# 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 in the queue, but service-in-progress 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

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 goal of this scenario is to evaluate the system’s performance under ideal stability.

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. 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 participation.

### 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 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 customer queue is full, the customer is rejected, and the number of rejections is incremented.
* If customer 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 4‑1Flow chart of main program

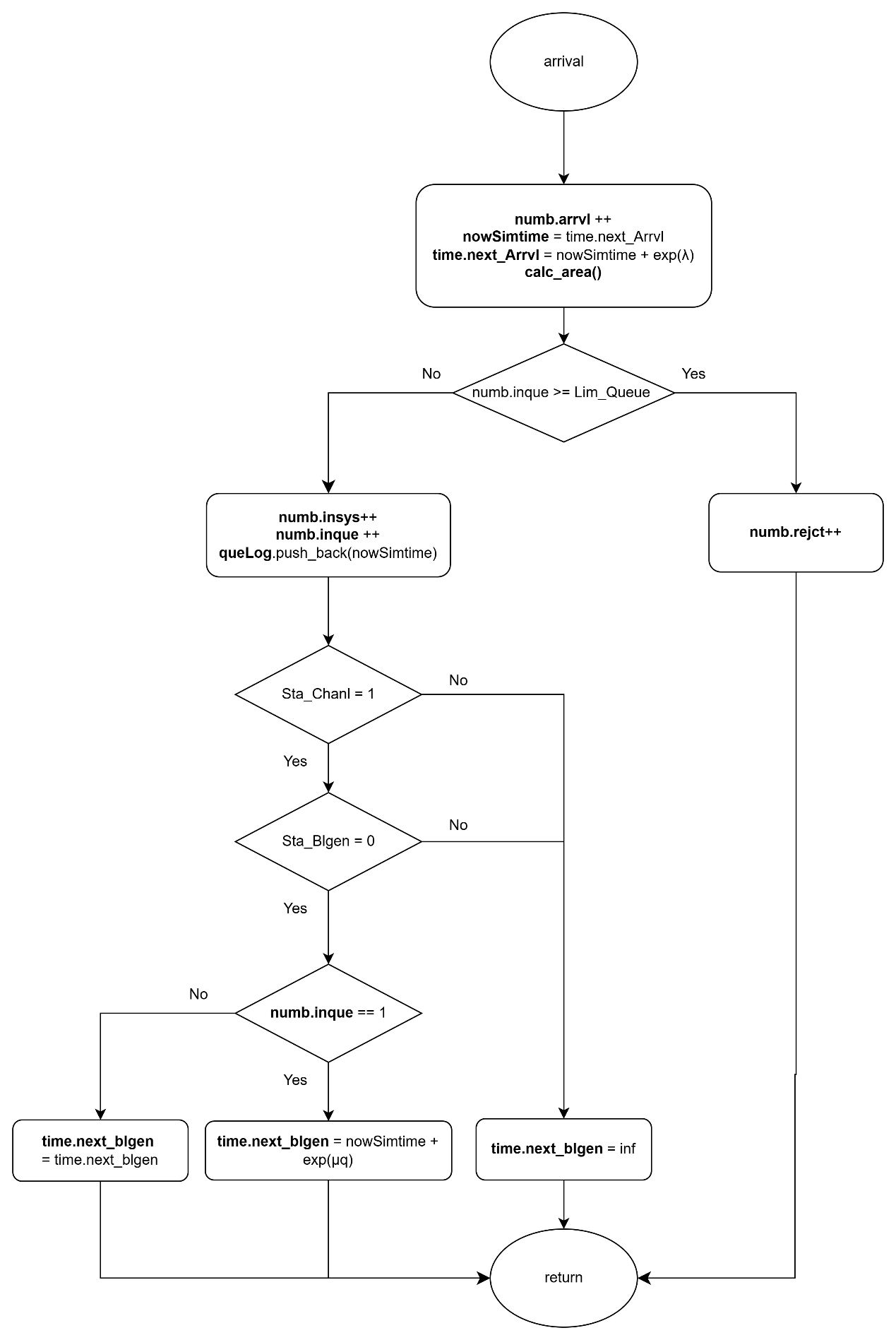


Figure 4‑2 Flow chart of arrival subprogram

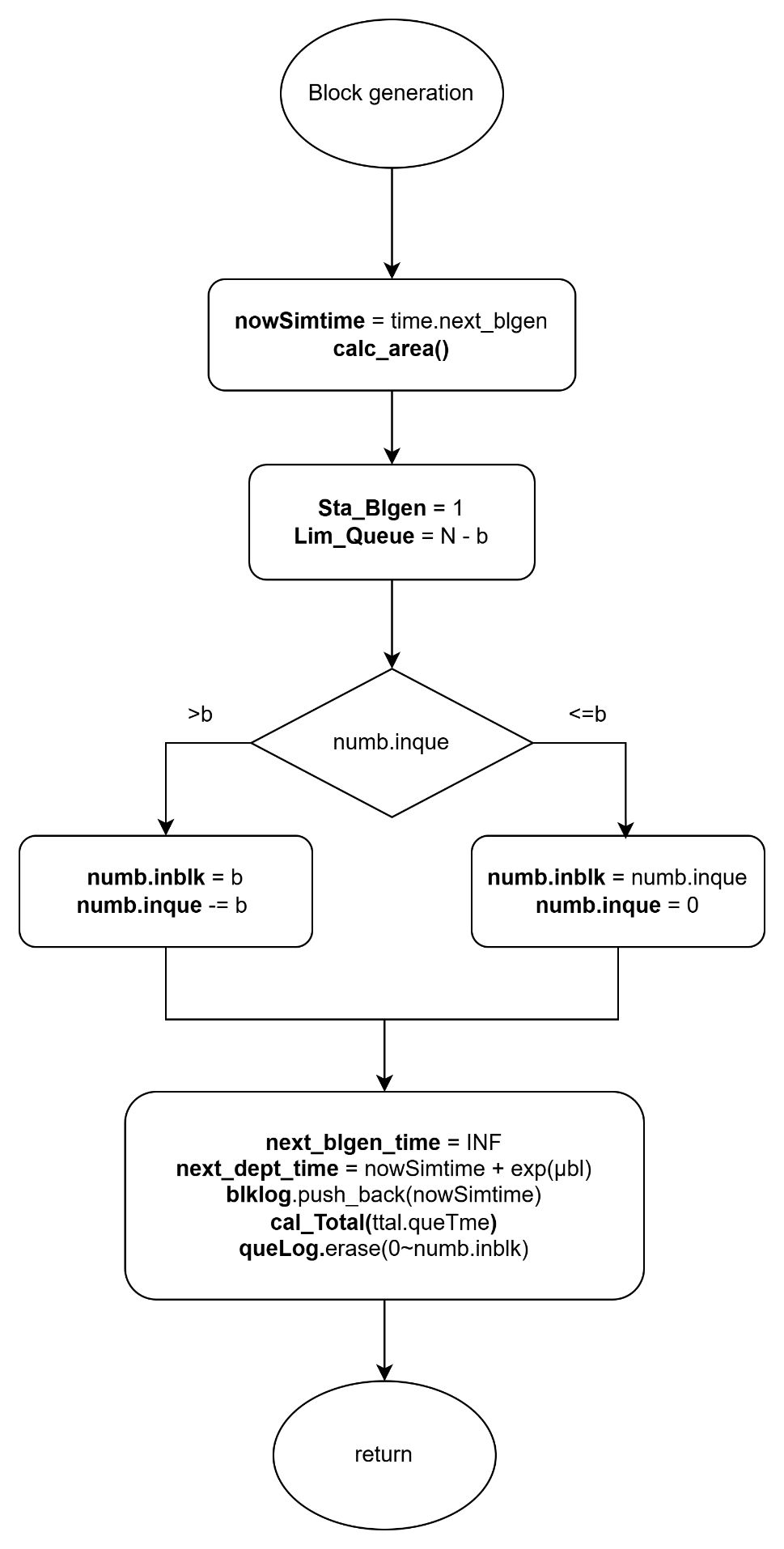


