PROJECT REPORT

“Token Bucket algorithm”

In a partial fulfilment of the requirements for

the award of the degree of

BACHELOR OF TECHNOLOGY

In

ELECTRONIC AND COMMUNICATION ENGINEERING

Prepared and Submitted by:

(U22EC100) AKHIL RAI	(U22EC133) SMIT B PANDYA
(U22EC120) HET CHAUDHARI	(U22EC134) VINIT SONI
(U22EC123) VARUN SHAH	(U22EC139) PRINCE SOLANKI
(U22EC125) MEET CHUDASAMA	(U22EC153) YASHWANTH RAM
(U22EC130) MAN SINGH	(U22EC159) PATEL BHAVYA

B.Tech – IIIrd Year – Vth Semester

DEPARTMENT OF ELECTRONICS ENGINEERING

SARDAR VALLBHBHAI NATIONAL INSTITUTE OF TECHNOLOGY SURAT,

395007, Gujarat, India

Certificate

This is to certify that Report – entitled

"Token Bucket algorithm"

Submitted and presented by

Sr No.	Name	Roll No.
1.	AKHIL RAI	U22EC100
2.	HET CHAUDHARI	U22EC120
3.	VARUN SHAH	U22EC123
4.	MEET CHUDASAMA	U22EC125
5.	MAN SINGH	U22EC130
6.	SMIT B PANDYA	U22EC133
7.	VINIT SONI	U22EC134
8.	PRINCE SOLANKI	U22EC139
9.	YASHWANTH RAM	U22EC153
10.	PATEL BHAVYA	U22EC159

B.Tech – III year–Vth semester in the partial fulfillment of the requirement for the award of B.Tech degree in Electronics and Communication Engineering for academic year 2024-25.

They have successfully and satisfactorily completed their report in all respects. We certify that the work is comprehensive, complete and fit for evaluation.

Signature

Dr. Raghavendra Pal

Abstract

This Scilab code implements an extended Token Bucket algorithm simulation to regulate bursty input traffic while analyzing token management and compliance. The simulation models dynamic token addition with a configurable bucket size, initial token rate, and optional variable token rates over time. It tracks key metrics such as overflow and underflow occurrences, token availability, and output traffic shaping efficiency. The code includes detailed visualization functions to compare input and output traffic, monitor token availability, and evaluate compliance ratios. Additionally, it generates summary metrics for performance evaluation, offering a comprehensive tool for studying traffic shaping and rate-limiting mechanisms in network systems.

Acknowledgment

We thank God Almighty for His boundless grace, guidance, and blessings that have made this project possible. Without His divine support, this journey would not have been achievable. We are deeply grateful for the strength, wisdom, and inspiration He has provided throughout the course of this project.

We extend our heartfelt thanks to Dr. Raghavendra Pal, whose mentorship, expertise, and constant encouragement have been pivotal in the successful execution of this project. His invaluable guidance has greatly contributed to shaping our understanding and the direction of our work.

Our sincere gratitude also goes to the entire faculty and staff of the Electronics Department, whose support and knowledge have provided a conducive learning environment. Their dedication, along with the resources and facilities provided, has played an essential role in completing this project.

We would also like to express our appreciation to our peers, friends, and families for their unwavering support and motivation. Their encouragement has been a constant source of strength and inspiration.

We dedicate the success of this endeavor to the grace and mercy of God Almighty.

Table of Contents

Introduction

Token Bucket algorithm, a widely used technique for traffic shaping in modern network environments. The program aims to implement the Token Bucket algorithm with configurable bucket size and token generation rate, allowing users to explore different traffic shaping scenarios; simulate the traffic flow through the bucket and observe the effect of token availability on data transmission; introduce bursty input traffic and compare the shaped output traffic; and plot the token availability and output data rate over time, providing valuable insights into the algorithm's behavior and its impact on network performance. By addressing these objectives, the SCILAB program serves as a valuable tool for understanding the principles of the Token Bucket algorithm and its practical applications in network traffic management, helping network administrators, researchers, and engineers optimize their traffic shaping strategies and improve the overall reliability and efficiency of their network infrastructure.

