# Introduction

Blockchain technology, first proposed by Nakamoto in 2008, has rapidly evolved into a foundational infrastructure for digital trust and decentralized systems. As a form of distributed ledger technology (DLT), blockchain is characterized by decentralization, transparency, and immutability, making it particularly suited for secure data storage and peer-to-peer value transfer. Each block contains a hash of the previous block, a timestamp, and a collection of transactions, forming an append-only chain that is nearly impossible to tamper with. Originally designed to support cryptocurrencies like Bitcoin, blockchain has since been adopted in numerous domains including finance, healthcare, media, logistics, and energy.

One of the most pressing technical challenges in today’s blockchain landscape is interoperability, that is the ability for independently operated blockchain networks to communicate and exchange assets or information. This challenge has spurred the development of cross-chain bridges, which act not as physical connections but rather as a collection of protocols and mechanisms that allow heterogeneous blockchains to interoperate. These bridges enable asset transfers, data exchange, and coordinated smart contract execution between public chains, consortium chains, and private chains, which are otherwise isolated due to architectural and consensus differences. Similar concerns have also emerged in layered edge-cloud systems, where performance evaluation frameworks are often built on queueing-theoretic modeling.

To address the performance analysis of blockchain systems, a number of prior works have employed queueing theory to model system dynamics under realistic assumptions. For instance, researchers have simulated edge-cloud offloading networks using M/G/1 and M/G/m models to assess task delays and system throughput in blockchain-based layered environments [1]. Another study applied M/M/n/L queues to model transaction processing and block generation in Bitcoin, demonstrating how queue length and block production rates impact performance [2].

In more structured systems such as Hyperledger Fabric, a queueing network model was proposed to divide the consensus process into execution, ordering, and validation stages, enabling analysis of latency across phases [3]. Other researchers combined queueing models with multidimensional Markov chains to analyze PBFT-based consensus systems with repairable voting nodes, quantifying system reliability and throughput under dynamic conditions [4]. Performance bottlenecks in Fabric’s architecture were also identified via benchmarking, helping guide practical optimizations [5].

Beyond consensus mechanics, theoretical models have captured the growth dynamics and reward allocation strategies in multi-mining pool environments such as Ethereum. One study introduced a tree-based blockchain structure and renewal reward theory to model stale and uncle blocks [6]. Priority-based transaction handling has also been modeled using non-preemptive limited-priority queues, illustrating the performance tradeoffs between high- and low-priority transaction classes [7]. To tackle intractable steady-state distributions in complex systems, another approach applied the maximum entropy principle to estimate probabilities based on observable statistics, providing flexible approximations without strong distributional assumptions [8].

In response to the lack of simple yet effective models for analyzing cross-chain systems, this thesis draws on examples from [9] and [10] to develop a queueing model for cross-chain transaction flows. The model abstracts the system into two interconnected queues: the customer queue, where transactions wait to be selected for block formation, and the consensus queue, where blocks undergo validation and finalization. To better capture realistic user behavior, the model considers multiple user classes with non-preemptive limited priority and partial batch service, as well as system states that alternate between ON and OFF. User impatience is also incorporated to reflect transaction abandonment in highly congested environments. These dynamics are analyzed across four scenarios: (1) Single-Class Customers without Impatience, (2) Two-Class Customers without Impatience, (3) Single-Class Customers with Impatience, and (4) Two-Class Customers with Impatience.

To evaluate the system's steady-state behavior under complex configurations, this thesis adopts a numerical iteration method based on the balance equations of the underlying Markov chain. A simulation is also performed for validation. This approach enables the computation of key performance metrics.

The remainder of this thesis is organized as follows. Chapter 2 introduces the system model, detailing the cross-chain process structure and the queuing assumptions used in this study. Chapter 3 presents the analytical model, which formalizes the system behavior under various parameter settings and derives key performance metrics. Chapter 4 describes the simulation model, providing implementation details and simulation strategies used to validate the analytical results. Chapter 5 reports the numerical results and performance evaluation across the proposed scenarios. Finally, Chapter 6 concludes the thesis and outlines potential directions for future work.

# System Model

We aim to study four blockchain scenarios, each involving two queues of finite capacity: the **customer queue** and the **consensus queue**. The maximum capacity of the customer queue is denoted by , while the consensus queue, which represents the block currently undergoing consensus, has a capacity of . Customers in the system first wait in the customer queue for the block generation process. Once this process is complete, a group of customers is moved to the consensus queue to undergo the consensus process.

The blocking generation process is based on a **partial batch service mechanism**. When the consensus queue becomes idle, if there are more than customers in the customer queue, the first customers are selected and moved to the consensus queue. If there are or fewer customers waiting, all of them are transferred instead. After the consensus process finishes, regardless of whether the result is successful, all customers in the consensus queue leave the system.

Additionally, the system may switch between **ON** and **OFF** periods. During the OFF period, caused by events such as hacking attacks or connection failures due to environmental factors, both the block generation and consensus processes are suspended. Once the system returns to the ON period, these processes resume as usual.

## Scenario 1: Single-Class Customers without Impatience

In the first scenario, we assume that there is only a single class of customers in the system, and the queueing discipline for the customer queue is First-Come-First-Served (FCFS). It is noted that if the consensus queue is empty, at most customers can wait in the customer queue for the block generation process. On the other hand, if the consensus queue is not empty, the maximum number of customers allowed in the customer queue is reduced to .

## Scenario 2: Two-Class Customers without Impatience

In the second scenario, we assume that there are two classes of customers in the system: high-priority customers and low-priority customers. Customers with the same priority are served according to the First-Come-First-Served (FCFS) discipline. Note that for high-priority customers are always placed ahead of low-priority customers in the customer queue. Customers with different priorities are served according to the non-preemptive priority discipline. Specifically, high-priority customers are always placed ahead of low-priority customers in the customer queue and the consensus process of the low-priority customers cannot be interrupted. Note that for high-priority customers, the maximum capacity of the customer queue is when the consensus queue is idle, and when it is not idle. On the other hand, for low-priority customers, the maximum capacity of the customer queue is always , regardless of whether the consensus queue is idle or not.

## Scenario 3: Single-Class Customers with Impatience

The third scenario considers a single class of customers with impatience. Customers still follow the First-Come-First-Served (FCFS) discipline, but they may leave the system while waiting in the customer queue if their waiting time exceeds their patience threshold. Once a customer enters the consensus queue, impatience is no longer considered. It is noted that if the consensus queue is empty, at most customers can wait in the customer queue for the block generation process. On the other hand, if the consensus queue is not empty, the maximum number of customers allowed in the customer queue is reduced to .

## Scenario 4: Two-Class Customers with Impatience

In the fourth scenario, we again consider two classes of customers with impatience —high-priority and low-priority. Customers of the same priority follow the First-Come-First-Served (FCFS) discipline, and customers of different priorities follow the non-preemptive discipline, i.e., high-priority customers are given precedence over low-priority customers in the queue and the consensus process of the low-priority customers cannot be interrupted. Customers from both priority classes may leave the queue if they wait too long. Each priority class may have its own impatience rate. Impatience is no longer relevant once customers enter the consensus queue. Note that for high-priority customers, the maximum capacity of the customer queue is when the consensus queue is idle, and when it is not idle. On the other hand, for low-priority customers, the maximum capacity of the customer queue is always , regardless of whether the consensus queue is idle or not.

# Analytical Model

In this chapter, we are going to present four different scenarios for modeling blockchain-based systems: (1) Single-Class Customers without Impatience, (2) Two-Class Customers without Impatience, (3) Single-Class Customers with Impatience, and (4) Two-Class Customers with Impatience. Each of these scenarios is built upon a queuing-based abstraction of the blockchain process and aims to capture distinct behavioral features related to customer priority and abandonment. In all cases, as shown in Figure 3‑1, the system is composed of two queues with limited capacity: the customer queue, which temporarily holds users before block generation, and the consensus queue, which represents the stage where users participate in the consensus protocol after being grouped into a block.

Assume that the arrivals of customers follow a Poisson process, where the arrival rate is denoted by λ. In the multi-class scenarios, we further distinguish between high-priority and low-priority customers, whose respective arrival rates are and , so that the total arrival rate satisfies . After arriving at the customer queue, users wait for the block generation process, which occurs at a rate of (or and in the two-class case). Once a block is formed, a group of users is transferred to the consensus queue, where the consensus process is carried out at a service rate denoted by (or and depending on customer class).

In scenarios that involve impatience, we assume that customers may abandon the system while waiting in the customer queue if their waiting time exceeds a certain threshold. The impatience threshold is modeled as an exponential random variable with a rate for single-class users, and rates and for high-priority and low-priority users, respectively. Once a customer enters the consensus queue, impatience is no longer considered. In addition, we consider the operational reliability of the system by incorporating the possibility of the system state alternating between ON and OFF periods. During ON periods, both block generation and consensus operations are allowed to proceed, while during OFF periods, these operations are suspended. The durations of both ON and OFF periods are exponentially distributed. The transition rates between the two states are given by (ON to OFF) and (OFF to ON) respectively.

