Riga Technical University

Faculty of Computer Science, Information Technology and Energy Institute of Telecommunications

Final Report

Teletraffic Theory

Submitted by:

ARUN KUMAR CHETHAKUDAM SHAJI

231AEM012

Riga 2024

1. INTRODUCTION

1.1. Teletraffic Theory

Teletraffic theory focuses on the analysis, modelling, and management of traffic in telecommunication networks. It provides a mathematical framework for understanding and predicting the behaviour of traffic flows, resource utilization, and performance measures in communication systems. The term "teletraffic" refers to the traffic generated by users or applications in a network, including voice calls, data transfers, video streaming, and other types of communication (Akimaru, and Kawashima, 2012).

The goal of teletraffic theory is to develop models and techniques that enable network designers, operators, and planners to efficiently design, dimension, and manage telecommunication networks to meet the demands of users while maintaining acceptable levels of quality of service. By understanding the fundamental principles of teletraffic behaviour, it becomes possible to optimize network resources, minimize congestion, and ensure efficient utilization of available capacity.

Teletraffic theory draws upon concepts from queuing theory, probability theory, statistics, and stochastic processes to analyse and predict the behaviour of telecommunication networks. It involves studying various performance metrics such as call blocking probability, traffic intensity, queuing delays, packet loss rates, and capacity requirements (Basharin, et al., 2009).

Key concepts in teletraffic theory include traffic models, which describe the arrival and departure patterns of traffic in a network, and queuing models, which capture the behaviour of queues formed by traffic in the system. Different traffic models, such as Poisson processes, Markov modulated processes, and self-similar processes, are used to represent different types of traffic sources and their statistical properties.

By analysing these models and applying mathematical techniques, teletraffic theory provides insights into how network performance is affected by factors such as traffic load, routing algorithms, network topology, and resource allocation strategies. This knowledge can then be used to make informed decisions regarding network planning, capacity provisioning, and traffic engineering.

1.2. Queueing Theory

Queueing theory is a branch of applied mathematics that deals with the mathematical modelling and analysis of waiting lines, or queues, in various systems. It provides a framework for studying and understanding the behaviour of queues, the utilization of resources, and the performance of systems where customers or entities arrive, wait, and are served (Cooper, 1981).

The main objectives of queueing theory are to quantify and optimize system performance, predict waiting times and queue lengths, evaluate resource utilization, and make informed decisions regarding system design and management (Adan, and Resing, 2002).

Key components and concepts in queueing theory include:

- Arrival Process: The arrival process describes how customers or entities arrive at the system. It can be modelled using stochastic processes, such as Poisson processes, where arrivals occur randomly and independently over time.
- Service Time Distribution: The service time distribution represents the time taken to serve a customer. It can be modelled using various probability distributions, such as exponential, normal, or general distributions.
- Queueing Discipline: The queueing discipline determines the order in which customers are served. Common disciplines include first-come, first-served (FCFS), last-come, first-served (LCFS), priority-based, and more complex scheduling algorithms.
- System Capacity: The system capacity refers to the maximum number of customers the system can accommodate or the number of available servers. It can be finite or infinite, depending on the system's design (Sztrik, 2012).
- Performance Measures: Various performance measures are used to evaluate the effectiveness and efficiency of a queueing system. These measures include average waiting time, queue length, system utilization, throughput, and customer satisfaction.
- Queueing theory provides mathematical models to analyse and predict system performance based on these components.

Some commonly used queueing models include:

- M/M/1: Represents a single-server queueing system with Poisson arrivals, exponentially distributed service times, and an infinite queue.
- M/M/c: Represents a queueing system with multiple identical servers, Poisson arrivals, exponentially distributed service times, and an infinite queue.
- M/G/1: Represents a single-server queueing system with Poisson arrivals, a general service time distribution (not necessarily exponential), and an infinite queue.

Queueing theory allows for the calculation of performance metrics and the evaluation of different system configurations to optimize performance. It helps in making decisions regarding resource allocation, capacity planning, service level agreements, and system design.

Queueing theory finds applications in a wide range of fields, including telecommunications, computer networks, transportation, healthcare, manufacturing, service industries, and more. By applying queueing theory, practitioners can gain insights into system behaviour, make informed decisions, and improve the efficiency and effectiveness of systems with waiting lines.