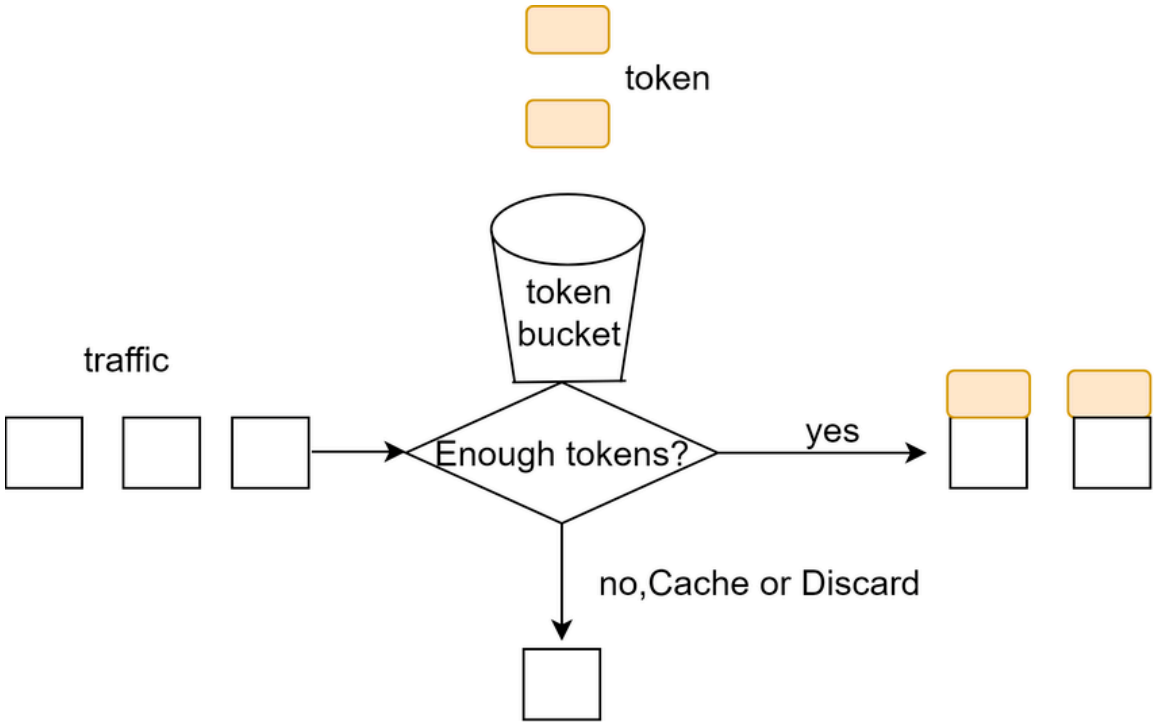


Figure 1.1: *Introduction to Token Bucket Algorithm*

Objective

The primary objective of this SCILAB program is to provide a comprehensive simulation of the Token Bucket algorithm for traffic shaping, with the aim of implementing the algorithm with configurable bucket size and token generation rate to allow for the exploration of different traffic shaping scenarios; simulating the traffic flow through the bucket and observing the effect of token availability on data transmission to gain insights into the dynamics of the algorithm; introducing bursty input traffic and comparing the shaped output traffic to demonstrate the algorithm's ability to smooth out the data flow; and generating visual representations of the token availability and output data rate over time to offer valuable insights into the performance and behavior of the Token Bucket algorithm, enabling network administrators, researchers, and engineers to optimize their traffic shaping strategies and improve the overall efficiency and reliability of their network infrastructure.