一張含有 黑色, 黑暗 的圖片

AI 產生的內容可能不正確。

Figure 3‑1

We assume the queueing discipline is First-Come-First-Served (FCFS) for customers of the same class. In the two-class scenarios, customers are additionally scheduled under a non-preemptive priority rule, in which high-priority customers are placed ahead of low-priority ones in the customer queue, but once a customer enters the consensus queue, their service cannot be interrupted. These settings allow us to examine the interplay between system structure, service prioritization, impatience-driven abandonment, and queue dynamics in a blockchain-inspired environment. The parameters used in different scenarios are shown in Table 3.1

|  |  |  |
| --- | --- | --- |
| Description | Single-class | Two-class |
| Arrival rate |  |  |
|  |
| BLock generation rate |  |  |
|  |
| Consensus rate |  |  |
|  |
| Impatient rate |  |  |
|  |
| Transition rate (ON to OFF) |  |  |
| Transition rate (OFF to ON) |  |  |

Table 3.1 The parameters used in different scenarios

## Scenario 1: Single-Class Customer without Impatience

In this scenario, we consider a single-class customer system without impatience, where arrivals follow a Poisson process and customers are served based on the First-Come-First-Served (FCFS) discipline. The service process is divided into block generation and consensus phases, and the system switches between ON and OFF states, affecting service availability.

Assume that the arrivals of customers follow a Poisson process, where the arrival rate is denoted by λ. After arriving at the customer queue, users wait for the block generation process, which occurs at a rate of . Each block is generated according to the partial batch policy, i.e., each block can contain 1 to customers. Once a block is formed, a group of users is transferred to the consensus queue, where the consensus process is carried out at a service rate denoted by .

In addition, we consider the operational reliability of the system by incorporating the possibility of the system state alternating between ON and OFF periods. During ON periods, both block generation and consensus operations are allowed to proceed, while during OFF periods, these operations are suspended. The durations of both ON and OFF periods are exponentially distributed. The transition rates from ON to OFF and from OFF to ON are given by and , respectively.

### State Balance Equations

The system under consideration is described as a three-dimensional Markov chain with state denoted by , where denotes the number of customers in the customer queue, denotes the number of customers in the consensus queue, and denotes the system state. When , the maximum number of customers in the customer queue is . When , meaning that the consensus queue is occupied, the maximum number of customers allowed in the customer queue is reduced to . The system state indicates that the system is in the ON state, where customers can enter the customer queue and both block generation and consensus operations can proceed. On the other hand, when , the system is in the OFF state, during which only customer arrivals to the queue are permitted, while block generation and consensus are suspended. The state space can be denoted as follows:

Hence, the total number of feasible states is given by:

For example, if and , the number of feasible states is 1182. The steady state probability of state is denoted as . In this scenario, the feasible states can be categorized into 16 distinct cases, as described below.

#### System off,

#### System on,

Given the large number of equations presented above, it is impractical to illustrate all the corresponding state transition diagrams. Therefore, we focus on a relatively complex case, specifically Case 11, as a representative example, shown in Figure 3‑2.

一張含有 文字, 螢幕擷取畫面, 圓形, 字型 的圖片

AI 產生的內容可能不正確。

Figure 3‑2 The state transition diagram of Case 11:

### Iterative Algorithm

We use the iterative algorithm provided below, and perform calculations on the state balance equations until they converge, allowing us to determine the steady-state distribution of the system.

###### Iterative algorithm:

**Step 1**: Initialize for all , where is the total number of feasible states.

**Step 2**: Substitute into the balance equations from Case 1 to Case 16 to find , .

**Step 3**: Normalize , .

**Step 4**: If , then stop the iteration. Otherwise, set , , and return to **Step 2**.

In our analysis, the convergence threshold is set to , and the algorithm typically converges after about 75 iterations.

### Performance Measure

After obtaining the steady-state probabilities through the iterative algorithm, we proceed to compute several performance metrics to evaluate the effectiveness of the system.

First of all, the average number of customers in the whole system, denoted by , is given by:

Second, the average number of customers in customer queue, denoted by , is given by:

Third, the average number of customers in consensus queue, denoted by , is given by:

Fourth, the blocking probability of the system, denoted by , is given by:

Fifth, the throughput of the system, denoted by , is given by:

Sixth, the average waiting time in the system, denoted by , is given by:

Seventh, the average waiting time in the customer queue, denoted by , is given by:

Eighth, the average waiting time in the consensus queue, denoted by , is given by:

Finally, the average number of blocks participating in the consensus process, denoted by , is given below.

## Scenario 2: Two-Class Customer without Impatience

In this scenario, we consider a two-class customer system without impatience, where the arrivals of high-priority and low-priority customers follow the independent Poisson processes, with arrival rates denoted by and , respectively. Customers are served based on the non-preemptive priority discipline, where high-priority customers are placed ahead of low-priority ones in the queue, but ongoing service cannot be interrupted.

The service process is divided into block generation and consensus phases. After arriving at the customer queue, users wait for the block generation process, which occurs at a rate of and for high-priority and low-priority customers, respectively. Each block is generated according to the partial batch policy, i.e., each block can contain 1 to customers of the same policy class. Once a block is formed, it is transferred to the consensus queue, where the consensus process is carried out at the service rate denoted by and for high-priority and low-priority customers, respectively.

In addition, we consider the operational reliability of the system by incorporating the possibility of the system state alternating between ON and OFF periods. During ON periods, both block generation and consensus operations are allowed to proceed, while during OFF periods, these operations are suspended. The durations of both ON and OFF periods are exponentially distributed. The transition rates from ON to OFF and from OFF to ON are given by and , respectively.

### State Balance Equations

The system under consideration is described as a five-dimensional Markov chain denoted by , where and represent the number of high-priority and low-priority customers in the customer queue, respectively. and represent the number of high-priority and low-priority customers in the consensus queue, respectively. And denotes the system state. When the consensus queue is empty (i.e., and ), the maximum number of customers allowed in the customer queue is , implying that . However, when the consensus queue is occupied (i.e., or ), the maximum number of customers in the customer queue is reduced to , and thus .

Customers are scheduled according to the non-preemptive priority discipline, where high-priority customers are always placed ahead of low-priority ones in the queue, but service already in progress cannot be interrupted. When a block is generated, it must contain customer(s) of only one priority class, and is transferred into the consensus queue as a batch for processing without preemption.

The system state indicates that the system is in the ON state, where customers can enter the customer queue and both block generation and consensus operations can proceed. On the other hand, when , the system is in the OFF state, during which only customer arrivals to the queue are permitted, while block generation and consensus are suspended. The state space can be denoted as follows:

Hence, the total number of feasible states is given by:

For example, if and , the number of feasible states is 22524. The steady state probability of state is denoted as . In this scenario, the feasible states can be categorized into 99 distinct cases, as described below.

Given the large number of equations presented above, it is impractical to illustrate all the corresponding state transition diagrams. Therefore, we focus on a relatively complex case, specifically Case 45, as a representative example, shown in Figure 3‑3.

**一張含有 螢幕擷取畫面, 圓形, 設計, 圖解 的圖片

AI 產生的內容可能不正確。**

Figure 3‑3 The state transition diagram of Case 45:

### Iterative Algorithm

We use the iterative algorithm provided below, and perform calculations on the state balance equations until they converge, allowing us to determine the steady-state distribution of the system.

#### Iterative algorithm:

**Step 1**: Initialize for all , where is the total number of feasible states.

**Step 2**: Substitute into the balance equations from Case 1 to Case 16 to find , .

**Step 3**: Normalize , .

**Step 4**: If , then stop the iteration. Otherwise, set , , and return to **Step 2**.

In our analysis, the convergence threshold is set to , and the algorithm typically converges after about 80 iterations.

### Performance Measure

After obtaining the steady-state probabilities through the iterative algorithm, we proceed to compute several performance metrics to evaluate the effectiveness of the system.

First of all, the average number of high-priority and low-priority customers in the whole system, denoted by and , respectively, is given by:

The average number of customers in the whole system, denoted by , is given by:

Second, the average number of high-priority and low-priority customers in customer queue, denoted by and , respectively, is given by:

The average number of customers in customer queue, denoted by , is given by:

Third, the average number of high-priority and low-priority customers in consensus queue, denoted by and , respectively, is given by:

The average number of customers in consensus queue, denoted by , is given by:

Fourth, the blocking probability of high-priority and low-priority customers in the system, denoted by and , respectively, is given by:

The blocking probability of the system, denoted by , is given by:

Fifth, the throughput of high-priority and low-priority customers in the system, denoted by and , respectively, is given by:

The throughput of the system, denoted by , is given by:

Sixth, the average waiting time of the high-priority and low-priority customers in the system, denoted by and , respectively, is given by:

The average waiting time in the system, denoted by , is given by:

Seventh, the average waiting time of the high-priority and low-priority customers in the customer queue, denoted by and , respectively, is given by:

The average waiting time in the customer queue, denoted by , is given by:

Eighth, the average waiting time of the high-priority and low-priority customers in the consensus queue, denoted by and , respectively, is given by:

The average waiting time in the consensus queue, denoted by , is given by:

Finally, the average number of high-priority and low-priority blocks participating in the consensus process, denoted by and , is given by:

The average number of customers participating in the consensus process within a block, denoted by , is given by:

## Scenario 3: Single-Class Customer with Impatience

In this scenario, we consider a single-class customer system with impatience, where arrivals follow a Poisson process and customers are served based on the First-Come-First-Served (FCFS) discipline. The service process is divided into block generation and consensus phases, and the system switches between ON and OFF states, affecting service availability.

Assume that the arrivals of customers follow a Poisson process, where the arrival rate is denoted by λ. After arriving at the customer queue, users wait for the block generation process, which occurs at a rate of . Each block is generated according to the partial batch policy, i.e., each block can contain 1 to customers. Once a block is formed, a group of users is transferred to the consensus queue, where the consensus process is carried out at a service rate denoted by .

To account for customer impatience, we assume that customers in the customer queue may leave the system if they wait too long. The patience time is assumed to follow an exponential distribution, with impatience rates denoted by . Customers in the consensus queue are assumed to be committed and will not abandon once service has started.