2. Single Node Model

Little's Law

Little's Law is a fundamental theorem in queueing theory which states that the long-term average number of customers in a stable system

L is equal to the long-term average effective arrival rate λ multiplied by the average time a customer spends in the system W.

The formula is: $L=\lambda W$

Erlang Equations

Erlang equations are used to describe the behaviour of queues in telephony and computer networks. For a single server system (M/M/1 queue), the key metrics can be derived as follows:

The Erlang used to calculate the probability that an arriving customer will have to wait for service (i.e., the system is busy). For an M/M/1 queue, the utilization factor ρ (or traffic intensity) is defined as:

$\rho = \lambda/\mu$

Where:

λ is the arrival rate.

μ is the service rate.

Key Metrics for M/M/1 Queue

Average number of customers in the system (L)

$L = \lambda/\mu - \lambda$

Average time a customer spends in the system (W):

$W = 1/\mu - \lambda$

Average number of customers in the queue (L_q):

$L_q = \lambda^2 / \mu(\mu - \lambda)$

Average time a customer spends waiting in the queue (W_q):

$W_q = \lambda / \mu(\mu - \lambda)$

Probability that the server is busy

$\rho = \lambda/\mu$

2. SINGLE NODE MODEL

2.1 Finite Systems

DTMC (Discrete-Time Markov Chains)

Time is considered in discrete steps, time progressing in fixed intervals, like seconds, minutes, or even customer arrivals. Transitions between states can only happen at these intervals.

Key features of DTMCs:

State Space: A finite set of states the system

$S = \{S_1, S_2, \dots, S_n\}$

Transition Probabilities: The probability of moving from one state to another in a single time step. These are represented by a transition matrix, where each entry shows the probability of transitioning from one state (row) to another (column).

It is denoted as P_{ij}, which represents the probability of moving from state S_i to state S_j.

Transition Matrix: A square matrix where each element P_{ij} represents the transition probability from state S_i to state S_j. For a DTMC, this matrix has the property that each row sums to 1.

$$P = \begin{bmatrix} P_{11} & P_{12} & \dots & P_{1n} \\ P_{21} & P_{22} & \dots & P_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & \dots & P_{nn} \end{bmatrix}$$

Markov Property: The probability of transitioning to the next state depends only on the current state, not the history of previous states.

$$P(X_{n+1}=j|X_n=i,X_{n-1}=i_{n-1},\dots,X_0=i_0)=P(X_{n+1}=j|X_n=i)$$

CTMC (Continuous-Time Markov Chains)

Time is considered continuous; Transitions can happen at any time within a given interval.

Key features of CTMCs:

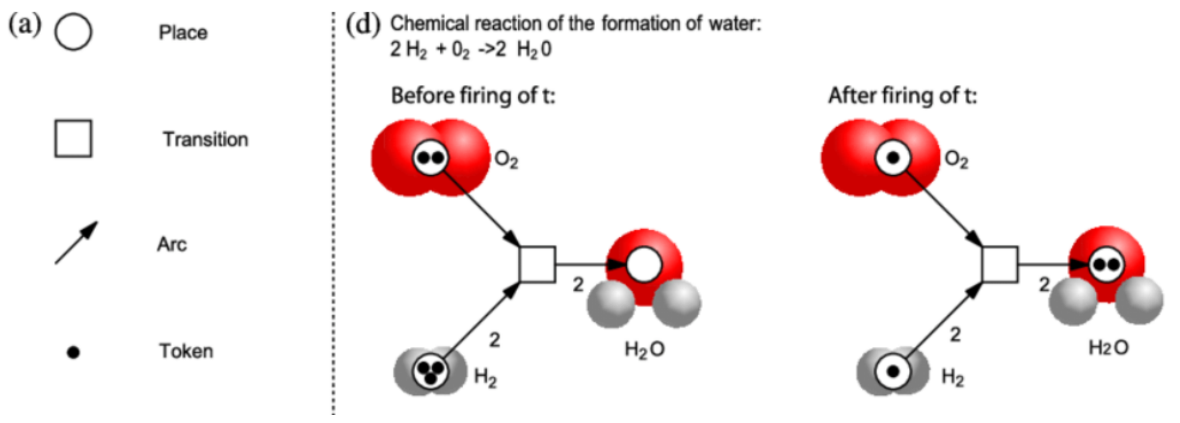
State Space: a finite set of states the system can be in.

