# 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 as follows:

* **If the system transitions from ON to OFF:**
  + The channel status is set to OFF.
  + The next switch event is scheduled based on an exponential random variable with rate (representing the OFF duration).
  + All ongoing service operations are suspended by setting both the block generation and block departure event times to infinity.
* **If the system transitions from OFF to ON:**
  + The channel status is set to ON.
  + The next switch event is scheduled based on an exponential random variable with rate (representing the ON duration).
  + If there are customers in the queue and the block generator is currently idle:
    - A block generation event is scheduled based on an exponential random variable with rate .
  + If a block is currently in progress:
    - A block departure event is scheduled using an exponential random variable with rate .

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. During the ON period, block generation and consensus operations proceed as normal. During the OFF period, these processes are temporarily halted while new customer arrivals may still occur.



Figure ‑: Flow chart of main program

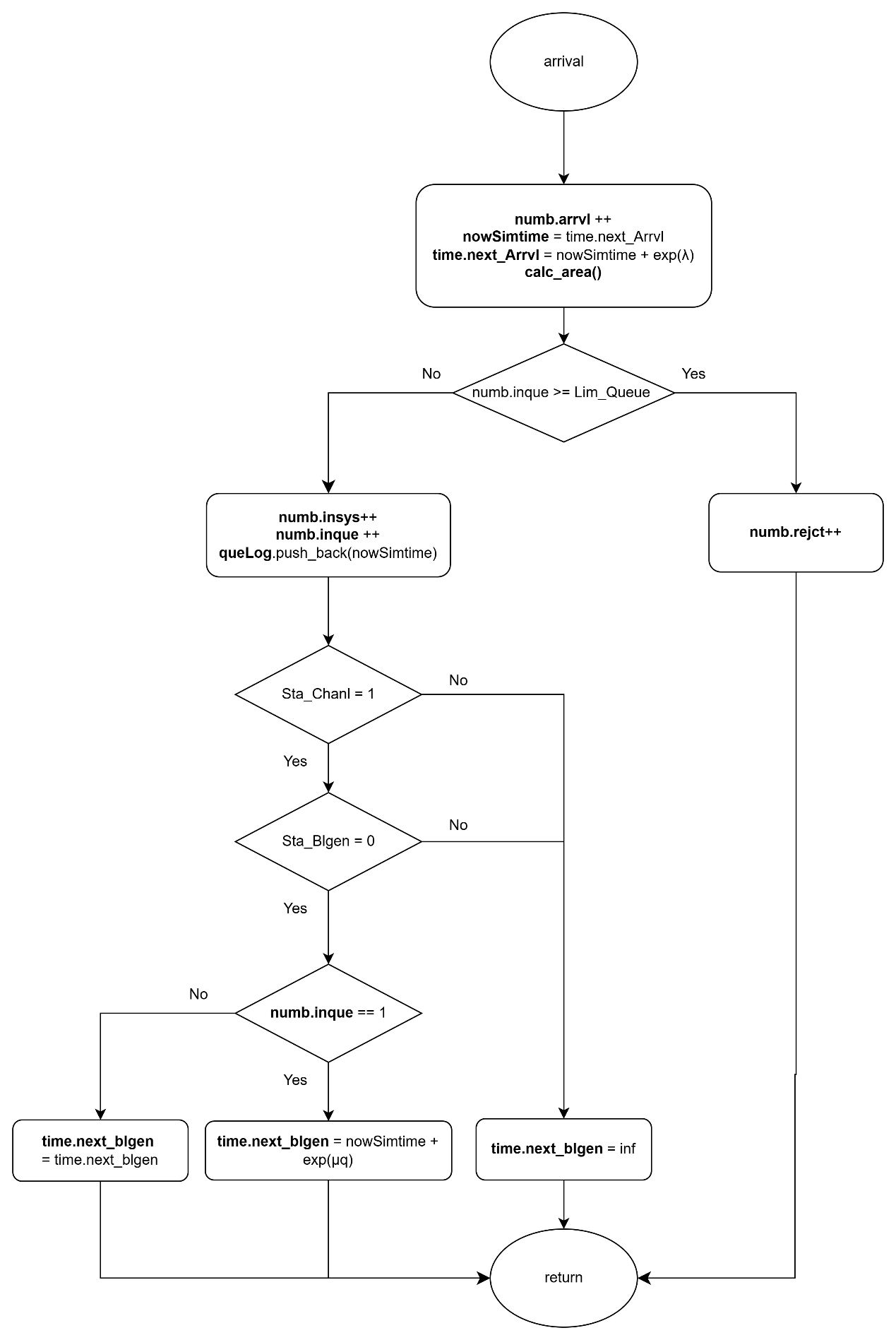


Figure ‑: Flow chart of arrival subprogram

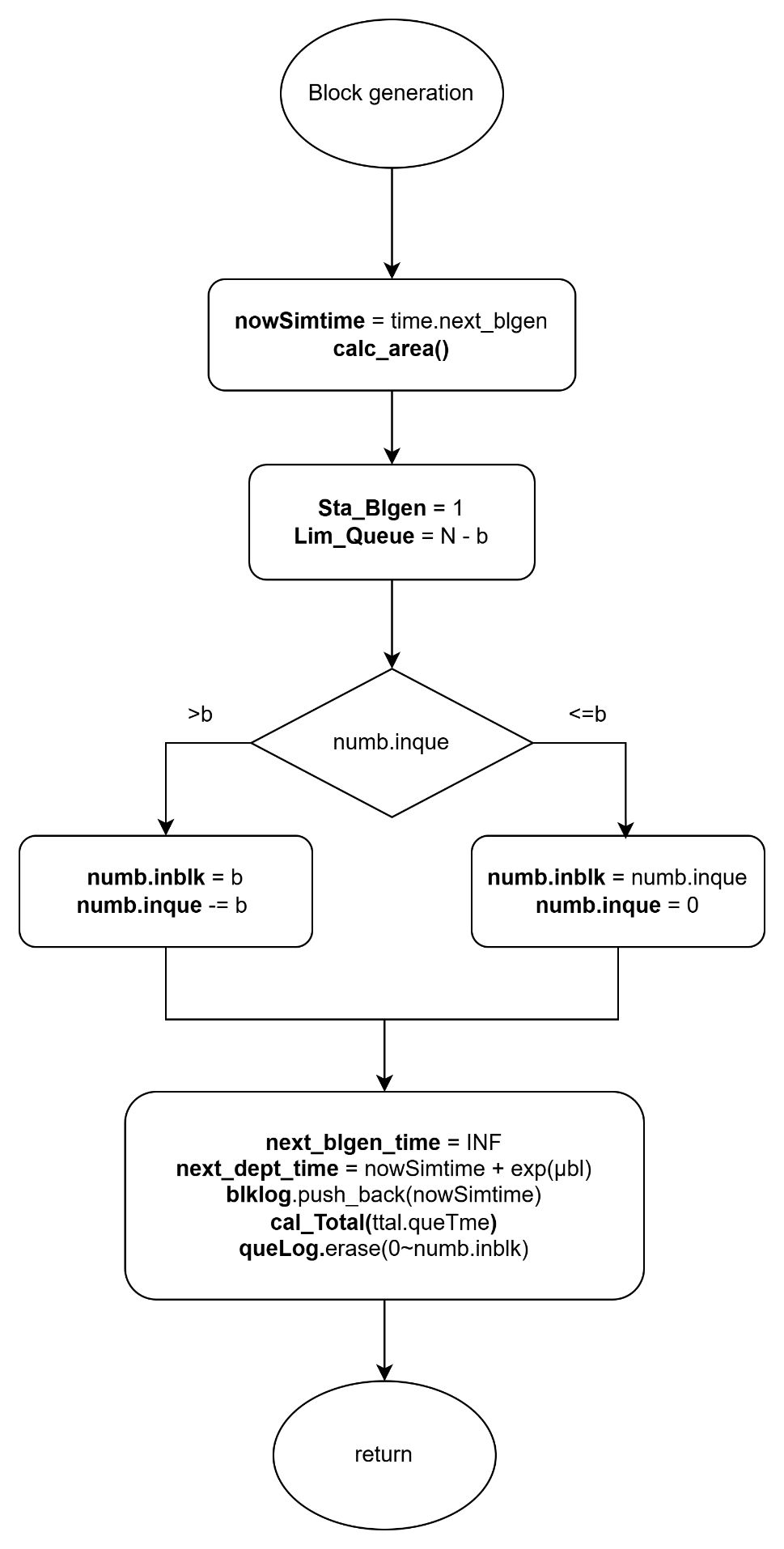


Figure ‑:Flow chart of block generation subprogram

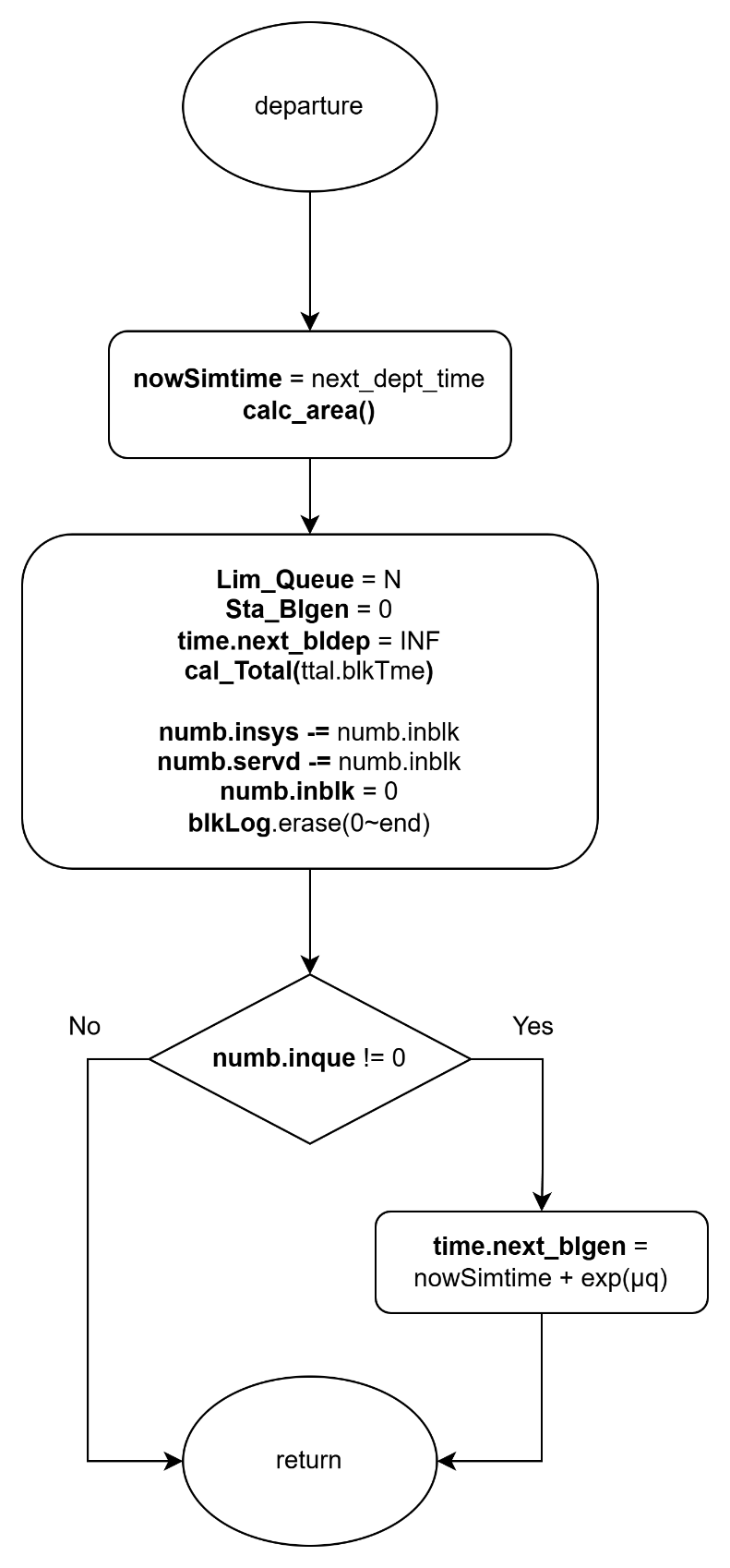


Figure ‑: Flow chart of block departure subprogram



Figure ‑: 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 per unit time, 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 does 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 Figure 4‑6. 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 Figure 4‑7, 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 Figure 4‑8, 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 the number of low-priority customers in the queue has reached , or
* The total number of customers in the queue has reached the current queue limit,

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 Figure 4‑9, 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 next block generation time is set to infinity to prevent immediate retriggering, and the block generator status is marked as active.