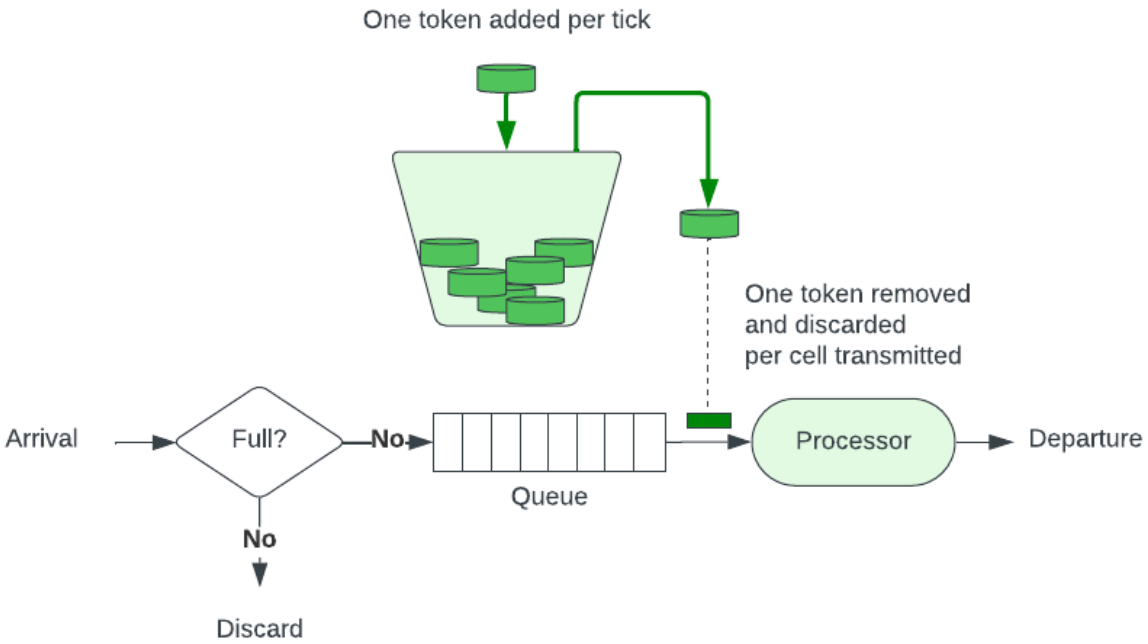


Figure 1.2: *Token Bucket Algorithm*

Algorithm Overview

Initialization

1. **Set Parameters**:
    - `bucketSize`: Maximum capacity of the token bucket (maximum tokens it can hold).
    - `initialTokenRate`: Initial rate at which tokens are added to the bucket per time unit.
    - `timeDuration`: Total duration of the simulation.
2. **Initialize Variables**:
    - `currentTokens`: Tracks the number of tokens currently in the bucket (starts full or empty based on configuration).
    - `outputTraffic`: Array to record the amount of traffic allowed to pass through the bucket at each time step.
    - `tokenAvailability`: Array to track the number of tokens available at each time step.
    - `overflowCount`: Counter to record instances where tokens exceed the bucket's maximum capacity (overflow).
    - `underflowCount`: Counter to record instances where insufficient tokens lead to dropped traffic (underflow).

Simulation Loop (Runs for the specified simulation duration):

1. **Determine Token Rate**:
    - Use the token rate for the current time step if a variable token rate is provided.
    - Otherwise, use the `initialTokenRate`.
2. **Add Tokens to the Bucket**:
    - Add tokens to the bucket based on the current token rate.
    - Ensure the bucket does not exceed its maximum capacity (`bucketSize`).
    - Increment the `overflowCount` if tokens are discarded due to overflow.

3. **Process Incoming Traffic**:
    - If the incoming traffic for the current time step is less than or equal to the available tokens:
        - Allow the entire traffic to pass and deduct the corresponding tokens.
    - If the incoming traffic exceeds the available tokens:
        - Allow only the available tokens to pass through.
        - Set remaining tokens to zero and increment the `underflowCount`.
4. **Update Token Availability**:
    - Record the current number of tokens remaining in the bucket in the `tokenAvailability` array.

Post-Simulation Analysis:

1. **Calculate Metrics**:
    - Compute the average input traffic, output traffic, and compliance ratio (output/input).
    - Count total occurrences of overflow and underflow.
2. **Generate Plots**:
    - Plot input and output traffic over time to visualize traffic shaping.
    - Plot token availability over time to observe token dynamics.
    - Plot compliance ratio over time to evaluate how closely the output aligns with the input.
    - Display overflow and underflow counts in a bar chart to summarize token performance.
3. **Display Summary**:
    - Output key metrics to the console, including average traffic, compliance ratio, and total overflow/underflow counts.

Key Highlights:

- The **Token Bucket Algorithm** efficiently regulates bursty traffic by using tokens as a metaphor for available data units that can be transmitted.
- **Token rate** determines the average allowed transmission rate, while **bucket size** defines the maximum burst capacity.

- The simulation ensures fairness and prevents congestion by enforcing limits on transmitted data.
- This implementation enables the analysis of traffic-shaping performance under various conditions and parameters, providing detailed insights into network resource allocation and traffic conformance.