Transition Rates: The rate (per unit time) at which the system leaves a particular state. These rates determine the exponential distribution of the time spent in each state.

Markov Property: probability of transitioning to the next state depends only on the current state, not the history.

3.PETRI NETS

A Petri Net is a graphical modelling tool used to describe the behaviour of concurrent systems, particularly distributed systems. It helps visualize the flow of information or resources within a system.



Components of a Petri Net:

- Places: Represented by circles, they denote the conditions or states the system can be in. They can hold tokens (black dots).
- Transitions: Represented by rectangles, they represent events or actions that cause the system to change from one state to another.
- Arcs: Directed arrows connecting places and transitions. They show the flow of tokens between them.
  - Input Arcs: Enter a transition, specifying the conditions required for it to fire (consume tokens).
  - Output Arcs: Leave a transition, specifying the state changes that occur when it fires (produces tokens).
- Tokens: Black dots residing in places, representing objects or resources flowing through the system.

How Petri Nets Work:

1. Initial Marking: The initial distribution of tokens in places defines the starting state of the system.
2. Enabled Transitions: A transition is enabled to fire if all its input places have at least one token.
3. Firing a Transition: When an enabled transition fires, it consumes a token from each input place and produces a token in each output place.
4. System Dynamics: The firing of transitions changes the distribution of tokens in places, representing the system's state changes.

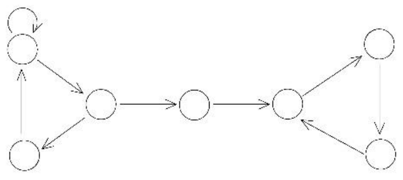
3.1 Petri Nets vs Markov Chains

Petri Nets:

- Strengths:
  - Visual Representation: Petri Nets visually depicting system flow and state changes. Places (circles) hold tokens (black dots) representing resources or

information, and transitions (rectangles) represent actions that move tokens between places. Arcs connect places and transitions, showing how tokens flow.
    - Concurrency Modeling: Well-suited for modeling concurrent systems where multiple actions can happen simultaneously.
    - Analysis Capabilities: Can be used to analyse properties like deadlocks (no further transitions can fire) and liveness (all reachable states are eventually visited).
- Weaknesses:
    - Scalability: Can become complex and unwieldy for very large systems with many elements.
    - Performance Analysis: Not as efficient for detailed performance evaluation compared to other techniques.

Markov Chains:



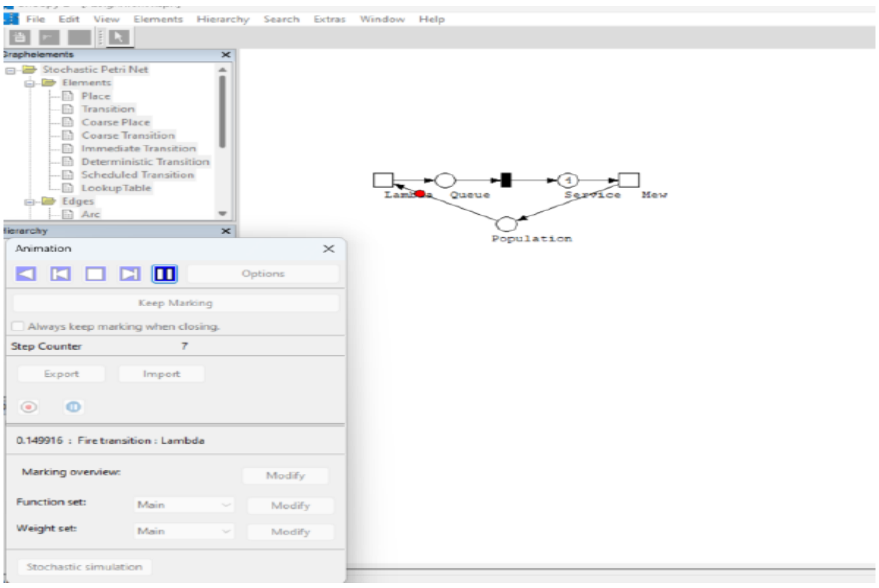
- Strengths:
    - Mathematical Analysis: Well-established mathematical framework allows for efficient calculation of probabilities of reaching specific states, average time spent in states, etc.
    - Performance Evaluation: Powerful tool for analyzing system performance metrics like throughput, latency, and resource utilization.
    - Scalability: Can handle larger systems more efficiently than Petri Nets due to their simpler structure.
- Weaknesses:
    - Limited Visualization: Relies on state transition diagrams, which can be less intuitive for complex systems compared to the visual flow of Petri Nets.
    - Concurrency Limitations: May struggle to model highly concurrent systems where multiple actions can occur simultaneously without specific ordering.