In addition, we consider the operational reliability of the system by incorporating the possibility of the system state alternating between ON and OFF periods. During ON periods, both block generation and consensus operations are allowed to proceed, while during OFF periods, these operations are suspended. The durations of both ON and OFF periods are exponentially distributed. The transition rates from ON to OFF and from OFF to ON are given by and , respectively.

### State Balance Equations

The system under consideration is described as a three-dimensional Markov chain with state denoted by , where denotes the number of customers in the customer queue, denotes the number of customers in the consensus queue, and denotes the system state. When , the maximum number of customers in the customer queue is . When , meaning that the consensus queue is occupied, the maximum number of customers allowed in the customer queue is reduced to .

In this scenario, customers may abandon the customer queue if their waiting time exceeds a certain threshold. The patience time is assumed to follow an exponential distribution with rate , and abandonment occurs only while the customer is waiting in the customer queue.

The system state indicates that the system is in the ON state, where customers can enter the customer queue and both block generation and consensus operations can proceed. On the other hand, when , the system is in the OFF state, during which only customer arrivals to the queue are permitted, while block generation and consensus are suspended. The state space can be denoted as follows:

Hence, the total number of feasible states is given by:

For example, if and , the number of feasible states is 1182. The steady state probability of state is denoted as . In this scenario, the feasible states can be categorized into 16 distinct cases, as described below.

#### System off,

#### System on,

Given the large number of equations presented above, it is impractical to illustrate all the corresponding state transition diagrams. Therefore, we focus on a relatively complex case, specifically Case 12, as a representative example, shown in Figure 3‑4.

一張含有 文字, 圓形, 螢幕擷取畫面, 字型 的圖片

AI 產生的內容可能不正確。

Figure 3‑4 The state transition diagram of Case 12:

### Iterative Algorithm

We use the iterative algorithm provided below, and perform calculations on the state balance equations until they converge, allowing us to determine the steady-state distribution of the system.

#### Iterative algorithm:

**Step 1**: Initialize for all , where is the total number of feasible states.

**Step 2**: Substitute into the balance equations from Case 1 to Case 16 to find , .

**Step 3**: Normalize , .

**Step 4**: If , then stop the iteration. Otherwise, set , , and return to **Step 2**.

In our analysis, the convergence threshold is set to , and the algorithm typically converges after about 72 iterations.

### Performance Measure

After obtaining the steady-state probabilities through the iterative algorithm, we proceed to compute several performance metrics to evaluate the effectiveness of the system.

First of all, the average number of customers in the whole system, denoted by , is given by:

Second, the average number of customers in customer queue, denoted by , is given by:

Third, the average number of customers in consensus queue, denoted by , is given by:

Fourth, the blocking probability of the system, denoted by , is given by:

Fifth, the impatient probability of the system, denoted by , is given by:

Sixth, the throughput of the system, denoted by , is given by:

Seventh, the average waiting time in the customer queue, denoted by , is given by:

Eighth, the average waiting time in the consensus queue, denoted by , is given by:

Ninth, the average waiting time in the system, denoted by , is given by:

Finally, the average number of blocks participating in the consensus process, denoted by , is given below.

## Scenario 4: Two-Class Customer with Impatience

In this scenario, we consider a two-class customer system without impatience, where the arrivals of high-priority and low-priority customers follow the independent Poisson processes, with arrival rates denoted by and , respectively. Customers are served based on the non-preemptive priority discipline, where high-priority customers are placed ahead of low-priority ones in the queue, but ongoing service cannot be interrupted.

The service process is divided into block generation and consensus phases. After arriving at the customer queue, users wait for the block generation process, which occurs at a rate of and for high-priority and low-priority customers, respectively. Each block is generated according to the partial batch policy, i.e., each block can contain 1 to customers of the same policy class. Once a block is formed, it is transferred to the consensus queue, where the consensus process is carried out at the service rate denoted by and for high-priority and low-priority customers, respectively.

To account for customer impatience, we assume that customers in the customer queue may leave the system if they wait too long. The patience time is assumed to follow an exponential distribution, with impatience rates​ and ​ for high-priority and low-priority customers, respectively. Customers in the consensus queue are assumed to be committed and will not abandon once service has started.

In addition, we consider the operational reliability of the system by incorporating the possibility of the system state alternating between ON and OFF periods. During ON periods, both block generation and consensus operations are allowed to proceed, while during OFF periods, these operations are suspended. The durations of both ON and OFF periods are exponentially distributed. The transition rates from ON to OFF and from OFF to ON are given by and , respectively.

### State Balance Equations

The system under consideration is described as a five-dimensional Markov chain denoted by , where and represent the number of high-priority and low-priority customers in the customer queue, respectively. and represent the number of high-priority and low-priority customers in the consensus queue, respectively. And denotes the system state. When the consensus queue is empty (i.e., and ), the maximum number of customers allowed in the customer queue is , implying that . However, when the consensus queue is occupied (i.e., or ), the maximum number of customers in the customer queue is reduced to , and thus .

Customers are scheduled according to the non-preemptive priority discipline, where high-priority customers are always placed ahead of low-priority ones in the queue, but service already in progress cannot be interrupted. When a block is generated, it must contain customer(s) of only one priority class, and is transferred into the consensus queue as a batch for processing without preemption.

The system state indicates that the system is in the ON state, where customers can enter the customer queue and both block generation and consensus operations can proceed. On the other hand, when , the system is in the OFF state, during which only customer arrivals to the queue are permitted, while block generation and consensus are suspended. The state space can be denoted as follows:

Hence, the total number of feasible states is given by:

For example, if and , the number of feasible states is 22524. The steady state probability of state is denoted as . In this scenario, the feasible states can be categorized into 99 distinct cases, as described below.

Given the large number of equations presented above, it is impractical to illustrate all the corresponding state transition diagrams. Therefore, we focus on a relatively complex case, specifically Case 45, as a representative example, shown in Figure 3‑5.

一張含有 文字, 螢幕擷取畫面, 圓形, 圖解 的圖片

AI 產生的內容可能不正確。

Figure 3‑5 The state transition diagram of Case 45:

### Iterative Algorithm

We use the iterative algorithm provided below, and perform calculations on the state balance equations until they converge, allowing us to determine the steady-state distribution of the system.

#### Iterative algorithm:

**Step 1**: Initialize for all , where is the total number of feasible states.

**Step 2**: Substitute into the balance equations from Case 1 to Case 16 to find , .

**Step 3**: Normalize , .

**Step 4**: If , then stop the iteration. Otherwise, set , , and return to **Step 2**.

In our analysis, the convergence threshold is set to , and the algorithm typically converges after about 80 iterations.

### Performance Measure

After obtaining the steady-state probabilities through the iterative algorithm, we proceed to compute several performance metrics to evaluate the effectiveness of the system.

First of all, the average number of high-priority and low-priority customers in the whole system, denoted by and , respectively, is given by:

The average number of customers in the whole system, denoted by , is given by:

Second, the average number of high-priority and low-priority customers in customer queue, denoted by and , respectively, is given by:

The average number of customers in customer queue, denoted by , is given by:

Third, the average number of high-priority and low-priority customers in consensus queue, denoted by and , respectively, is given by:

The average number of customers in consensus queue, denoted by , is given by:

Fourth, the blocking probability of high-priority and low-priority customers in the system, denoted by and , respectively, is given by:

The blocking probability of the system, denoted by , is given by:

Fifth, the impatient probability of high-priority and low-priority customers in the system, denoted by and , respectively, is given by:

The impatient probability of the system, denoted by , is given by:

Sixth, the throughput of high-priority and low-priority customers in the system, denoted by and , respectively, is given by:

The throughput of the system, denoted by , is given by:

Seventh, the average waiting time of the high-priority and low-priority customers in the customer queue, denoted by and , respectively, is given by:

The average waiting time in the customer queue, denoted by , is given by:

Eighth, the average waiting time of the high-priority and low-priority customers in the consensus queue, denoted by and , respectively, is given by:

The average waiting time in the consensus queue, denoted by , is given by:

Ninth, the average waiting time of the high-priority and low-priority customers in the system, denoted by and , respectively, is given by:

The average waiting time in the consensus queue, denoted by is given by:

Finally, the average number of high-priority and low-priority blocks participating in the consensus process, denoted by and , is given by:

The average number of customers participating in the consensus process within a block, denoted by , is given by:

# Simulation Model

In this chapter, we present a detailed explanation of four simulation scenarios, each corresponding to a different configuration of blockchain queueing behavior. These scenarios are designed to reflect the structural and behavioral differences introduced by customer priority and impatience. All simulation models incorporate both First-Come-First-Served (FCFS) and non-preemptive priority disciplines, as appropriate to each case.

The first simulation model represents a single-class customer system without impatience. In this case, customers arrive and are served strictly in arrival order, and no abandonment occurs even if the waiting time is long. The second simulation model introduces two customer classes, high-priority and low-priority, handled with non-preemptive scheduling but without impatience. High-priority customers are always placed ahead of low-priority customers in the queue, but service-in-progress of any customer cannot be interrupted.

The third simulation model considers a single-class system with impatience, where customers may abandon the queue if they wait too long. This adds a stochastic abandonment dynamic based on patience thresholds. The final simulation model incorporates both customer priority and impatience. High-priority and low-priority customers are managed with non-preemptive priority, and both classes have their own impatience rates. This complex setting allows us to examine how prioritization and abandonment interact in a congested blockchain environment.

In all cases, the simulation captures system dynamics under partial batch service, and models ON/OFF channel behavior, where the service is suspended during OFF periods. These scenarios are simulated independently to compare their performance metrics, including throughput, queue lengths, waiting time, blocking probability, and, where applicable, abandonment probability.