Methodology

Overview of the Token Bucket Algorithm

The Token Bucket algorithm is a network traffic-shaping mechanism that regulates data flow by allowing transmission only when sufficient tokens are available. Each token represents permission to transmit a specific data unit. Tokens are added at either a fixed or variable rate up to the bucket's maximum capacity, with excess tokens discarded in cases of overflow. When tokens are depleted, traffic may be delayed or partially transmitted, resulting in underflow. This approach ensures fairness and prevents congestion, making it effective for traffic control in networks.

Program Structure

The program is designed in Scilab with a modular structure, including separate functions for simulation, visualization, and analysis, ensuring code clarity, reusability, and ease of maintenance. The following components form the foundation of the methodology:

1. **Simulation Function**:
    - Simulates the behavior of the Token Bucket algorithm based on input parameters.
    - **Inputs**:
        - `bucketSize`: Maximum token capacity of the bucket.
        - `initialTokenRate`: Fixed rate of token generation when no dynamic rate is provided.
        - `timeDuration`: Duration of the simulation in time units.
        - `burstyTraffic`: Input traffic data representing variable transmission demands.
        - `variableTokenRate`: Optional array defining dynamic token generation rates over time.
    - **Outputs**:

        - `outputTraffic`: Allowed traffic transmitted at each time step.
        - `tokenAvailability`: Tokens available over time.
        - `overflowCount`: Number of instances when tokens exceeded the bucket capacity.
        - `underflowCount`: Number of instances when token depletion prevented full traffic transmission.
2. **Visualization Functions**:
    - Generate plots to analyze the simulation results:
        - Input vs. output traffic, showing the shaping effect.
        - Token availability over time.
        - Compliance ratio (output/input traffic).
        - Overflow and underflow counts as bar charts.
3. **Summary Metrics**:
    - Compute and display:
        - Average input and output traffic rates.
        - Average compliance ratio to evaluate adherence to traffic limits.
        - Total overflow and underflow counts for performance assessment.
4. **Main Driver Function**:
    - Serves as the program's entry point and orchestrates the simulation:
        - Calls the simulation function with the defined parameters.
        - Invokes visualization functions to generate insightful plots.
        - Displays a summary of the simulation's key metrics.
    - Provides a seamless workflow for executing the program.

Key Features

1. **Dynamic Token Rate Handling**:

- Supports dynamic token generation rates through a variable array. If not provided, the program defaults to a static `initialTokenRate`.

2. **Detailed Logging**:
   - Logs intermediate results during simulation, such as token availability, overflow, and underflow events, enhancing transparency and debugging.
3. **Comprehensive Metrics**:
   - Tracks essential indicators, including compliance ratios, overflow/underflow events, and traffic rates, to evaluate the algorithm's efficiency.
4. **Enhanced Visualization**:
   - Provides multiple plots to interpret traffic shaping, token management, and algorithm compliance comprehensively.
5. **Realistic Traffic Patterns**:
   - Allows simulation of realistic network scenarios using a bursty traffic array, effectively testing the robustness of the Token Bucket algorithm.

Flow of Execution

1. **Initialization**:
   - Define key parameters, including `bucketSize`, `initialTokenRate`, `timeDuration`, `burstyTraffic`, and optionally `variableTokenRate`.
2. **Simulation**:
   - The `tokenBucketSimulation` function processes traffic step-by-step, updating tokens and tracking overflow and underflow instances.

3. **Visualization**:
    - Generate four detailed plots to analyze the simulation results from multiple perspectives.
4. **Summary Metrics**:
    - Compute and display metrics to summarize the algorithm's performance in terms of compliance, traffic rates, and token management.
5. **Output**:
    - Present simulation results, enabling users to assess the efficiency of the Token Bucket algorithm.

Extensibility and Use Cases

The program is designed for extensibility, allowing users to adapt it for more complex scenarios such as:

- Hierarchical traffic shaping using multiple token buckets.
- Incorporating additional models like latency or packet loss.
- Testing under varied traffic patterns or network conditions.

This Scilab implementation serves as both a robust educational tool for understanding the Token Bucket algorithm and a practical framework for analyzing traffic shaping mechanisms in network systems.