4. SNOOPY PETRI NETS

A Snoopy Petri net could refer to a custom Petri net model or visualization that incorporates Snoopy-themed elements or graphics. It could involve representing system states, transitions, and resource flows using Snoopy characters or illustrations, adding a playful and visually appealing touch to the Petri net representation.

The analysis and techniques used for a Snoopy Petri net would still adhere to the principles and methodologies of Petri nets. The focus would be on modeling concurrent processes, resource allocation, and state transitions, while leveraging the Snoopy theme for aesthetic purposes or to enhance engagement, particularly in educational or creative settings.

Snoopy petri net tool has been used to analyse different types of models. Here, four seasonal model has been analysed using snoopy tool.



Always keep marking when closing.

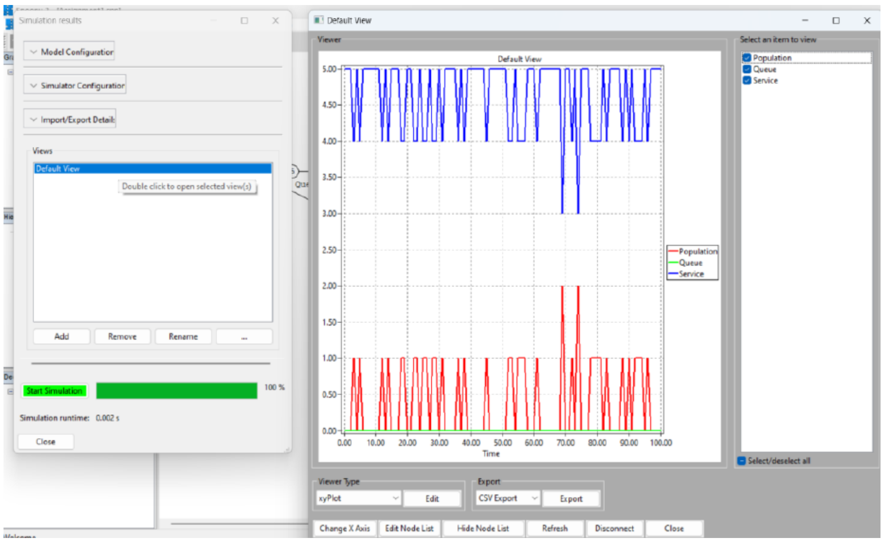
Step Counter 7

0.149916 : Fire transition : Lambda

Marking overview:

Function set: Main

Weight set: Main

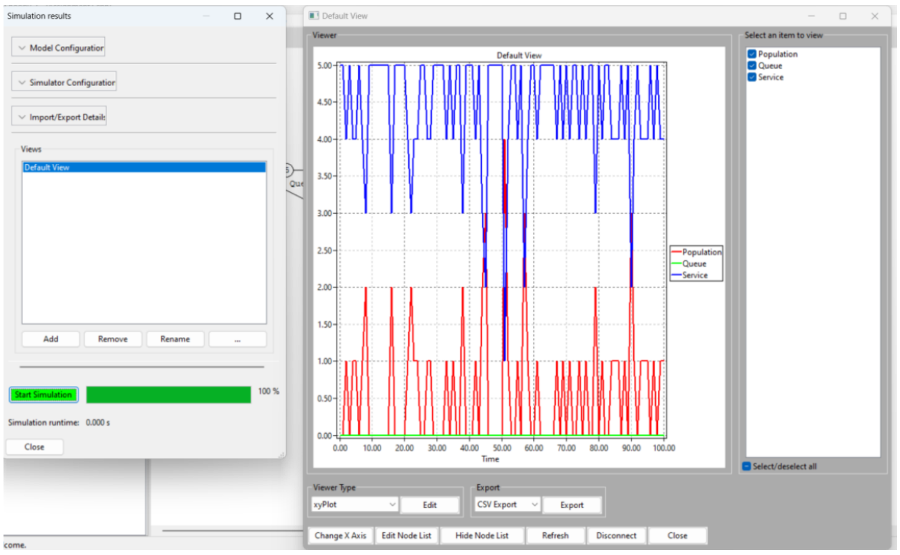


100 %

Simulation runtime: 0.002 s

Viewer Type: xyPlot

Export: CSV Export



100 %

Simulation runtime: 0.000 s

Viewer Type: xyPlot

Export: CSV Export

5. PRISM MODEL

PRISM is a widely used modeling language and tool for the specification, simulation, and formal verification of systems. It is particularly well-suited for modeling and analysing probabilistic and real-time systems. PRISM allows you to express system behaviors, properties, and requirements in a formal and mathematical manner. Here's an overview of PRISM models and their importance:

- Modeling Capabilities: PRISM provides a rich modeling language that allows you to describe the structure and behavior of a system. You can define discrete states, probabilistic transitions, real-time constraints, and various other components, depending on the nature of the system you are modeling.

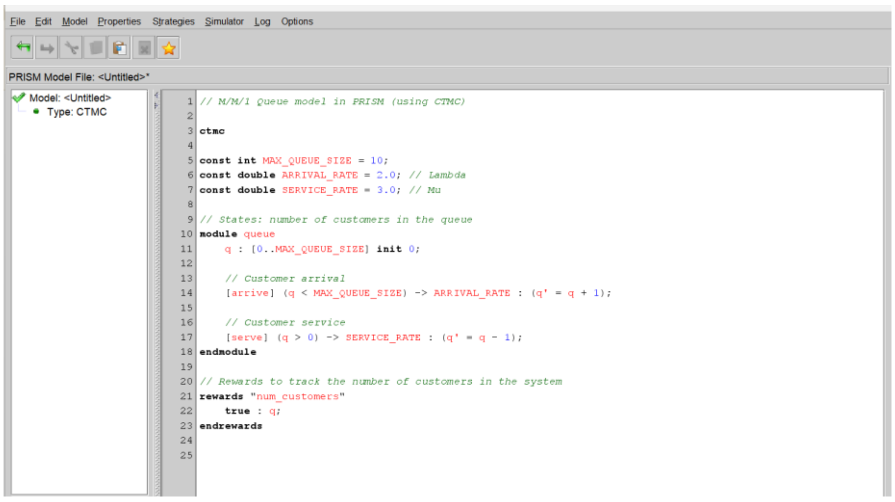
- System Specification: With PRISM, you can specify properties and requirements of the system using temporal logic formulas. These formulas allow you to express safety properties (e.g., "the system never reaches an unsafe state") and liveness properties (e.g., "eventually, a certain condition will always hold").

- Simulation and Analysis: PRISM offers simulation capabilities to explore the behavior of the system over time. You can generate traces, observe system properties, and evaluate performance measures. Simulation helps in gaining insights into system dynamics, identifying potential issues, and refining the model.

