leaky Bucket Algorithm, a fundamental traffic shaping mechanism, by integrating both theoretical analysis and practical implementation using object-oriented programming in C++.

The motivation stems from the growing demand for stable data transmission across various applications including real-time video, online gaming, and cloud services. The core research question centers on how effectively the Leaky Bucket Algorithm manages network congestion by controlling packet flow in high-load conditions.

The methodology involves simulating a leaky bucket mechanism where packets enter a buffer and exit at a fixed rate. The implementation models real-world scenarios by tracking packet acceptance, buffering, and dropping events. The simulation was developed using C++ to provide hands-on understanding of traffic regulation.

Results from the simulation demonstrate the algorithm's deterministic behavior in smoothing traffic flow and preventing buffer overflows. Packets exceeding the buffer capacity were appropriately dropped, confirming the algorithm's limitations in handling bursty traffic.

In conclusion, the Leaky Bucket Algorithm provides a predictable and efficient method for congestion control in static network conditions. While it lacks adaptability to dynamic traffic, the study serves as a foundational exploration of traffic shaping techniques, paving the way for future work on hybrid algorithms like Token Bucket and RED.

Table of Contents

1. Introduction

1.1 Motivation

In the ever-evolving world of computer networks, ensuring efficient and reliable data transmission has become more critical than ever. With the exponential rise in internet usage, including services such as real-time video streaming, online gaming, Internet of Things (IoT) communications, cloud computing, and virtual meetings, network infrastructures are under immense pressure. One of the primary challenges in such networks is congestion—a condition that arises when the volume of data packets exceeds the processing or forwarding capacity of routers and network devices.

Congestion can cause severe performance degradation including increased packet delays, jitter, retransmissions, and complete packet loss. These issues ultimately affect the quality of service (QoS) experienced by end users. Managing congestion effectively is vital to maintain optimal data flow and prevent system bottlenecks, especially in high-speed and high-traffic environments.

To mitigate congestion, several algorithms have been developed. Among these, the Leaky Bucket Algorithm offers a simplistic yet robust mechanism for traffic shaping and rate regulation. This algorithm controls the data flow into the network to prevent sudden bursts that could overwhelm network components. Its relevance is particularly significant in understanding how basic queue management can impact overall network performance.

1.2 Objective

This report aims to provide a deep technical insight into the Leaky Bucket Algorithm and demonstrate its functioning through an object-oriented C++ simulation. The objectives of this report are:

- To explain the concept of congestion in data networks and why it occurs.
- To outline traditional congestion control techniques and their roles.
- To introduce and analyze the Leaky Bucket Algorithm.
- To implement the algorithm in C++ using OOP principles, showcasing how real-world networking scenarios can be emulated.

- To evaluate the benefits and limitations of this method through simulation.

This report serves as an educational tool for computer science students, networking enthusiasts, and software developers, laying a strong foundation for exploring advanced congestion control techniques used in modern network protocols.

2. Congestion Control Fundamentals

2.1 What is Congestion in Networks?

Congestion in computer networks occurs when the demand for bandwidth exceeds the available capacity at one or more points in the system. This leads to excessive queuing, increased transmission delays, packet drops, and in extreme cases, a complete denial of service. Routers and switches, equipped with finite buffer memory, cannot accommodate unbounded traffic, especially during peak loads or sudden bursts.

For example, if multiple devices on a local area network (LAN) initiate high-bandwidth activities like downloading large files simultaneously, the upstream router may struggle to process all requests in real-time, causing congestion.

2.2 Traditional Congestion Control Techniques

Various congestion control mechanisms have been developed over the years, such as:

- TCP Congestion Control: Includes slow start, congestion avoidance, fast retransmit, and fast recovery. It dynamically adjusts the rate of data transmission based on network conditions.
- Token Bucket Algorithm: Unlike Leaky Bucket, this allows bursty traffic by accumulating tokens that permit packet transmissions.
- Random Early Detection (RED): A proactive method where routers probabilistically drop packets before buffers overflow.
- Explicit Congestion Notification (ECN): Marks packets to notify end systems of congestion without discarding them.