## Scenario 1: Single-Class Customer without Impatience

In this simulation model, we consider a blockchain system that handles a single class of users, where customers arrive according to a Poisson process and are served under the First-Come-First-Served (FCFS) discipline.

The system consists of two queues: the customer queue, where users wait for block generation, and the consensus queue, where users participate in the consensus process after being grouped into a block. Block generation follows a partial batch service policy, allowing 1 to users to form a block. Once a block is formed, it is transferred to the consensus queue. Upon completion of the consensus process, all users in the block exit the system.

During the OFF state, caused by interruptions such as attacks or connectivity issues, both block generation and consensus processes are suspended, although new users may still arrive and be admitted. During the ON state, all services resume normally. To preserve system integrity, a constraint is imposed on the maximum number of customers allowed in the customer queue: when the consensus queue is empty, up to users may wait; otherwise, the limit is reduced to .

Since customer impatience is not considered in this model, all customers remain in the queue until they are served. This makes the first scenario a baseline case for performance comparison, focusing on metrics such as throughput, average queue length, and system utilization under a stable environment with uninterrupted user service.

### Main program

The main program executes a series of steps to simulate the blockchain queuing system, illustrated in Figure 4‑1. At the beginning of each simulation run, all relevant variables are initialized. This includes resetting statistical parameters, setting the next block generation time and next departure time to infinity, marking the system status as ON, initializing the block generation status as idle, and setting the customer queue limit to .

Next, the system parameters are configured. These include the maximum customer queue capacity (), the maximum number of users per block (), the arrival rate (), the block generation rate (), the consensus (block departure) rate (), and the ON/OFF switching rates ( and ) for the system channel.

The program generates the next arrival time and channel switch time using exponential random variables based on the corresponding system parameters. During the simulation, it compares the scheduled times of four events and selects the earliest event to execute its corresponding subprogram.

Finally, a while loop is used to repeat the simulation until a predefined number of customer arrivals has been reached. Once this condition is met, the simulation terminates and the performance statistics are output.

### Arrival Subprogram

Figure 4‑2 illustrated the flow chart of the arrival subprogram, simulates the arrival of a new customer to the system. Upon invocation, the total number of arrivals is incremented, and the simulation time is updated to the scheduled arrival time. The time for the next arrival is then scheduled using an exponential interarrival time generated with the arrival rate . Then, the area calculation function is invoked to update all time-averaged statistics based on the elapsed time since the last event.

Next, the system checks whether the customer queue has reached its capacity limit.

* If the queue is full, the arriving customer is rejected, and the number of rejections is incremented.
* If the queue is not full, the arriving customer is admitted. In this case, both the number of customers in the system and in the queue are incremented, and the customer's arrival time is recorded in the queue log.

Finally, the system determines whether to initiate block generation:

* If the channel status is in ON state, and block generator is idle, and this customer is the only one in the queue, a new block generation event is scheduled based on an exponential random variable with rate .
* If more than one customer is in the queue, the block generation time remains unchanged.
* If the block generator is busy or the channel is OFF, the next block generation time is set to infinity to suspend the process.

### Block Generation Subprogram

Figure 4‑3 illustrates the flow chart of the block generation subprogram, which simulates the initiation of a block generation process. When this event is triggered, the simulation time is updated to the scheduled block generation time. Then, the area calculation function is invoked to update all time-averaged statistics based on the elapsed time since the last event.

Next, the block generator status is set to busy, indicating that a block is currently being generated. To ensure sufficient space for the upcoming consensus process, the capacity limit of the customer queue is reduced from to , where is the maximum number of customers allowed in a block.

The system then determines how many customers should be transferred from the queue into the block:

* If more than customers are waiting in the queue, exactly are selected.
* Otherwise, all remaining customers in the queue are moved into the block.

The number of customers transferred into the block is recorded, and the queue size is adjusted accordingly. A block departure event is then scheduled based on an exponential random variable with rate . After this, the next block generation time is set to infinity to prevent immediate retriggering.

For each customer that enters the block, their corresponding arrival time is logged into the block log. These timestamps are subsequently used to compute the cumulative queueing time. This calculation is performed using the total waiting time function, which sums the time differences between the current simulation time and each customer's original queue entry time.

Finally, the corresponding entries in the queue log are removed to reflect that these customers have exited the queue and are now participating in the consensus process.

### Block Departure Subprogram

Figure 4‑4 illustrates the flow chart of the departure subprogram, which simulates the completion of a block consensus process. When this event is triggered, the simulation time is updated to the scheduled block departure time. Then, the area calculation function is invoked to update all time-averaged statistics based on the elapsed time since the last event.

At this point, the block generation status is reset to idle, and the customer queue capacity limit is restored to its original value , allowing the queue to accept new customers at full capacity. The block departure event is considered completed and is therefore cleared.

The program then calculates the total time that the current block of customers spent in the consensus stage. This is achieved using the block time accumulation function, which computes the total time difference between the current simulation time and each customer's recorded entry into the block.

After consensus completion, the number of customers currently in the system is decreased by the number of customers in the departing block, and the total number of customers served is incremented accordingly. The block is now empty, and all associated entries in the block log are removed.

Finally, if there are still customers waiting in the queue, a new block generation event is scheduled based on an exponential random variable with rate .

### Switch Subprogram

Figure 4‑5 illustrates the flow chart of the switch subprogram, which simulates the transition of the system between ON and OFF states. When this event is triggered, the simulation time is updated to the scheduled switch time. Then, the area calculation function is invoked to update all time-averaged statistics based on the elapsed time since the last event.

The system channel status is then toggled:

* If the current state is ON, the system transitions to the OFF state. In this case, a new switch event is scheduled based on an exponential random variable with rate (representing the OFF duration). During the OFF period, both block generation and block departure events are suspended by setting their scheduled times to infinity.
* If the current state is OFF, the system transitions to the ON state. A new switch event is scheduled based on an exponential random variable with rate α (representing the ON duration).

Upon reentering the ON state, the system checks whether conditions allow for service resumption:

* If there are customers in the queue and no block is currently being generated, a block generation event is scheduled based on an exponential random variable with rate . However, to ensure this event occurs before the next switch, the event is only retained if its scheduled time is earlier than the next switch; otherwise, it is canceled.
* If a block is currently in progress, a block departure event is also scheduled based on an exponential random variable with rate , again under the condition that the event completes before the next switch.

Through this subprogram, the simulation captures the stochastic availability of the system by alternating between operational and suspended phases, reflecting real-world unreliability such as downtime or external disruptions.