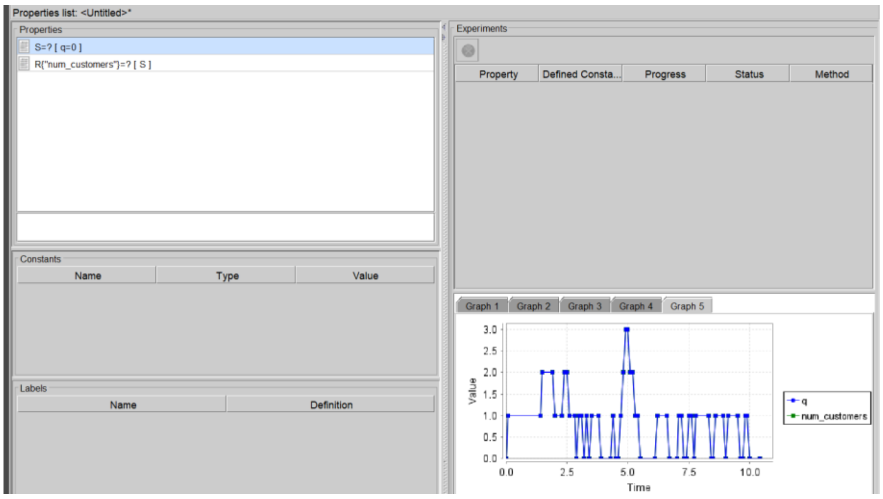
- Formal Verification: One of the key advantages of PRISM is its formal verification capabilities. It allows you to rigorously analyze and verify properties of the system. PRISM supports various formal verification techniques such as model checking, probabilistic model checking, and quantitative analysis. These techniques help in automatically checking whether specified properties hold or not, and they can provide counterexamples when properties are violated.

- Performance Evaluation: PRISM enables quantitative analysis of system performance, such as computing probabilities, expected values, and other performance measures. This is particularly useful for systems with probabilistic behavior or real-time constraints, where assessing performance is critical.

- Tool Integration: PRISM provides a user-friendly graphical interface, as well as a command-line interface, making it accessible to both novices and experts. Additionally, it offers various tools and utilities for model manipulation, visualization, and result analysis.


File Edit Model Properties Strategies Simulator Log Options
PRISM Model File: <Untitled>*
Model: <Untitled>
Type: CTMC

```
// M/M/1 Queue model in PRISM (using CTMC)

ctmc

const int MAX_QUEUE_SIZE = 10;
const double ARRIVAL_RATE = 2.0; // Lambda
const double SERVICE_RATE = 3.0; // Mu

// States: number of customers in the queue
module queue
    q : [0..MAX_QUEUE_SIZE] init 0;

    // Customer arrival
    [arrive] (q < MAX_QUEUE_SIZE) -> ARRIVAL_RATE : (q' = q + 1);

    // Customer service
    [serve] (q > 0) -> SERVICE_RATE : (q' = q - 1);
endmodule

// Rewards to track the number of customers in the system
rewards "num_customers"
    true : q;
endrewards
```



```
S=? [ q=0 ]
R{"num_customers"}=? [ S ]
```

5. Queueing Model with RabbitMQ

RabbitMQ is a powerful open-source message broker that implements a queuing model for asynchronous communication between applications. It acts as a central hub for messages, allowing producers (applications sending tasks) to decouple from consumers (applications processing tasks). This concept is ideal for building scalable and reliable systems.

Components of a Queueing Model with RabbitMQ:

- Producers: Applications that publish messages (tasks) to a queue in RabbitMQ. These messages can contain any data relevant to the task.
- Queue: A virtual buffer within RabbitMQ that stores messages in a First-In-First-Out (FIFO) order.
- Consumers: Applications that subscribe to a queue and receive messages for processing. RabbitMQ delivers messages to consumers one by one, ensuring fairness and preventing overloading.
- Exchange (Optional): An intermediary component that can route messages based on routing keys or patterns. This allows for more flexible message delivery to specific queues.