While these methods vary in complexity and efficiency, the Leaky Bucket Algorithm stands out due to its simplicity and deterministic behavior, making it ideal for foundational learning and applications in low-level embedded networking systems.

3. Leaky Bucket Algorithm

3.1 Concept and Working

The Leaky Bucket Algorithm enforces a fixed output rate irrespective of how bursty the incoming traffic is. Conceptually, it resembles a bucket with a small hole at the bottom:

- Incoming packets (data) are poured into the bucket.
- Data leaves the bucket at a fixed rate (output rate).
- If the bucket overflows due to incoming data exceeding the remaining space, the excess packets are discarded.

This mechanism is ideal for shaping traffic into a steady stream and preventing sudden bursts that could overwhelm the receiver or intermediary network devices.

3.2 Advantages and Limitations

Advantages:

- Predictable Output: Maintains a constant and controlled transmission rate.
- Simplicity: Easy to implement and understand.
- Buffer Management: Prevents buffer overflow by rejecting excess traffic.

Limitations:

- Bursty Traffic Handling: Not suitable for environments where variable burst tolerance is required.
- Packet Loss: High traffic rates result in more frequent drops, leading to inefficiency in high-speed networks.

4. Implementation in C++

4.1 Code Snippet

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>
#include <iomanip>

using namespace std;

class LeakyBucket {
private:
    int bucketSize;
    int outputRate;
    int buffer;
    int currentTime;
    vector<int> incomingPackets;

public:
    LeakyBucket(int bSize, int oRate, vector<int> packets)
        : bucketSize(bSize), outputRate(oRate), buffer(0), currentTime(0), incomingPackets(packets) {}

    void displayInitialInfo() {
        cout << "\n[INFO] Bucket Size     : " << bucketSize << " KB" << endl;
        cout << "[INFO] Output Rate     : " << outputRate << " KB/s" << endl;
        cout << "[INFO] Total Packets   : " << incomingPackets.size() << endl;
        cout << "-------------------------------------------------------------\n";
    }

    void simulateTraffic() {
        cout << left << setw(8) << "Time"
             << setw(15) << "Packet Size"
             << setw(15) << "Buffer Size"
             << setw(15) << "Action" << endl;
```

```
    for (int i = 0; i < incomingPackets.size(); ++i) {
        currentTime++;
        int pkt = incomingPackets[i];

        cout << left << setw(8) << currentTime
             << setw(15) << (to_string(pkt) + " KB");

        if (pkt + buffer <= bucketSize) {
            buffer += pkt;
            cout << setw(15) << (to_string(buffer) + " KB")
                 << "Accepted" << endl;
        } else {
            cout << setw(15) << (to_string(buffer) + " KB")
                 << "Dropped" << endl;
        }

        transmit();
    }

    while (buffer > 0) {
        currentTime++;
        cout << left << setw(8) << currentTime
             << setw(15) << "0 KB"
             << setw(15) << (to_string(buffer) + " KB")
             << "Sent" << endl;
        transmit();
    }
}
```

```
private:
    void transmit() {
        if (buffer > outputRate)
            buffer -= outputRate;
        else
            buffer = 0;
    }
};

vector<int> generatePackets(int n) {
    vector<int> packets;
    srand(time(0));
    for (int i = 0; i < n; ++i) {
        packets.push_back(rand() % 100 + 1); // Packet size between 1-100 KB
    }
    return packets;
}

int main() {
    int bucketSize, outputRate, numPackets;

    cout << "=== Leaky Bucket Algorithm (OOP Simulation) ===\n";
    cout << "Enter Bucket Size (KB): ";
    cin >> bucketSize;

    cout << "Enter Output Rate (KB/s): ";
    cin >> outputRate;

    cout << "Enter Number of Packets: ";
    cin >> numPackets;
```

```cpp
int main() {
    int bucketSize, outputRate, numPackets;

    cout << "=== Leaky Bucket Algorithm (OOP Simulation) ===\n";
    cout << "Enter Bucket Size (KB): ";
    cin >> bucketSize;

    cout << "Enter Output Rate (KB/s): ";
    cin >> outputRate;

    cout << "Enter Number of Packets: ";
    cin >> numPackets;

    vector<int> packets = generatePackets(numPackets);