Code

```
// Main simulation function with added parameters and detailed output
function [outputTraffic, tokenAvailability, overflowCount,
underflowCount] = tokenBucketSimulation(bucketSize, initialTokenRate,
timeDuration, burstyTraffic, variableTokenRate)
    // Simulates a token bucket algorithm with enhanced features:
    // - bucketSize: Maximum tokens the bucket can hold
    // - initialTokenRate: Starting rate of token addition per time
unit
    // - timeDuration: Total duration of the simulation
    // - burstyTraffic: Array of input traffic data over time
    // - variableTokenRate: Optional array of token rates over time for
dynamic rates

    // Initialize variables
    currentTokens = bucketSize;                    // Start with a full
bucket
    outputTraffic = zeros(1, timeDuration);        // Output traffic
array
    tokenAvailability = zeros(1, timeDuration);    // Token
availability tracking
    overflowCount = 0;                             // Counter for token
overflow instances
    underflowCount = 0;                            // Counter for token
depletion instances

    disp("Starting Extended Token Bucket Simulation...");
    disp("Bucket Size: " + string(bucketSize));
    disp("Initial Token Rate: " + string(initialTokenRate));
    disp("Time Duration: " + string(timeDuration));

    // Simulation loop
    for t = 1:timeDuration
        // Determine token rate at time t (dynamic or static)
        if variableTokenRate(t) <> -1 then
            tokenRate = variableTokenRate(t);
```

```
        else
            tokenRate = initialTokenRate;
        end

        // Add tokens, respecting bucket size limit
        previousTokens = currentTokens;
        currentTokens = min(currentTokens + tokenRate, bucketSize);
        if currentTokens > bucketSize then
            overflowCount = overflowCount + 1;
        end

        disp("Time " + string(t) + ": Tokens before traffic processing:
" + string(currentTokens));

        // Check if enough tokens are available for current bursty
traffic
        if burstyTraffic(t) <= currentTokens then
            outputTraffic(t) = burstyTraffic(t);
            currentTokens = currentTokens - burstyTraffic(t);
            disp("  Full transmission allowed: " +
string(outputTraffic(t)) + " tokens left: " + string(currentTokens));
        else
            outputTraffic(t) = currentTokens;
            currentTokens = 0;
            underflowCount = underflowCount + 1;
            disp("  Partial transmission: " + string(outputTraffic(t))
+ " tokens depleted to zero.");
        end

        // Log token availability at each step
        tokenAvailability(t) = currentTokens;
    end

    disp("Simulation Complete.");
    disp("Total Overflow Instances: " + string(overflowCount));
    disp("Total Underflow Instances: " + string(underflowCount));
endfunction

// Helper function to plot input vs output traffic
function plotTrafficVsOutput(timeDuration, burstyTraffic,
```

```
outputTraffic)
    // Plot input bursty traffic and shaped output traffic
    t = 1:timeDuration;
    subplot(4, 1, 1);
    plot(t, burstyTraffic, 'r-', t, outputTraffic, 'b-', "LineWidth",
2);
    title("Bursty Input Traffic vs. Shaped Output Traffic");
    xlabel("Time");
    ylabel("Data Rate");
    legend(["Input", "Output"], "location", "northwest");
endfunction

// Helper function to plot token availability
function plotTokenAvailability(timeDuration, tokenAvailability)
    // Plot token availability over time
    t = 1:timeDuration;
    subplot(4, 1, 2);
    plot(t, tokenAvailability, 'g-', "LineWidth", 2);
    title("Token Availability Over Time");
    xlabel("Time");
    ylabel("Tokens Available");
    legend("Tokens");
endfunction

// Helper function to plot compliance ratio
function plotComplianceRatio(burstyTraffic, outputTraffic,
timeDuration)
    // Plot compliance ratio (output/input)
    complianceRatio = outputTraffic ./ burstyTraffic;
    t = 1:timeDuration;
    subplot(4, 1, 3);
    plot(t, complianceRatio, 'b-', "LineWidth", 2);
    title("Rate Compliance of Shaped Output Traffic");
    xlabel("Time");
    ylabel("Compliance Ratio");
    legend("Compliance Ratio");
endfunction

// Helper function to plot overflow and underflow counts
function plotOverflowUnderflow(overflowCount, underflowCount)
```

```
    // Display overflow and underflow instances as a bar chart
    subplot(4, 1, 4);
    bar([overflowCount, underflowCount], "stacked");
    title("Token Overflow and Underflow Instances");
    xticks(1:2);
    xticklabels(["Overflow", "Underflow"]);
    ylabel("Count");
endfunction

// Function to display summary metrics
function displaySummaryMetrics(overflowCount, underflowCount,
outputTraffic, burstyTraffic)
    // Display key performance metrics
    avgOutputTraffic = mean(outputTraffic);
    avgInputTraffic = mean(burstyTraffic);
    complianceRatioAvg = mean(outputTraffic ./ burstyTraffic);

    disp("Simulation Summary:");
    disp("  Average Input Traffic: " + string(avgInputTraffic));
    disp("  Average Output Traffic: " + string(avgOutputTraffic));
    disp("  Average Compliance Ratio: " + string(complianceRatioAvg));
    disp("  Total Overflow Count: " + string(overflowCount));
    disp("  Total Underflow Count: " + string(underflowCount));
endfunction

// Main function to run the simulation and generate plots
function runSimulation(bucketSize, initialTokenRate, timeDuration,
burstyTraffic, variableTokenRate)
    // Run the token bucket simulation
    disp("Running the Extended Token Bucket Simulation...");
    [outputTraffic, tokenAvailability, overflowCount, underflowCount] =
...
        tokenBucketSimulation(bucketSize, initialTokenRate,
timeDuration, burstyTraffic, variableTokenRate);

    // Generate plots
    clf();
    plotTrafficVsOutput(timeDuration, burstyTraffic, outputTraffic);
    plotTokenAvailability(timeDuration, tokenAvailability);
    plotComplianceRatio(burstyTraffic, outputTraffic, timeDuration);
```

```
    plotOverflowUnderflow(overflowCount, underflowCount);

    // Display summary metrics
    displaySummaryMetrics(overflowCount, underflowCount, outputTraffic,
burstyTraffic);
    disp("Simulation and Visualization Complete.");
endfunction

// Simulation Parameters
bucketSize = 10;                // Maximum tokens in the bucket
initialTokenRate = 2;           // Token generation rate
timeDuration = 50;              // Simulation duration in time units

// Define bursty input traffic
burstyTraffic = [4, 5, 7, 1, 3, 9, 2, 0, 8, 10, 6, 1, 4, 5, 7, 2, 6, 3,
9, 10, ...
                 0, 2, 8, 4, 6, 3, 7, 9, 5, 1, 8, 2, 6, 4, 7, 9, 3, 5,
2, 8, ...
                 1, 7, 6, 9, 4, 2, 3, 8, 5, 6];

// Define dynamic token rate (optional)
variableTokenRate = [2, 2, 3, 3, 2, 1, 4, 2, 3, 2, 1, 3, 2, 3, 4, 2, 3,
1, 3, 2, ...
                     2, 2, 3, 2, 3, 4, 2, 2, 1, 3, 3, 2, 3, 3, 2, 4, 1,
3, 2, 3, ...
                     3, 1, 2, 3, 3, 2, 4, 2, 3, 1];

// Run the simulation
runSimulation(bucketSize, initialTokenRate, timeDuration,
burstyTraffic, variableTokenRate);
```