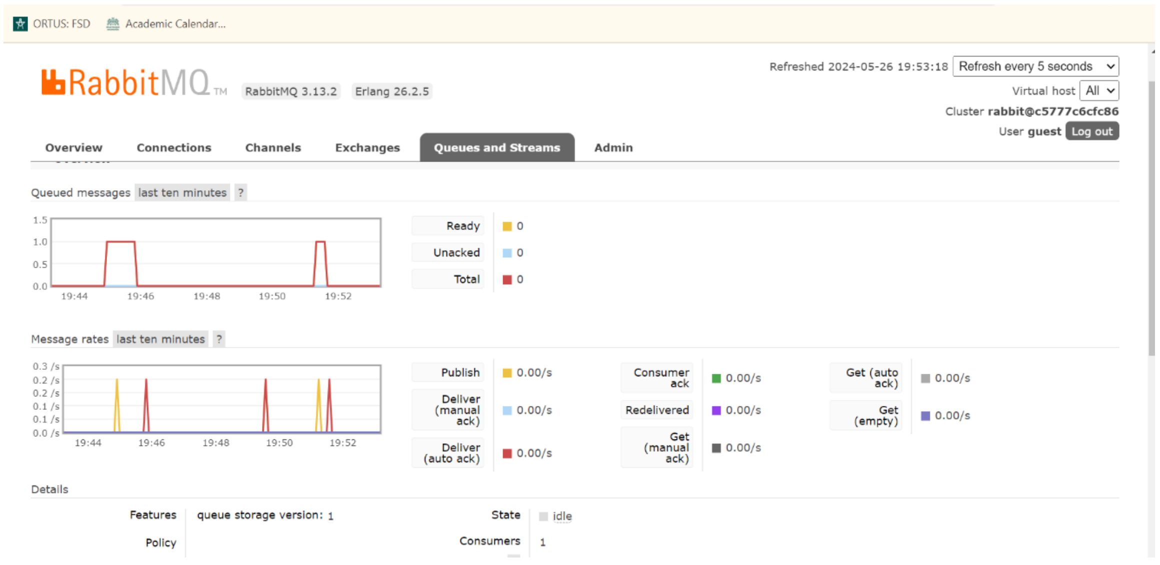
How it Works:

- Producers publish messages: Producers send messages containing task details to a specific queue in RabbitMQ.
- Messages are queued: RabbitMQ stores these messages in the queue in FIFO order.
- Consumers subscribe to queues: Consumers connect to RabbitMQ and subscribe to the queue(s) containing messages they can process.
- RabbitMQ delivers messages: RabbitMQ delivers messages from the queue to subscribed consumers, typically one at a time.
- Consumers process tasks: Consumers process the received message data, performing the intended tasks.

- Acknowledgment (Optional): Consumers can acknowledge receiving and processing a message, notifying RabbitMQ to remove it from the queue.

Benefits of using RabbitMQ for Queueing:

- Asynchronous Processing: Decouples producers from consumers, allowing independent scaling and development.
- Scalability: RabbitMQ can handle high volumes of messages efficiently, making it suitable for large-scale applications.
- Fault Tolerance: Messages are persisted in the queue, ensuring they are not lost if a consumer or producer fails temporarily.
- Load Balancing: RabbitMQ can distribute messages across multiple consumers for parallel processing, improving performance.



RabbitMQ 3.13.2 Erlang 26.2.5

Refreshed 2024-05-26 19:53:18 Refresh every 5 seconds

Virtual host All

Cluster **rabbit@c5777c6cfc86**

User **guest**

Queued messages last ten minutes ?

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates last ten minutes ?

Publish	🟨 0.00/s	Consumer ack	🟩 0.00/s	Get (auto ack)	⬜ 0.00/s
Deliver (manual ack)	🟦 0.00/s	Redelivered	🟪 0.00/s	Get (empty)	🟪 0.00/s
Deliver (auto ack)	🟥 0.00/s	Get (manual ack)	⬛ 0.00/s		

Details

Features	queue storage version: 1	State	idle
Policy		Consumers	1

5. REFERENCES

Adan, I. and Resing, J., 2002. Queueing theory.

Akimaru, H. and Kawashima, K., 2012. *Teletraffic: theory and applications*. Springer Science & Business Media.

Basharin, G.P., Samouylov, K.E.E., Yarkina, N.V. and Gudkova, I.A., 2009. A new stage in mathematical teletraffic theory. *Automation and Remote Control*, *70*, pp.1954-1964.

Ching, W.K. and Ng, M.K., 2006. Markov chains. *Models, algorithms and applications*.

Cooper, R.B., 1981, January. Queueing theory. In *Proceedings of the ACM'81 conference* (pp. 119-122).

Norris, J.R., 1998. *Markov chains* (No. 2). Cambridge university press.

Peterson, J.L., 1977. Petri nets. *ACM Computing Surveys (CSUR)*, *9*(3), pp.223-252.

Reisig, W., 2012. *Petri nets: an introduction* (Vol. 4). Springer Science & Business Media.

Revuz, D., 2008. *Markov chains*. Elsevier.

Sztrik, J., 2012. Basic queueing theory. *University of Debrecen, Faculty of Informatics*, *193*, pp.60-67.