Figure 4‑1: Flow chart of main program

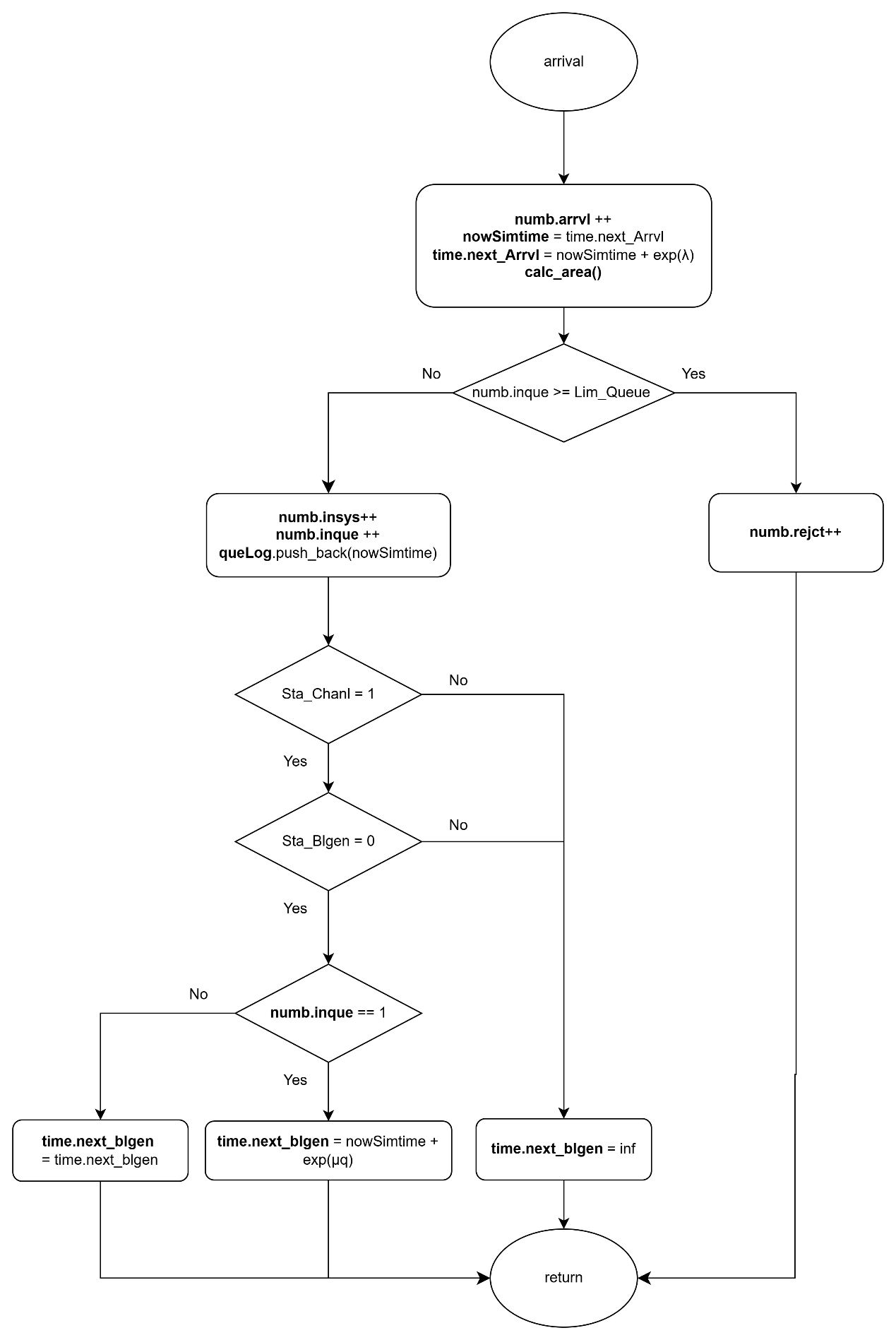


Figure 4‑2: Flow chart of arrival subprogram

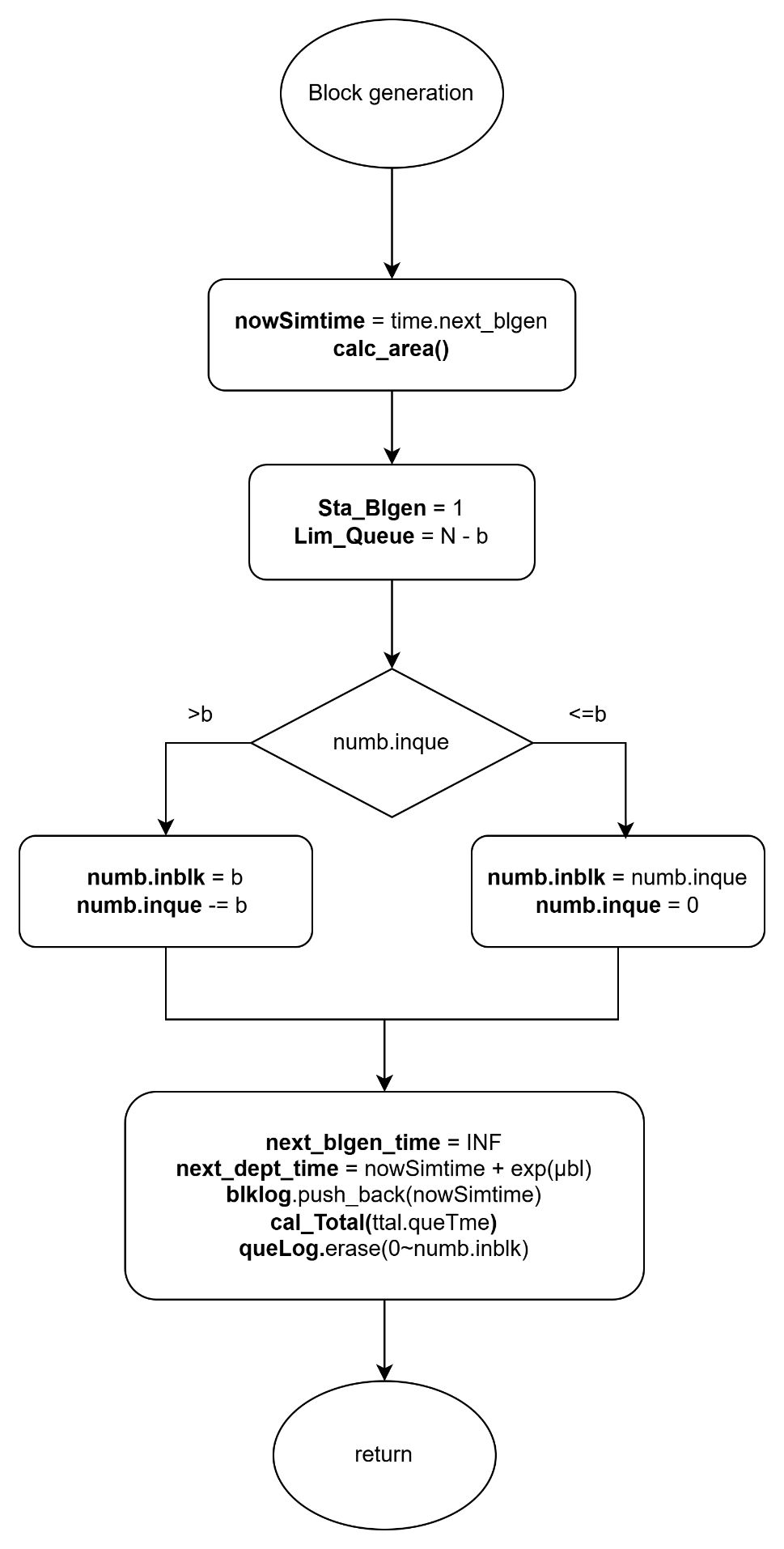


Figure 4‑3:Flow chart of block generation subprogram

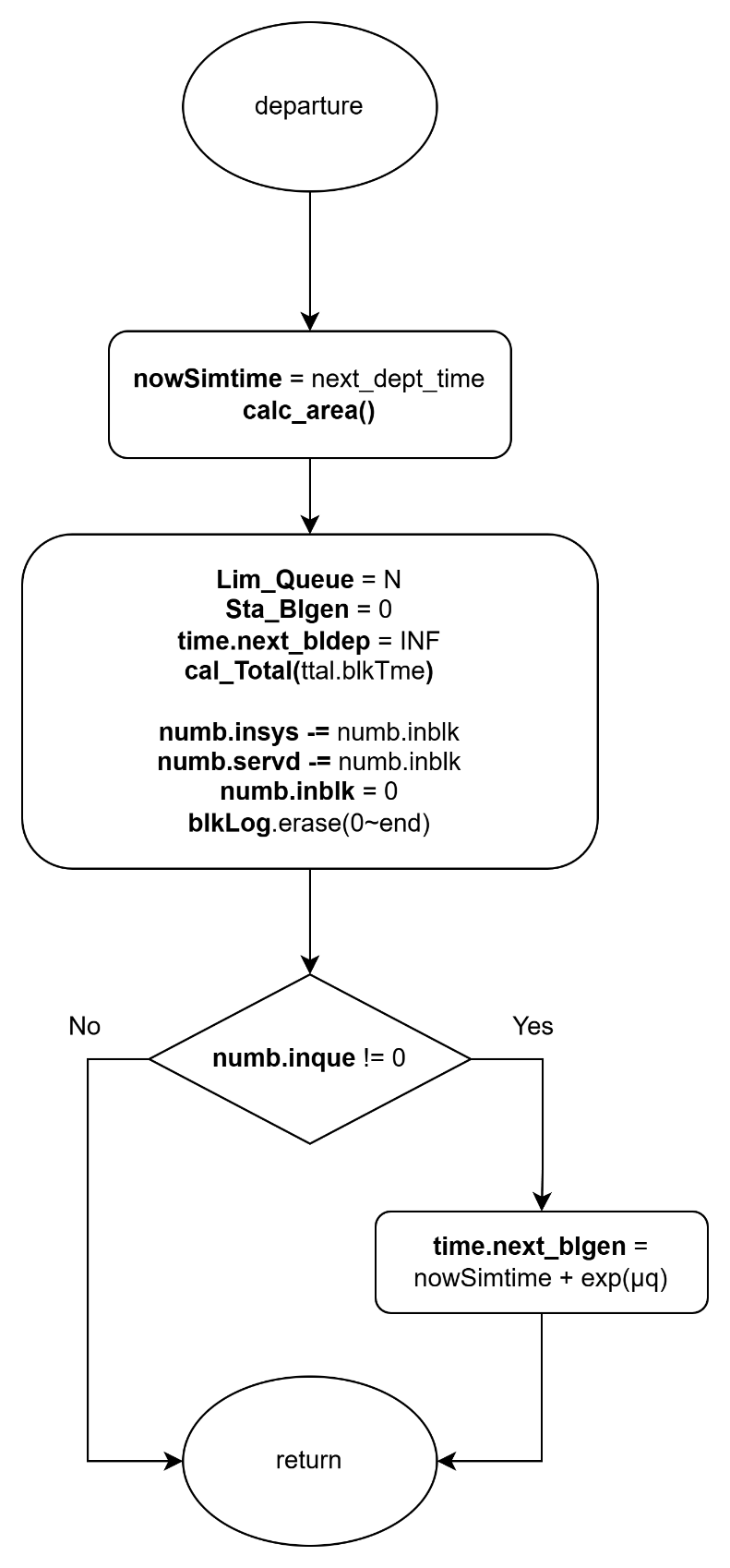


Figure 4‑4: Flow chart of block departure subprogram

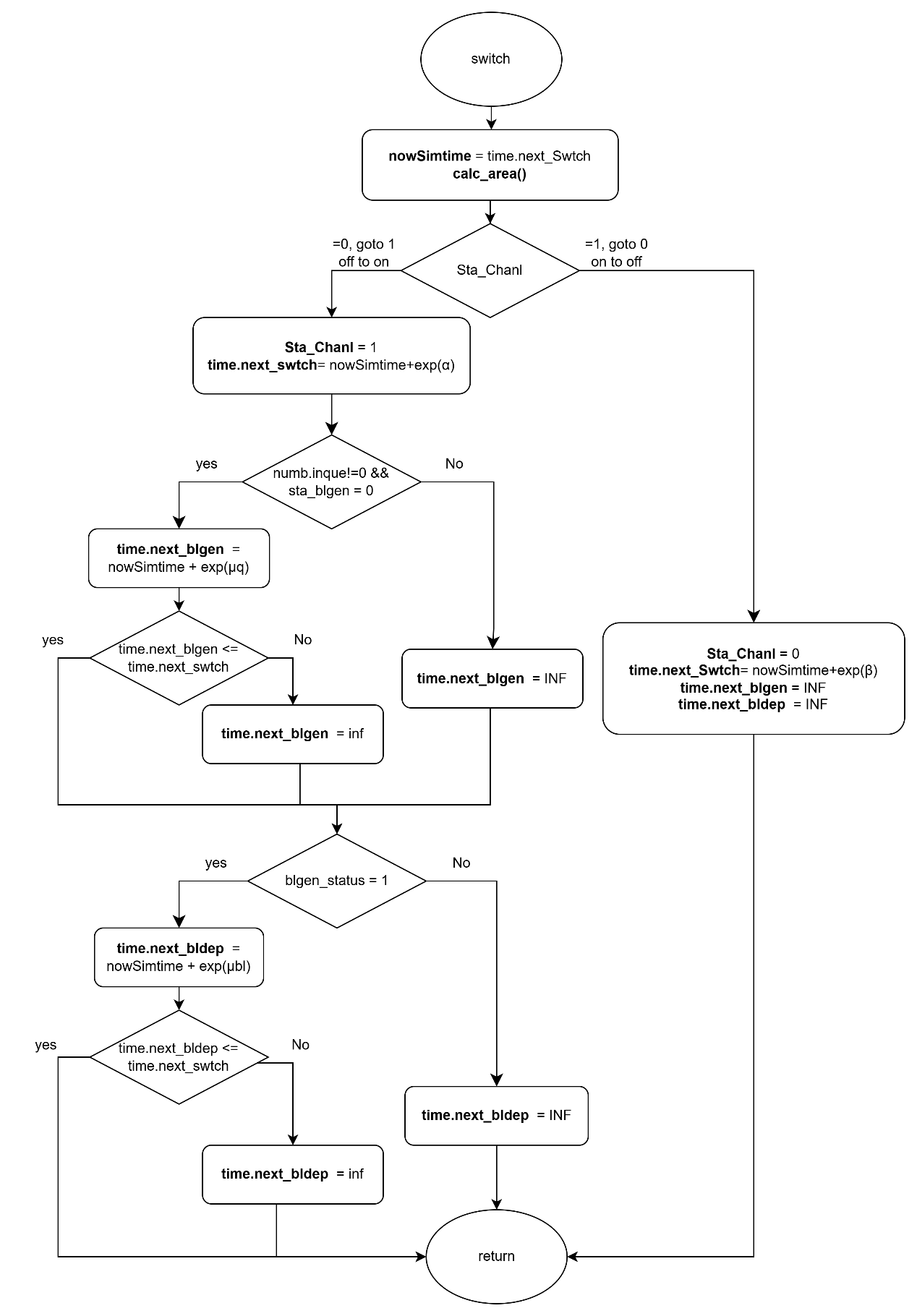


Figure 4‑5: Flow chart of switch subprogram

### Performance Index

To evaluate the system’s performance, we compute several performance indices based on the simulated results obtained from the simulation.

First of all, the average number of customers in the whole system, denoted by , is given by:

Second, the average number of customers in customer queue, denoted by , is given by:

Third, the average number of customers in consensus queue, denoted by , is given by:

Fourth, the blocking probability of the system, denoted by , is given by:

Fifth, the throughput of the system, denoted by , is given by:

Sixth, the average waiting time in the system, denoted by , is given by:

Seventh, the average waiting time in the customer queue, denoted by , is given by:

Eighth, the average waiting time in the consensus queue, denoted by , is given by:

Finally, the average number of blocks participating in the consensus process, denoted by , is given below.

## Scenario 2: Two-Class Customer without Impatience

In this simulation model, we consider a blockchain system that handles a two class of users: high-priority and low-priority customers. Customers arrive according to independent Poisson process, and users of the same priority are served under the First-Come-First-Served (FCFS) discipline. Customers of different priorities are handled using a non-preemptive priority rule, where high-priority customers are always placed ahead of low-priority customers in the queue, but ongoing service for a low-priority block cannot be interrupted once initiated.

The system consists of two queues: the customer queue and the consensus queue. High-priority and low-priority customers both enter the customer queue upon arrival. Block generation follows a partial batch service policy and operates on customers of only one priority class at a time. Each block can include at most users. Once a block that composed solely of high-priority or low-priority is formed, it is transferred to the consensus queue. After completing the consensus process, all users in the block exit the system.

The system alternates between ON and OFF periods to reflect external interruptions such as cyberattacks or connection failures. During the OFF state, both block generation and consensus process are suspended, though new customers may still arrive and be queued. During the ON state, all service resume as normal.

To preserve system integrity and fairness, different queue capacity constraints apply based on user class and system state. For high-priority customers, the maximum number allowed in the customer queue is when the consensus queue is empty, and when it is occupied. For low-priority customers, the customer queue capacity is always limited to , regardless of the consensus queue state.

Since this model dose not include customer impatience, all arriving users remain in the system until they are served. This scenario is designed to study the impact of non-preemptive priority scheduling on performance, focusing on metrics such as per-class throughput, queue length, and system utilization under stable yet priority-sensitive operation.

### Main program

The main program executes a series of steps to simulate the blockchain queuing system with two classes of customers, as illustrated in \_\_\_\_. At the beginning of each simulation run, all relevant variables are initialized. This includes resetting statistical parameters, setting the next block generation time and next departure time to infinity, marking the system status as ON, initializing the block generation status as idle, and setting the customer queue limit to .

Next, the system parameters are configured. These include the maximum customer queue capacity (), the maximum number of users per block (), the arrival rates for high-priority and low-priority customers ( and ), the block generation rates ( and ), the consensus (block departure) rate ( and ), and the ON/OFF switching rates ( and ) for the system channel.

The program then generates the next arrival time for both high-priority and low-priority customers, as well as the channel switch time, using exponential random variables based on the corresponding system parameters. During the simulation, it compares the scheduled times of five events and selects the earliest event to execute its corresponding subprogram.

Finally, a while loop is used to repeat the simulation until a predefined number of customer arrivals has been reached. Once this condition is met, the simulation terminates and the performance statistics are output.

### High-Priority Arrival Subprogram

As illustrated in \_\_\_\_, the high-priority arrival subprogram simulates the arrival of a high-priority customer to the system. When this event is triggered, the simulation time is updated to the current arrival time. The arrival counters are incremented to reflect both the total number of customers and the number of high-priority arrivals. The next arrival event for high-priority customers is then scheduled based on an exponential random variable with rate . Immediately after, the area calculation function is invoked to update all time-averaged statistics based on the elapsed time since the last event.