    LeakyBucket lb(bucketSize, outputRate, packets);
    lb.displayInitialInfo();
    lb.simulateTraffic();

    return 0;
}
```

Results:

```
[INFO] Bucket Size     : 10 KB
[INFO] Output Rate     : 4 KB/s
[INFO] Total Packets   : 5
-------------------------------------------------------------
Time    Packet Size    Buffer Size    Action
-------------------------------------------------------------
1       3 KB           3 KB           Accepted
2       6 KB           5 KB           Accepted
3       4 KB           5 KB           Accepted
4       5 KB           6 KB           Dropped
5       2 KB           4 KB           Accepted
6       0 KB           2 KB           Sent
7       0 KB           0 KB           Sent
```

4.2 Explanation of Code and Results

The implementation of the Leaky Bucket Algorithm in C++ is designed to simulate a traffic shaping mechanism that regulates data flow in computer networks.

Key Concepts:
- The simulation accepts a number of incoming packets (from user input or predefined list).
- It uses a fixed-size buffer (bucket) to store the incoming data.
- An output rate determines how many packets can leave the buffer per unit time.
- If incoming packets exceed the bucket's capacity, they are dropped to prevent overflow.

Code Overview:
1. Variables are initialized to accept input for the number of packets, bucket size, and output rate.
2. Each incoming packet is checked against the buffer capacity. If it fits, it's accepted and added to the buffer; otherwise, it is dropped.
3. Packets in the buffer are transmitted at the defined output rate. Remaining data stays in the buffer for the next round.

Simulation Behavior:
- When traffic volume is within buffer capacity, all packets are accepted and transmitted.
- When traffic exceeds the capacity, excess packets are dropped, mimicking network congestion.
- The output remains steady and regulated due to the fixed leak rate.

a. User Input or Random Generation

int n, bucketSize, outputRate;

vector<int> packets;

- `n`: Number of packets.
- `bucketSize`: Maximum number of packets the bucket can hold.
- `outputRate`: Number of packets sent per unit time.
- `packets`: Stores sizes of incoming packets.

b. **Packet Arrival Simulation**

```
for (int i = 0; i < n; i++) {
   int pkt;
   cout << "Enter size of packet " << i+1 << ": ";
   cin >> pkt;
   packets.push_back(pkt);
}
```

- Accepts packet sizes from the user and stores them.
- These represent a traffic burst or variable packet stream.

c. **Bucket Processing Logic**

```
int buffer = 0;
for (int i = 0; i < n; i++) {
   if (packets[i] + buffer > bucketSize) {
      cout << "Packet " << i+1 << " of size " << packets[i] << " dropped.\n";
   } else {
      buffer += packets[i];
      cout << "Packet " << i+1 << " of size " << packets[i] << " accepted. Buffer = " << buffer << "\n";
   }