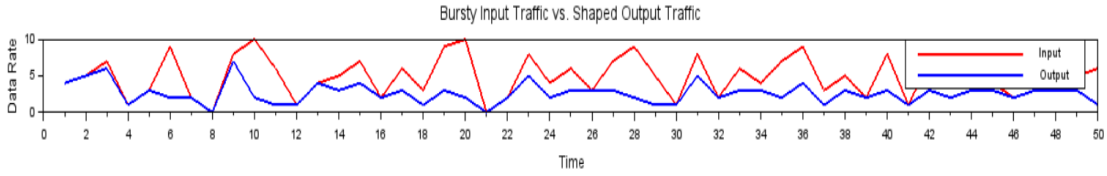
Simulation Details

```
"Running the Extended Token Bucket Simulation..."
  "Starting Extended Token Bucket Simulation..."
  "Bucket Size: 10"
  "Initial Token Rate: 2"
  "Time Duration: 50"
  "Time 1: Tokens before traffic processing: 10"
  "  Full transmission allowed: 4 tokens left: 6"
  "Time 2: Tokens before traffic processing: 8"
  "  Full transmission allowed: 5 tokens left: 3"
  "Time 3: Tokens before traffic processing: 6"
  "  Partial transmission: 6 tokens depleted to zero."
  "Time 4: Tokens before traffic processing: 3"
  "  Full transmission allowed: 1 tokens left: 2"
  "Time 5: Tokens before traffic processing: 4"
  "  Full transmission allowed: 3 tokens left: 1"
  "Time 6: Tokens before traffic processing: 2"
  "  Partial transmission: 2 tokens depleted to zero."
  "Time 7: Tokens before traffic processing: 4"
  "  Full transmission allowed: 2 tokens left: 2"
  "Time 8: Tokens before traffic processing: 4"
  "  Full transmission allowed: 0 tokens left: 4"
  "Time 9: Tokens before traffic processing: 7"
  "  Partial transmission: 7 tokens depleted to zero."
  "Time 10: Tokens before traffic processing: 2"
  "  Partial transmission: 2 tokens depleted to zero."
  "Time 11: Tokens before traffic processing: 1"
  "  Partial transmission: 1 tokens depleted to zero."
  "Time 12: Tokens before traffic processing: 3"
  "  Full transmission allowed: 1 tokens left: 2"
  "Time 13: Tokens before traffic processing: 4"
  "  Full transmission allowed: 4 tokens left: 0"
  "Time 14: Tokens before traffic processing: 3"
  "  Partial transmission: 3 tokens depleted to zero."
  "Time 15: Tokens before traffic processing: 4"
```