Next, the system checks whether the customer queue has reached its capacity limit.

* If the queue is full, the arriving customer is rejected. In this case, both the total number of rejections and the number of high-priority rejections is incremented.
* If the queue is not full, the arriving customer is admitted. The number of customers in the system and in the queue are both incremented, along with their corresponding high-priority counts. The arrival time of the customer is recorded in the high-priority queue log.

Then, the system updates the unified queue log by merging the high-priority and low-priority arrival records, ensuring that high-priority entries appear first. The system then sets the priority flag to indicate that high-priority customers are currently at the head of the queue.

Finally, the system determines whether to initiate block generation:

* If the channel status is in ON state, and block generator is idle, and this customer is the only one in the queue, a new block generation event is scheduled based on an exponential random variable with rate .
* If more than one high-priority customer is in the queue, the block generation time remains unchanged.
* If the block generator is busy or the channel is OFF, the next block generation time is set to infinity to suspend the process.

### Low-Priority Arrival Subprogram

As illustrated in \_\_\_\_, the low-priority subprogram simulates the arrival of a low-priority customer to the system. When this event is triggered, the simulation time is updated to the scheduled arrival time. The arrival counters are incremented to reflect both the total number of customers and the number of low-priority arrivals.

The next arrival event for low-priority customers is then scheduled based on an exponential random value with the rate . Then, the area calculation function is invoked to update all time-averaged statistics based on the elapsed time since the last event.

Next, the system checks whether the customer queue has reached its capacity limit. The admission condition depends on the system’s block generation state:

* If no block is being generated and either (1) the number of low-priority customers in the queue has reached , or (2) the total number of customers has reached , or
* If a block is currently being generated and the total number of customers in the queue has reached ,

Then the arriving customer is rejected. In this case, both the total number of rejections and the number of low-priority rejections are incremented.

If the customer queue is not full, the arriving customer is admitted. The number of customers in the system and in the queue are both incremented, along with their corresponding low-priority counts. The arrival time is recorded in the low-priority queue log. To maintain unified tracking, the system then refreshes the combined queue log by merging both priority records, ensuring high-priority entries appear first.

If there are no high-priority customers currently in the queue, the system sets the priority flag to indicate that low-priority customers are now at the head of the queue.

Finally, the system determines whether to initiate block generation:

* If the channel status is in ON state, and block generator is idle, and this customer is the only one in the queue, a new block generation event is scheduled based on an exponential random variable with rate .
* If more than one customer is in the queue, the block generation time remains unchanged.
* If the block generator is busy or the channel is OFF, the next block generation time is set to infinity to suspend the process.

### Block Generation Subprogram

As illustrated in \_\_\_\_, the block generation subprogram simulates the initiation of a block generation process. When this event is triggered, the simulation time is updated to the scheduled block generation time. Then, the area calculation function is invoked to update all time-averaged statistics based on the elapsed time since the last event.

The system marks the block generator status as active and reduces the queue capacity limited from to to ensure that new incoming customers do not overflow the system while the consensus queue is occupied.

The system then determines which priority of customers will be selected for the current block:

* If there are high-priority customers in the queue, they are always selected first.
  + If the number of high-priority customers exceeds the block size , exactly of them are moved into the block.
  + If there are fewer than high-priority customers, all of them are moved into the block. The system then checks whether any low-priority customers remain in the queue to update the queue head status accordingly.
  + The block is marked as high-priority, and the consensus process is scheduled based on an e
* If more than customers are waiting in the queue, exactly are selected.
* Otherwise, all remaining customers in the queue are moved into the block.