Figure 4‑3Flow chart of block generation subprogram

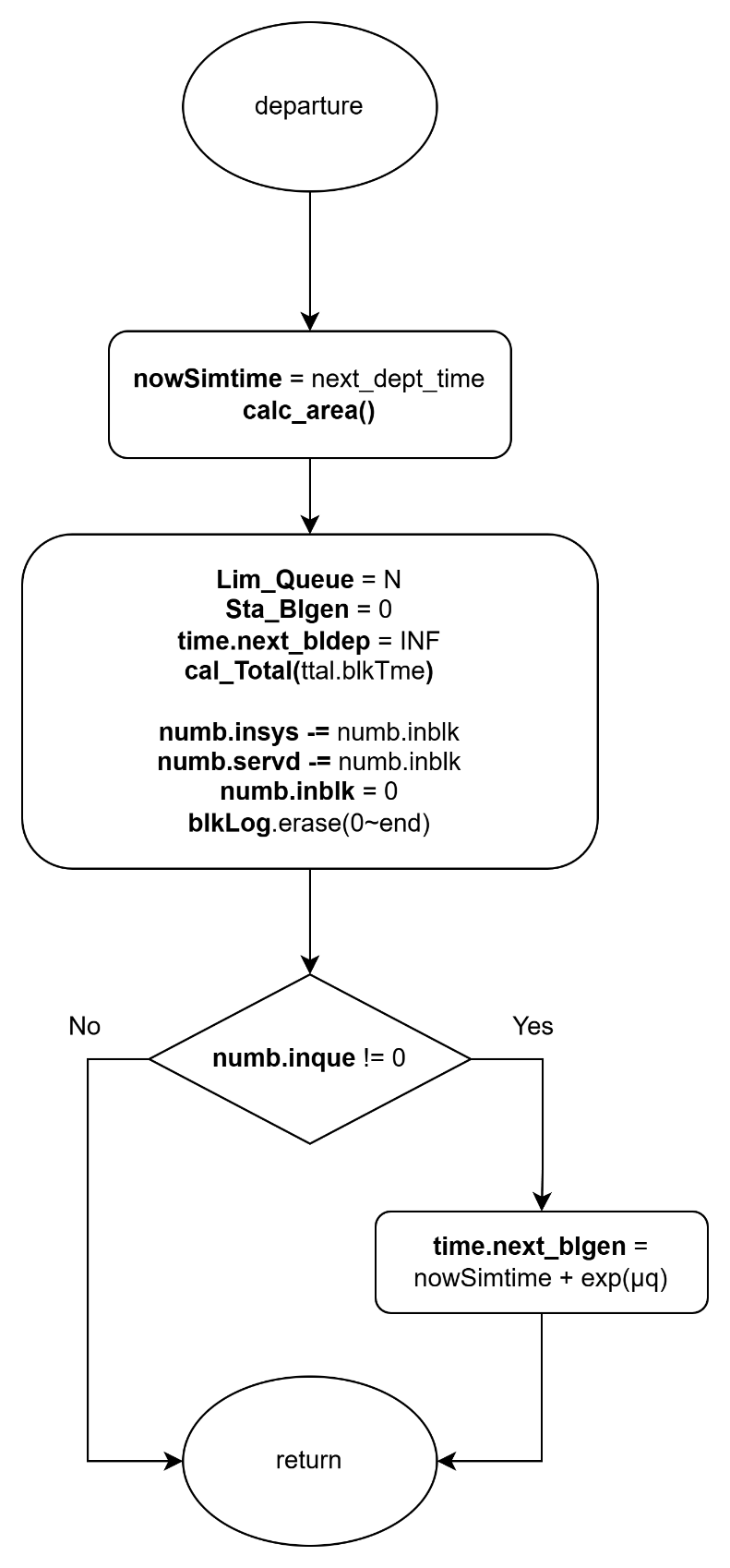


Figure 4‑4 Flow chart of block departure subprogram

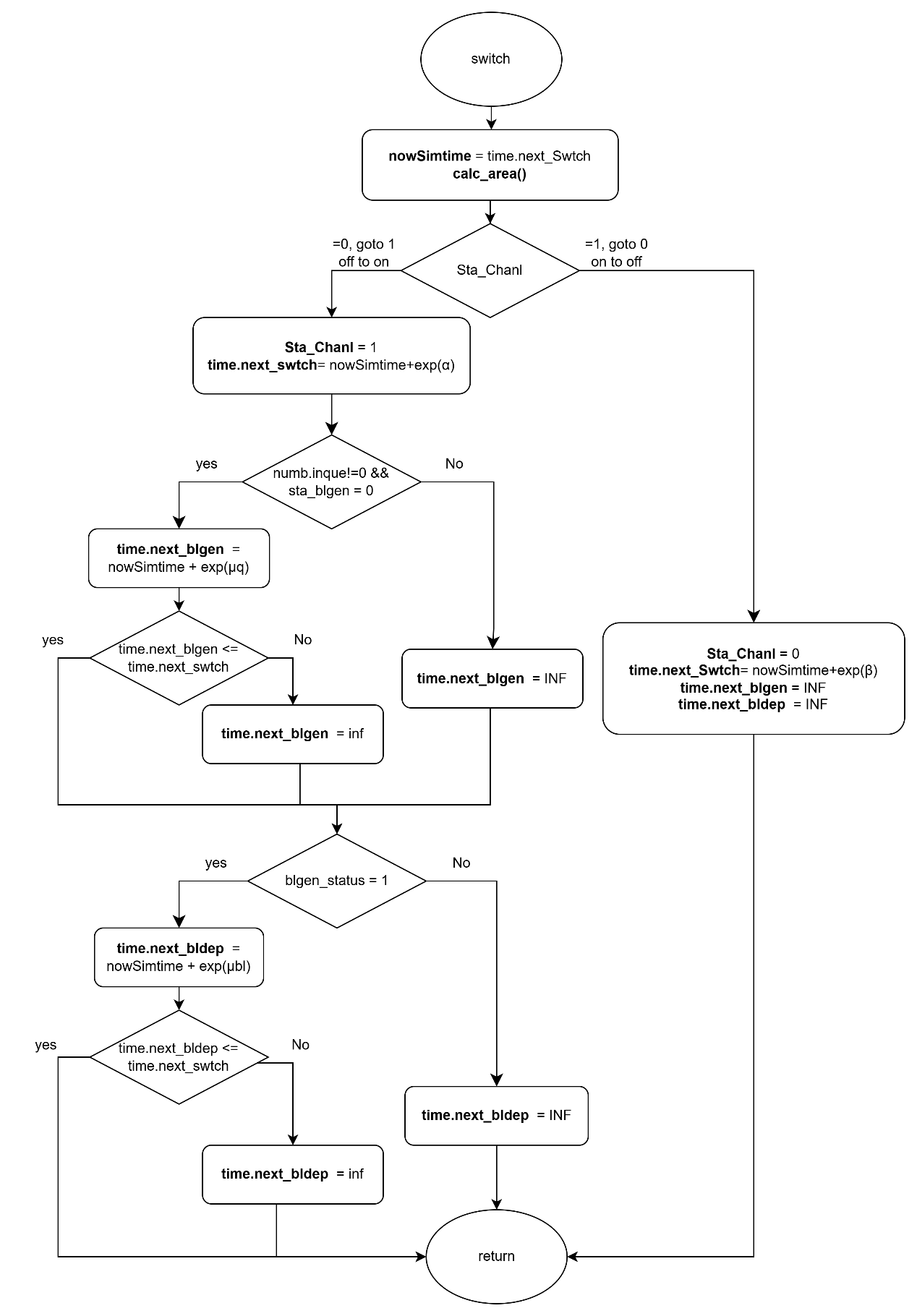


Figure 4‑5 Flow chart of switch subprogram

### 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.