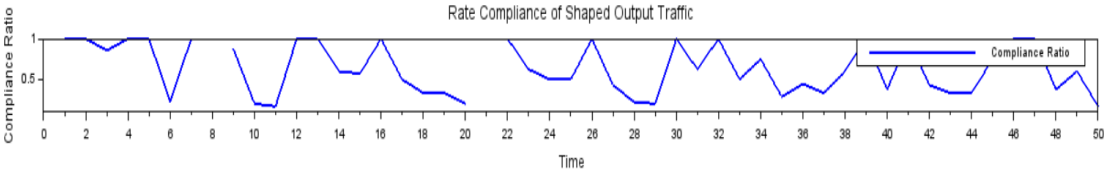
```
"  Partial transmission: 4 tokens depleted to zero."
"Time 16: Tokens before traffic processing: 2"
"  Full transmission allowed: 2 tokens left: 0"
"Time 17: Tokens before traffic processing: 3"
"  Partial transmission: 3 tokens depleted to zero."
"Time 18: Tokens before traffic processing: 1"
"  Partial transmission: 1 tokens depleted to zero."
"Time 19: Tokens before traffic processing: 3"
"  Partial transmission: 3 tokens depleted to zero."
"Time 20: Tokens before traffic processing: 2"
"  Partial transmission: 2 tokens depleted to zero."
"Time 21: Tokens before traffic processing: 2"
"  Full transmission allowed: 0 tokens left: 2"
"Time 22: Tokens before traffic processing: 4"
"  Full transmission allowed: 2 tokens left: 2"
"Time 23: Tokens before traffic processing: 5"
"  Partial transmission: 5 tokens depleted to zero."
"Time 24: Tokens before traffic processing: 2"
"  Partial transmission: 2 tokens depleted to zero."
"Time 25: Tokens before traffic processing: 3"
"  Partial transmission: 3 tokens depleted to zero."
"Time 26: Tokens before traffic processing: 4"
"  Full transmission allowed: 3 tokens left: 1"
"Time 27: Tokens before traffic processing: 3"
"  Partial transmission: 3 tokens depleted to zero."
"Time 28: Tokens before traffic processing: 2"
"  Partial transmission: 2 tokens depleted to zero."
"Time 29: Tokens before traffic processing: 1"
"  Partial transmission: 1 tokens depleted to zero."
"Time 30: Tokens before traffic processing: 3"
"  Full transmission allowed: 1 tokens left: 2"
"Time 31: Tokens before traffic processing: 5"
"  Partial transmission: 5 tokens depleted to zero."
"Time 32: Tokens before traffic processing: 2"
"  Full transmission allowed: 2 tokens left: 0"
"Time 33: Tokens before traffic processing: 3"
"  Partial transmission: 3 tokens depleted to zero."
"Time 34: Tokens before traffic processing: 3"
"  Partial transmission: 3 tokens depleted to zero."
```