To reserve sufficient space in the queue for block formation, the maximum queue size is reduced from to .

The system then determines the class of customers from which the block will be constructed, based on the current queue head status:

* **If high-priority customers are at the head of the queue**:
  + If the number of high-priority customers exceeds the block size , exactly of them are moved into the block.
  + Otherwise, all high-priority customers are included in the block. If low-priority customers remain in the queue, the queue head status is updated accordingly; otherwise, it is cleared.
  + The block is marked as high-priority, and the consensus process is scheduled based on an exponential random variable with rate .
  + The total waiting time of high-priority customers in the block is calculated using their individual arrival times stored in the high-priority queue log, which is then updated by removing the corresponding entries.
* **If high-priority customers are at the head of the queue**:
  + If there are more than low-priority customers, exactly are selected for the block.
  + Otherwise, all remaining low-priority customers are included, and the queue is emptied.
  + The block is marked as low-priority, and the block departure event is scheduled based on an exponential random variable with rate .
  + The total waiting time of low-priority customers is calculated using their queue log, which is then updated accordingly.

After determining the block content, the system updates the overall waiting time in the queue for all customers, using the unified queue log. The combined queue log is then refreshed to reflect the current state of the customer queue.

Finally, the block log is updated with the entry time for each customer in the new block. This log is used for downstream statistics related to block-based consensus activity.

### Block Departure Subprogram

As illustrated in Figure 4‑10, the block departure subprogram simulates the completion of a block’s consensus process. When this event is triggered, the simulation time is updated to the current departure time. The area calculation function is then invoked to update all time-averaged statistics based on the elapsed time since the last event. The next block departure time is set to infinity to suspend further departure scheduling until a new block is formed.

The block generator status is reset to idle, and the customer queue capacity limit is restored to its original value , allowing the system to admit new arrivals without restriction.

The total waiting time of the block in the consensus stage is accumulated using the block log. The total number of customers currently in the system is decreased by the number of customers in the departing block, and the number of customers served is incremented accordingly. The block is then cleared.

Depending on the priority class of the departing block:

* If it is a high-priority block, the corresponding class-specific consensus time is updated, and the number of high-priority customers in the system and served counters are adjusted.
* If it is a low-priority block, the low-priority statistics are updated in a similar manner.

After processing, the system clears the block log and resets the priority status of the block.

Finally, the system checks whether customers remain in the queue. If so, a new block generation event is scheduled based on the class at the head of the queue:

* If high-priority customers are at the front, the next block generation is scheduled based on an exponential random variable with rate .
* If low-priority customers are at the front, the event is scheduled with rate .

If the queue is empty, no block generation is scheduled, and the next block departure time is set to infinity.

### Switch Subprogram

As illustrated in Figure 4‑11, the switch subprogram simulates the transition of the system between ON and OFF operational states. When this event is triggered, the simulation time is updated to the current switch time. The area calculation function is then invoked to update all time-averaged statistics based on the elapsed time since the last event.

The system channel status is then toggled as follows:

* **If the system transitions from ON to OFF:**
  + The channel status is updated to OFF.
  + The next switch event is scheduled based on an exponential random variable with rate β (representing the OFF duration).
  + All pending block generation and block departure events are suspended by setting their scheduled times to infinity.
* **If the system transitions from OFF to ON:**
  + The channel status is updated to ON.
  + The next switch event is scheduled based on an exponential random variable with rate α (representing the ON duration).
  + If there are customers in the queue and the block generator is idle:
    - A block generation event is scheduled based on an exponential random variable with rate or , depending on whether high- or low-priority customers are at the head of the queue.
  + If a block is currently in progress:
    - A block departure event is scheduled using the appropriate rate ( or ) based on the class of the current block.

Through this subprogram, the system emulates the effects of environmental disruptions such as connectivity loss or cyberattacks by alternating between active and inactive service periods. During the ON state, both queuing and consensus operations proceed. During the OFF state, only customer arrivals are allowed, while service is temporarily halted.