   // Send packets at output rate
   if (buffer > outputRate)
      buffer -= outputRate;
   else
      buffer = 0;
}
```

- **Check Capacity**: Adds packet only if it doesn't overflow the bucket.
- **Accept/Deny**: If it fits → *accepted*; else → *dropped*.
- **Transmit Data**: Sends out up to `outputRate` worth of data per cycle.
- **Buffer Update**: The remaining data stays in the bucket

This code effectively demonstrates how the Leaky Bucket algorithm controls network traffic, limits burst transmissions, and prevents buffer overflows.

Potential Enhancements:
- Integrate live packet capture using socket programming.
- Create GUI-based visualization of packet flow and buffer usage.
- Add adaptive transmission rate logic to simulate dynamic real-world scenarios.
- Combine with Token Bucket for improved burst tolerance.

Overall, this simulation serves as a foundational model for understanding congestion control in computer networks.

- **Packet Generation**: A vector of packet sizes mimics real network data bursts. These can be either random or fixed values.
- **Buffer Check**: For every new packet, the algorithm checks if the buffer can accommodate it. If not, the packet is dropped—demonstrating congestion.
- **Transmission**: Packets are transmitted at a constant output rate. This process continues until all buffered packets are sent.

Observed Behavior:

- When traffic is **within buffer capacity**, all packets are accepted and processed smoothly.
- When traffic **exceeds buffer capacity**, overflow occurs, and packets are discarded.
- Once all incoming packets are handled, the buffer is gradually emptied at the set output rate.

5. Conclusion

The Leaky Bucket Algorithm is a reliable, deterministic traffic shaping tool that offers straightforward control over congestion in networking systems. Through our object-oriented C++ simulation, we demonstrated its practicality and relevance in understanding fundamental congestion control concepts.

The results of the simulation reinforce key takeaways:

- Congestion can be mitigated by controlling the rate at which data is introduced to the network.
- While simple, the Leaky Bucket does not accommodate traffic bursts, which might limit its applicability in modern dynamic networks.
- Nevertheless, it is an excellent starting point for understanding more complex algorithms and their impact on QoS.

This exercise underscores the importance of algorithmic modeling in network design and encourages further exploration of adaptive systems like TCP/IP's congestion control, which builds upon these fundamental principles.
The simulation output (console-based) displays time, packet size, current buffer, and the action taken (Accepted, Dropped, Sent). This allows easy interpretation of how well the algorithm performs under various conditions. It also highlights the importance of choosing optimal buffer size and output rate parameters

6. Future Scope and Enhancements

While the current implementation and simulation of the Leaky Bucket Algorithm demonstrate foundational congestion control principles, several enhancements and expansions can be explored to align with real-world complexities and advanced networking paradigms**.**

6.1 Real-Time Packet Capture

A significant future improvement involves integrating real-time data capture. Using socket programming or libraries like pcap, the algorithm could analyze live traffic from a network interface, allowing for more realistic simulation and validation against actual congestion scenarios.

6.2 Visualization Interface

Creating a graphical or web-based visualization using libraries such as Qt, SFML, or Matplotlib (via C++-Python integration) would make the simulation more intuitive. Dynamic displays of packet movement, buffer levels, and packet drops can improve educational value and user engagement.

6.3 Adaptive Output Rate

Incorporating adaptive behavior into the output rate based on buffer conditions, network load, or external feedback (similar to TCP dynamic adjustments) would simulate more intelligent congestion control. This could pave the way for hybrid algorithms blending Leaky Bucket with ECN or RED principles.

6.4 Token Bucket Integration

To address limitations in bursty traffic handling, combining the Leaky Bucket with a Token Bucket mechanism can help support variable packet arrivals while maintaining output rate constraints. A dual-bucket simulation can better reflect real-time ISP traffic shaping models.

6.5 Cross-Layer Simulation

For more advanced simulations, integrating this algorithm into a network simulation environment like NS-3, OMNeT++, or Mininet would allow analysis at various layers (MAC, transport, application), enabling end-to-end performance testing under diverse conditions.

6.6 Logging and Performance Metrics

Future versions could log detailed performance metrics like:

- Total packets accepted, dropped, and sent.
- Average queue length and buffer occupancy over time.
- Packet drop rates and their correlation with traffic bursts.

These analytics would help evaluate algorithm efficiency and suggest further optimizations.

Various congestion control mechanisms have been developed over the years, such as:

- TCP Congestion Control: Includes slow start, congestion avoidance, fast retransmit, and fast recovery. It dynamically adjusts the rate of data transmission based on network conditions.
- Token Bucket Algorithm: Unlike Leaky Bucket, this allows bursty traffic by accumulating tokens that permit packet transmissions.
- Random Early Detection (RED): A proactive method where routers probabilistically drop packets before buffers overflow.
- Explicit Congestion Notification (ECN): Marks packets to notify end systems of congestion without discarding them.

7. References

[1] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 5th ed., Pearson, 2010.
[2] B. A. Forouzan, *Data Communications and Networking*, 5th ed., McGraw-Hill, 2012.
[3] W. Stallings, *High-Speed Networks and Internets*, 2nd ed., Prentice Hall, 2002.
[4] IEEE Citation Guidelines. Available: https://ieeeauthorcenter.ieee.org