The number of customers transferred into the block is recorded, and the queue size is adjusted accordingly. A block departure event is then scheduled based on an exponential random variable with rate . After this, the next block generation time is set to infinity to prevent immediate retriggering.

For each customer that enters the block, their corresponding arrival time is logged into the block log. These timestamps are subsequently used to compute the cumulative queueing time. This calculation is performed using the total waiting time function, which sums the time differences between the current simulation time and each customer's original queue entry time.

Finally, the corresponding entries in the queue log are removed to reflect that these customers have exited the queue and are now participating in the consensus process.

### Block Departure Subprogram

Figure 4‑4 illustrates the flow chart of the departure subprogram, which simulates the completion of a block consensus process. When this event is triggered, the simulation time is updated to the scheduled block departure time. Then, the area calculation function is invoked to update all time-averaged statistics based on the elapsed time since the last event.

At this point, the block generation status is reset to idle, and the customer queue capacity limit is restored to its original value , allowing the queue to accept new customers at full capacity. The block departure event is considered completed and is therefore cleared.

The program then calculates the total time that the current block of customers spent in the consensus stage. This is achieved using the block time accumulation function, which computes the total time difference between the current simulation time and each customer's recorded entry into the block.

After consensus completion, the number of customers currently in the system is decreased by the number of customers in the departing block, and the total number of customers served is incremented accordingly. The block is now empty, and all associated entries in the block log are removed.

Finally, if there are still customers waiting in the queue, a new block generation event is scheduled based on an exponential random variable with rate .

### Switch Subprogram

Figure 4‑5 illustrates the flow chart of the switch subprogram, which simulates the transition of the system between ON and OFF states. When this event is triggered, the simulation time is updated to the scheduled switch time. Then, the area calculation function is invoked to update all time-averaged statistics based on the elapsed time since the last event.

The system channel status is then toggled:

* If the current state is ON, the system transitions to the OFF state. In this case, a new switch event is scheduled based on an exponential random variable with rate (representing the OFF duration). During the OFF period, both block generation and block departure events are suspended by setting their scheduled times to infinity.
* If the current state is OFF, the system transitions to the ON state. A new switch event is scheduled based on an exponential random variable with rate α (representing the ON duration).

Upon reentering the ON state, the system checks whether conditions allow for service resumption:

* If there are customers in the queue and no block is currently being generated, a block generation event is scheduled based on an exponential random variable with rate . However, to ensure this event occurs before the next switch, the event is only retained if its scheduled time is earlier than the next switch; otherwise, it is canceled.
* If a block is currently in progress, a block departure event is also scheduled based on an exponential random variable with rate , again under the condition that the event completes before the next switch.

Through this subprogram, the simulation captures the stochastic availability of the system by alternating between operational and suspended phases, reflecting real-world unreliability such as downtime or external disruptions.