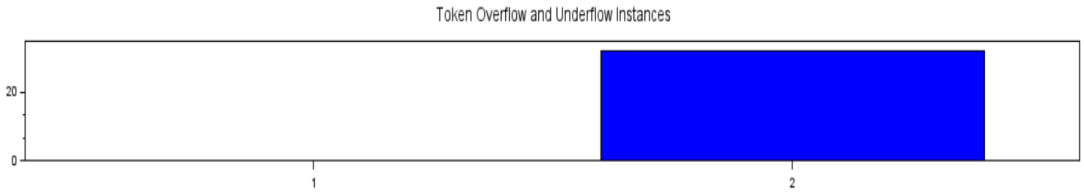
"Time 35: Tokens before traffic processing: 2"
" Partial transmission: 2 tokens depleted to zero."
"Time 36: Tokens before traffic processing: 4"
" Partial transmission: 4 tokens depleted to zero."
"Time 37: Tokens before traffic processing: 1"
" Partial transmission: 1 tokens depleted to zero."
"Time 38: Tokens before traffic processing: 3"
" Partial transmission: 3 tokens depleted to zero."
"Time 39: Tokens before traffic processing: 2"
" Full transmission allowed: 2 tokens left: 0"
"Time 40: Tokens before traffic processing: 3"
" Partial transmission: 3 tokens depleted to zero."
"Time 41: Tokens before traffic processing: 3"
" Full transmission allowed: 1 tokens left: 2"
"Time 42: Tokens before traffic processing: 3"
" Partial transmission: 3 tokens depleted to zero."
"Time 43: Tokens before traffic processing: 2"
" Partial transmission: 2 tokens depleted to zero."
"Time 44: Tokens before traffic processing: 3"
" Partial transmission: 3 tokens depleted to zero."
"Time 45: Tokens before traffic processing: 3"
" Partial transmission: 3 tokens depleted to zero."
"Time 46: Tokens before traffic processing: 2"
" Full transmission allowed: 2 tokens left: 0"
"Time 47: Tokens before traffic processing: 4"
" Full transmission allowed: 3 tokens left: 1"
"Time 48: Tokens before traffic processing: 3"
" Partial transmission: 3 tokens depleted to zero."
"Time 49: Tokens before traffic processing: 3"
" Partial transmission: 3 tokens depleted to zero."
"Time 50: Tokens before traffic processing: 1"
" Partial transmission: 1 tokens depleted to zero."
"Simulation Complete."
"Total Overflow Instances: 0"
"Total Underflow Instances: 32"

Results


Bursty Input Traffic vs. Shaped Output Traffic
Data Rate
Time
Input
Output


Token Availability Over Time
Tokens Available
Time
Tokens


Rate Compliance of Shaped Output Traffic
Compliance Ratio
Time
Compliance Ratio


Token Overflow and Underflow Instances

Conclusion

The SCILAB-based implementation and simulation of the Token Bucket algorithm have successfully demonstrated its effectiveness as a traffic shaping mechanism in network communication. By dynamically adjusting parameters such as bucket size and token generation rate, the algorithm showcased its capability to regulate data flow, manage bursty traffic, and prevent congestion. The analysis emphasized the critical role of token availability in determining output data rates and maintaining a balance between input demands and network capacity. These findings reaffirm the practical applicability of the Token Bucket algorithm in real-world scenarios, particularly for ensuring bandwidth management, stabilizing data transmission, and enhancing overall network performance.

Bibliography

1. Tanenbaum, A. S., & Wetherall, D. J. (2011). *Computer networks*. Pearson Education.

2. Stallings, W. (2014). *Data and computer communications*. Prentice Hall.

3. Forouzan, B. A. (2017). *Data communications and networking*. McGraw-Hill.

4. SCILAB Official Documentation. (n.d.). Retrieved from https://www.scilab.org/documentation

5. Comer, D. E. (2013). *Internetworking with TCP/IP: Principles, protocols, and architecture*. Pearson Education.

6. Kurose, J. F., & Ross, K. W. (2021). *Computer networking: A top-down approach*. Pearson.

7. Token Bucket Algorithm Explained. (n.d.). Online Networking Tutorials. Retrieved from https://www.networkingtutorials.com