Figure ‑: Flow chart of main program

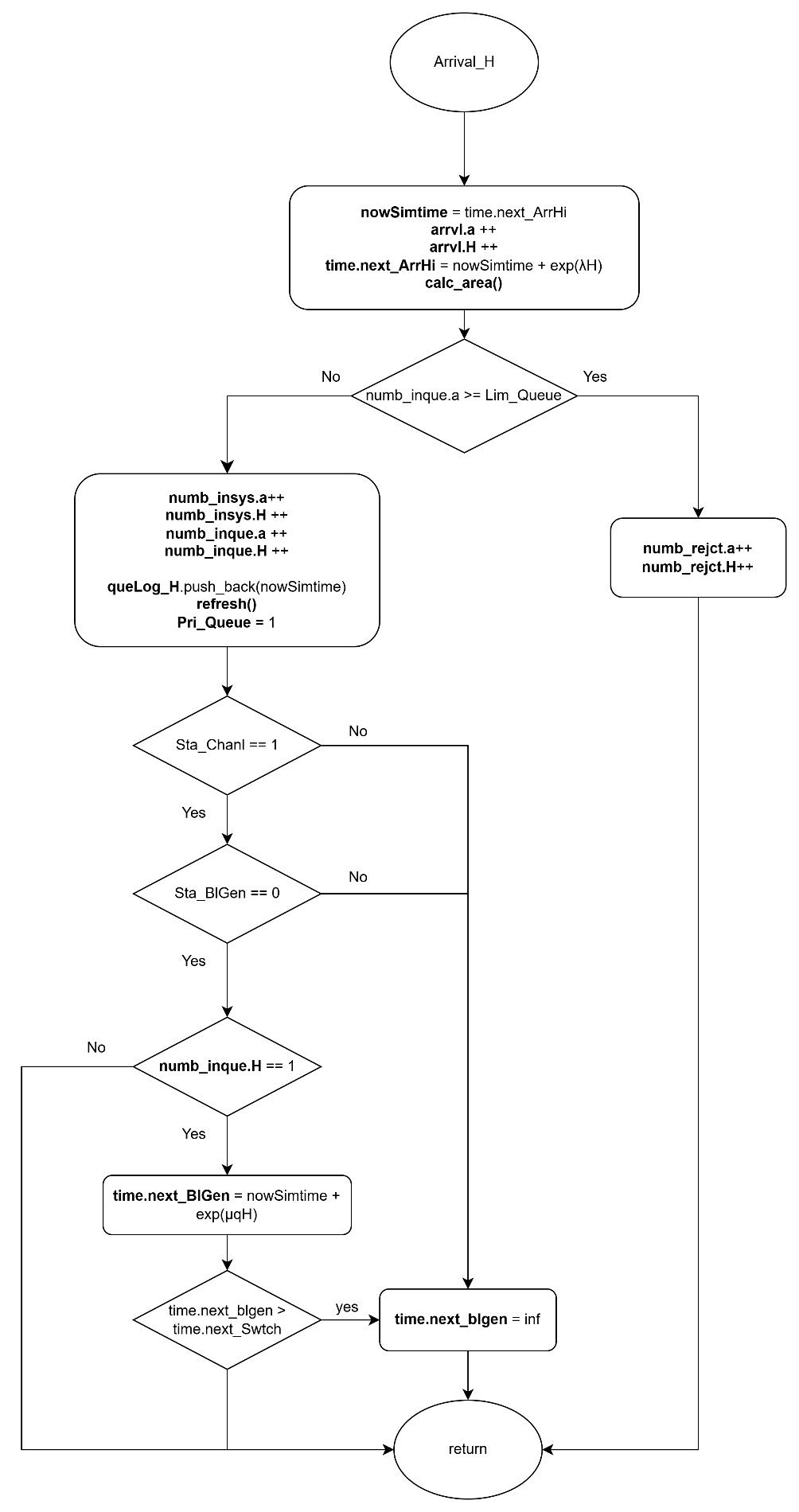


Figure ‑: Flow chart of high-priority arrival subprogram

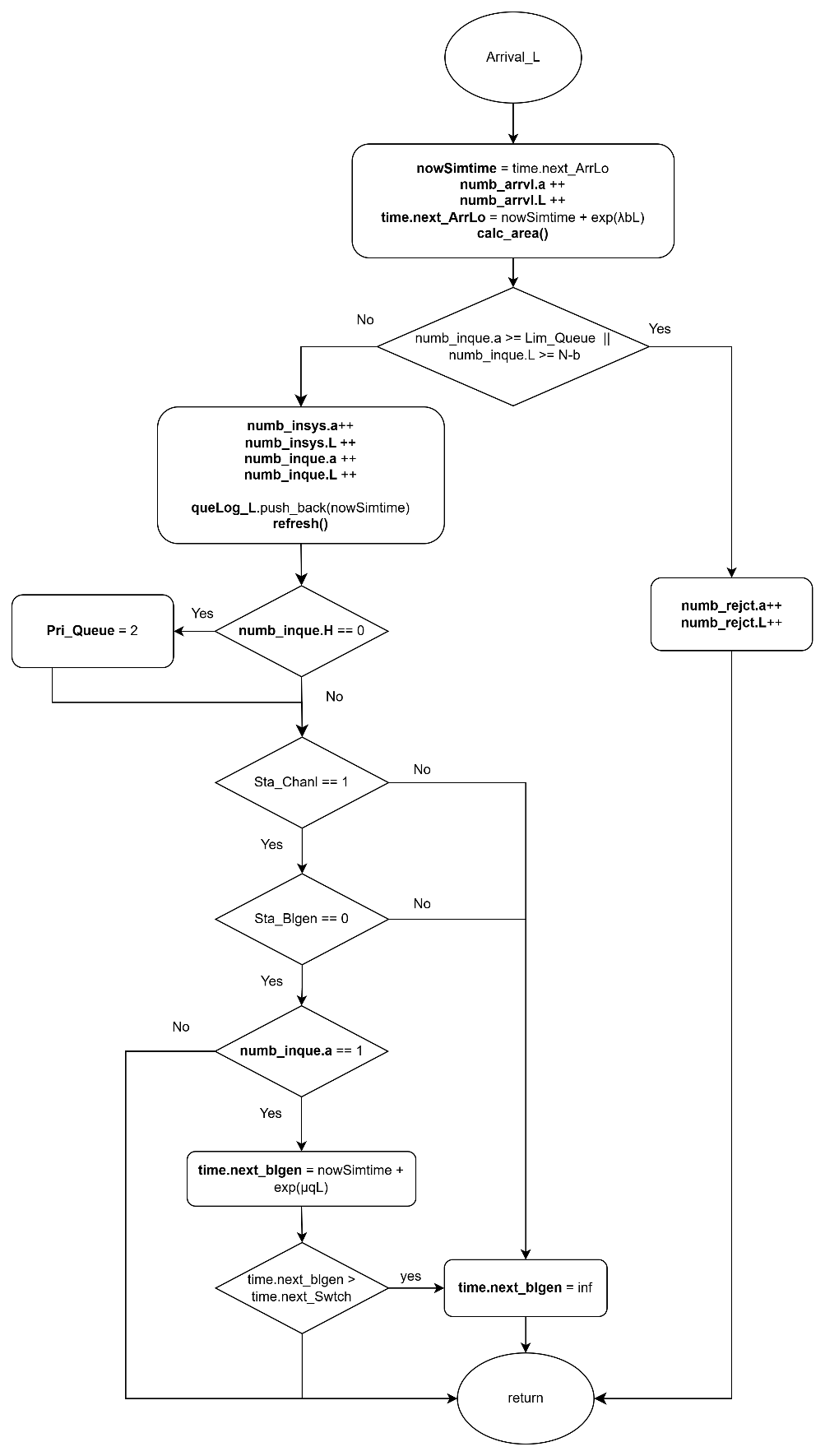


Figure ‑: Flow chart of low-priority arrival subprogram



Figure ‑:Flow chart of block generation subprogram



Figure ‑: Flow chart of block departure subprogram



Figure ‑: Flow chart of switch subprogram

### Performance Index

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 per unit time, denoted by and , is given by:

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

## Scenario 3: Single-Class Customer with Impatience

## Scenario 4: Two-Class Customer with Impatience