Figure 4‑1: Flow chart of main program

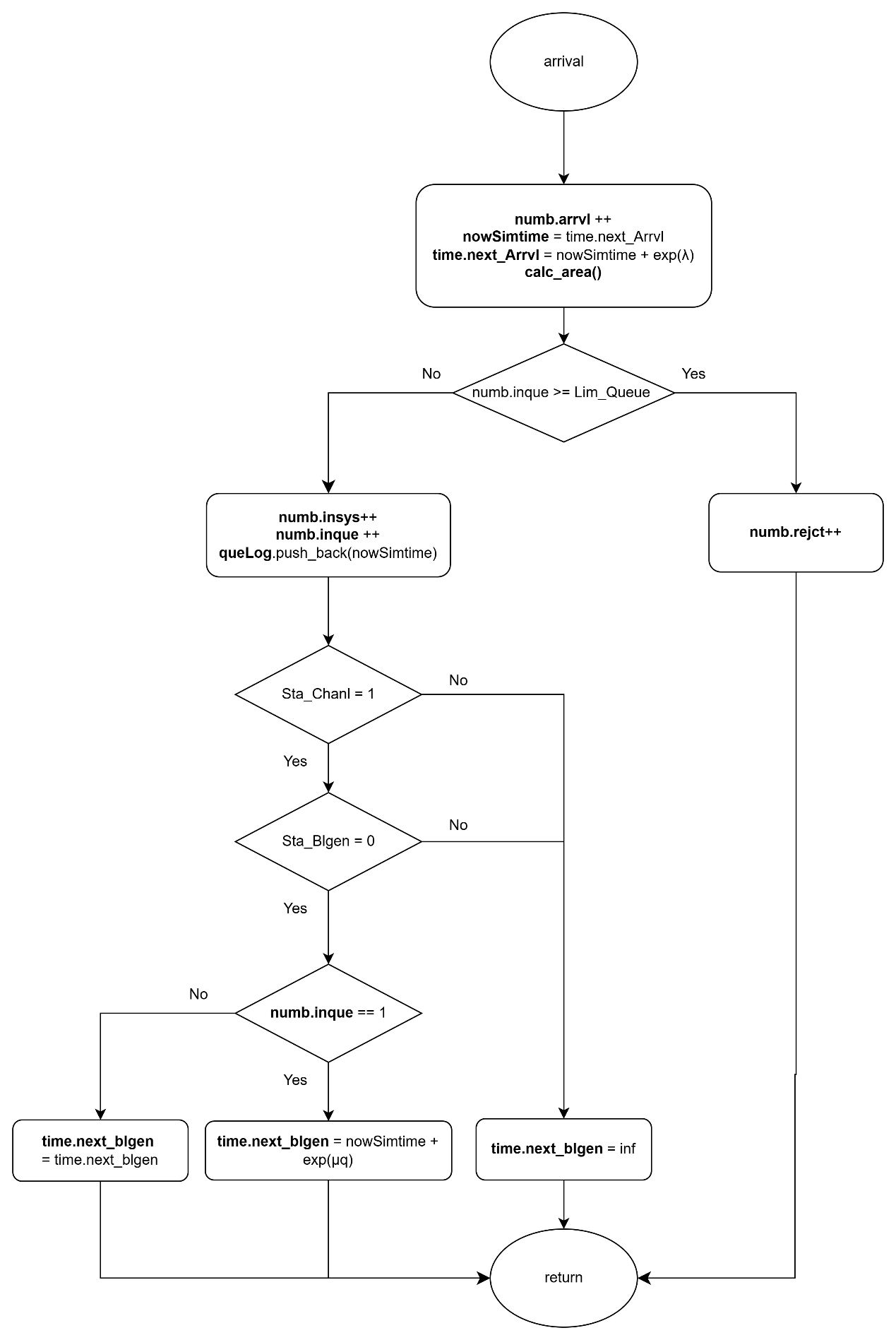


Figure 4‑2: Flow chart of arrival subprogram

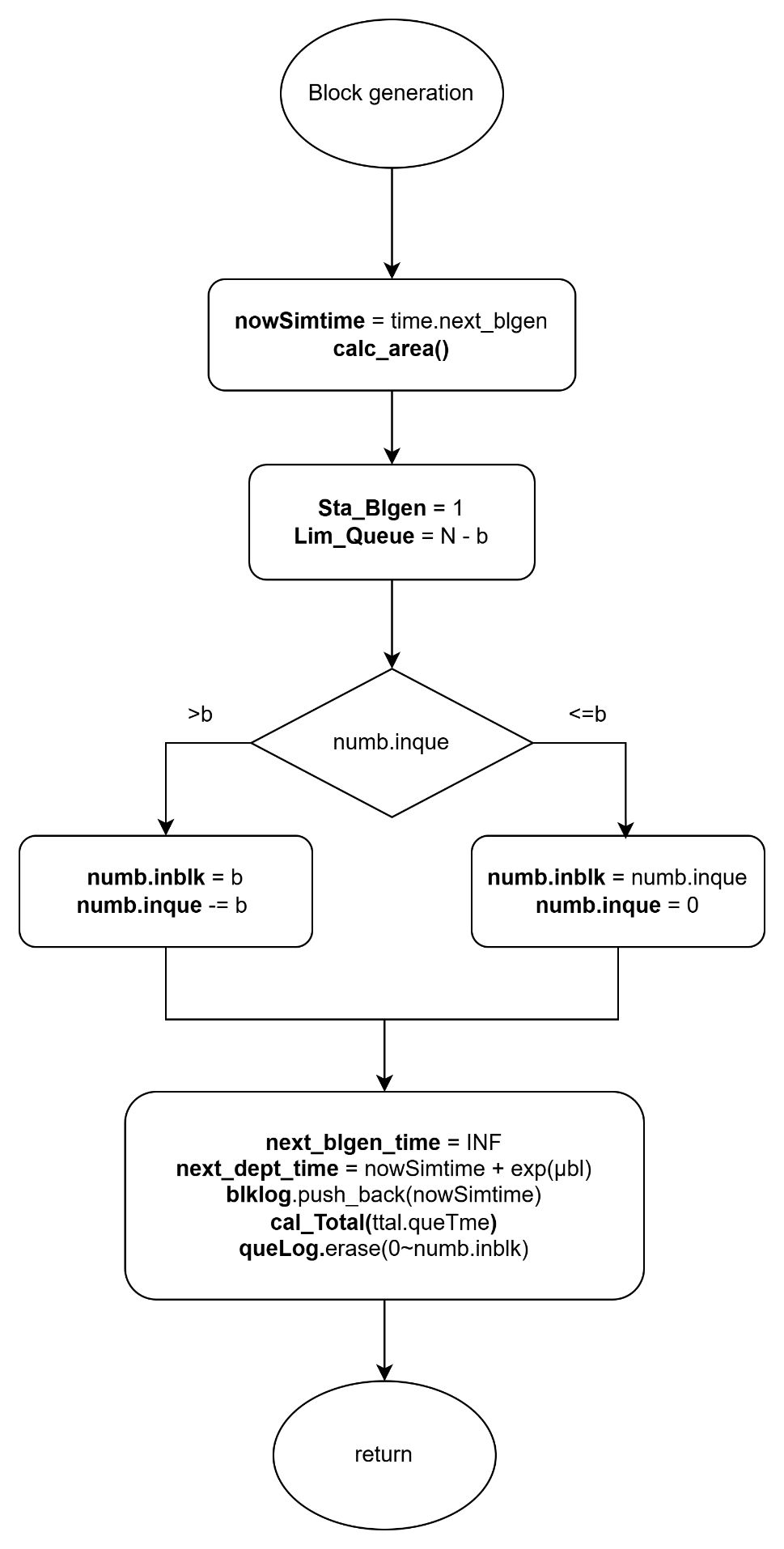


Figure 4‑3:Flow chart of block generation subprogram

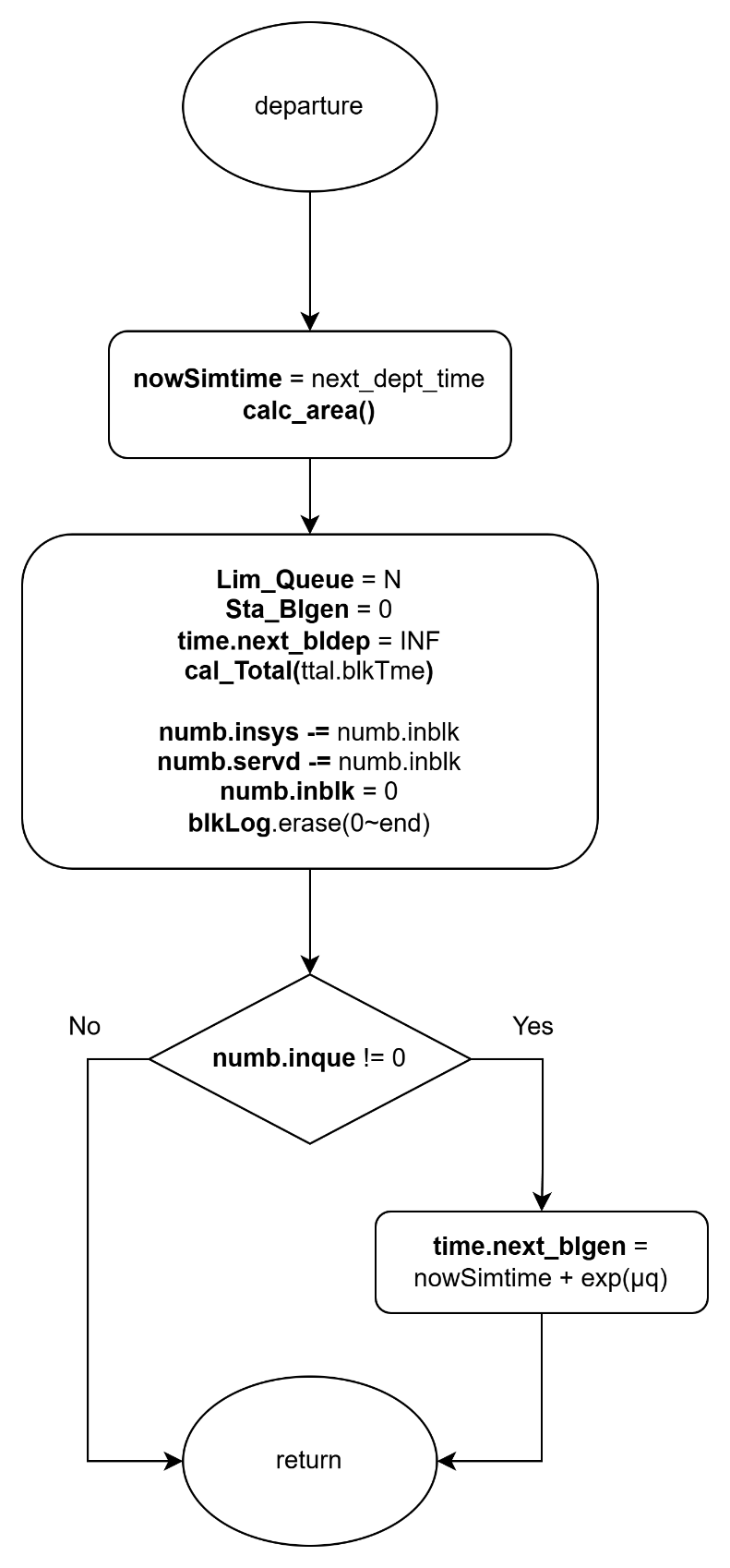


Figure 4‑4: Flow chart of block departure subprogram

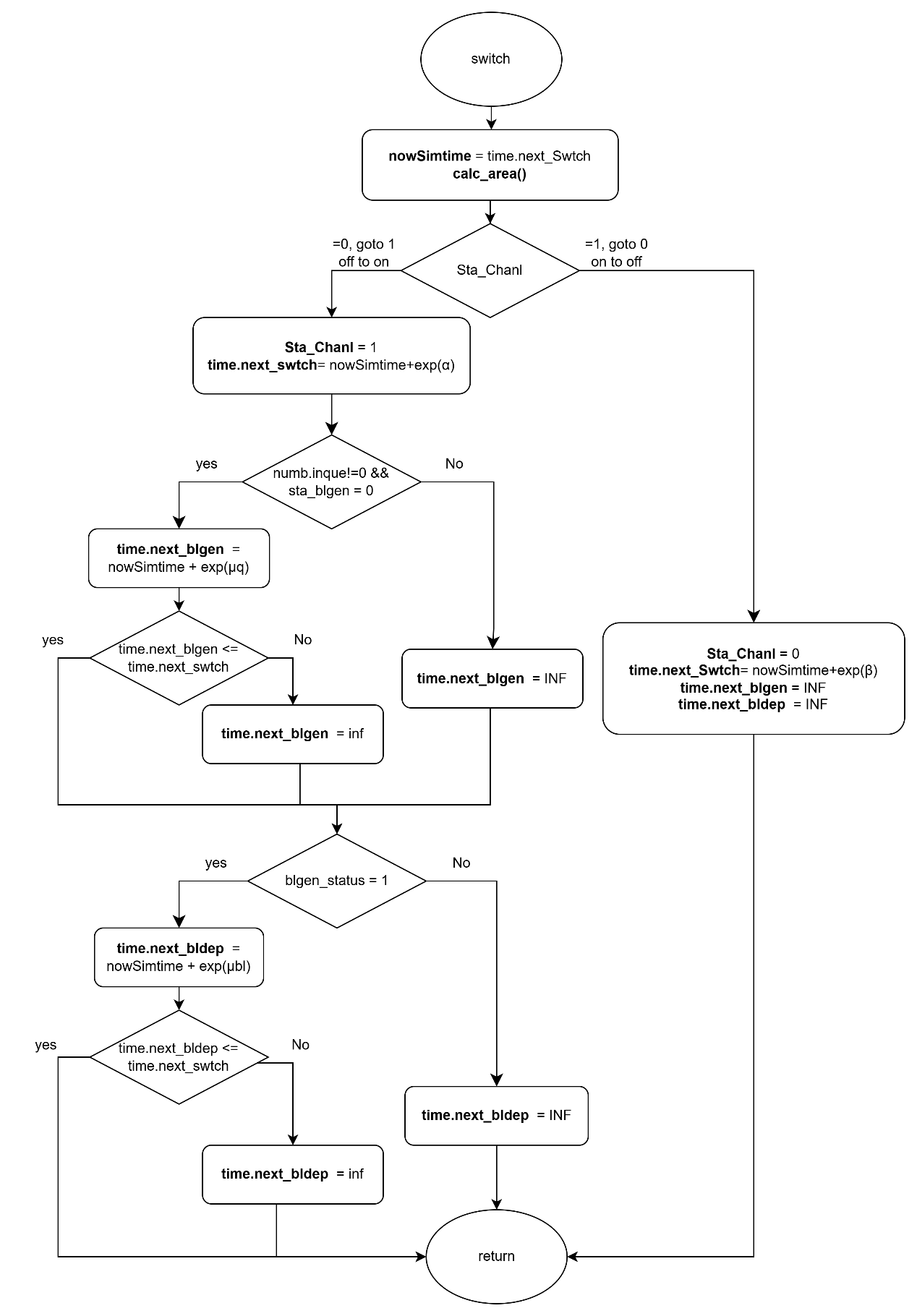


Figure 4‑5: Flow chart of switch subprogram

### Performance Index

To evaluate the system’s performance, we compute several performance indices based on the simulated results obtained from the simulation.

First of all, the average number of customers in the whole system, denoted by , is given by:

Second, the average number of customers in customer queue, denoted by , is given by:

Third, the average number of customers in consensus queue, denoted by , is given by:

Fourth, the blocking probability of the system, denoted by , is given by:

Fifth, the throughput of the system, denoted by , is given by:

Sixth, the average waiting time in the system, denoted by , is given by:

Seventh, the average waiting time in the customer queue, denoted by , is given by:

Eighth, the average waiting time in the consensus queue, denoted by , is given by:

Finally, the average number of blocks participating in the consensus process, denoted by , is given below.

## Scenario 3: Single-Class Customer with Impatience

## Scenario 4: Two-Class Customer with Impatience