Discrete event traffic simulation
Laboratory Report 3

RTEL 2024/2025 **Class: 2MEEC_T01**

António Lopes, Pedro Duarte

1 Introduction

This report outlines the development of a Traffic Simulation model designed to replicate the behavior of a Call Center System. The present project is built upon previous works developed in [1] and [2]. The main objective was to develop a system capable of receiving discrete call events, processed by either a general-purpose call center operator or an area-specific call center operator. After developing a functional system, the final step was to design the system parameters in order to satisfy the requested minimal performance goals. This project was developed as part of the Telecommunications Networks course at FEUP, utilizing Python as the programming language for its simplicity and versatility.

2 Methodology

2.1 Event generation

This work employs a discrete event simulation framework to model a telecom operator's call center, where the simulation state evolves based on **arrival** and **departure** events. An arrival event occurs when a call enters a system (general or specific). Departure events signify the end of a call's processing, freeing resources and potentially triggering subsequent events.

The general-purpose system has a *finite-length queue*, blocking calls when full, while the area-specific system uses an *infinite queue*, where calls are delayed until resources become available. As it was necessary to sort the events between both systems, the *Event* class created in [1] was updated, being structured as follows:

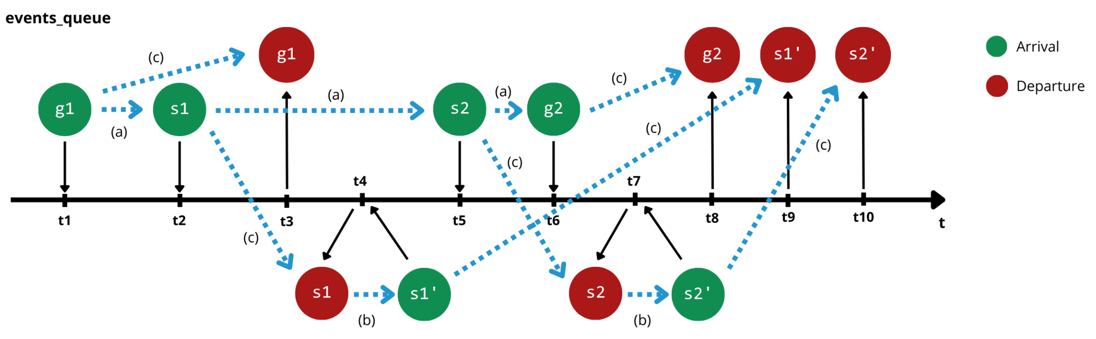
Table 1: Event class parameters.

identifier	event ID used for tracking event flow.
type	event type: "arrival" or "departure"
curr_system	system that will process the event: "general" or "specific"
target_system	event target system: "general" or "specific"
time	time of event processing

In this setup, a priority-based event queue manages event execution by chronological order. Arrival events are generated in two occasions: (a) when processing an arrival on the general system; (b) when processing a departure on the general system of a specific targeted event. Departure events are generated every time a system processes an arrival and has free resources (c). An example of this process is depicted in Figure 1, containing a table where can be seen the progress of each system.

2.2 Simulation Program

The *Simulation* class developed in [2], was adapted and divided into two different classes: 1) *System*, used for handling the arrivals and departures of events in each system; 2) *Simulation*, which has two *System* objects (general and specific) and is responsible for orchestrating the



System	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
General_purpose	g1	g1 s1	s1		s2	s2 g2	g2			
Specific				s1'	s1'	s1'	s1' s2'	s1' s2'	s2'	

Figure 1: Event generation example.

processing of events. Figure 2 presents the flow diagram of the main logic functions for each class, with the corresponding functions described below.

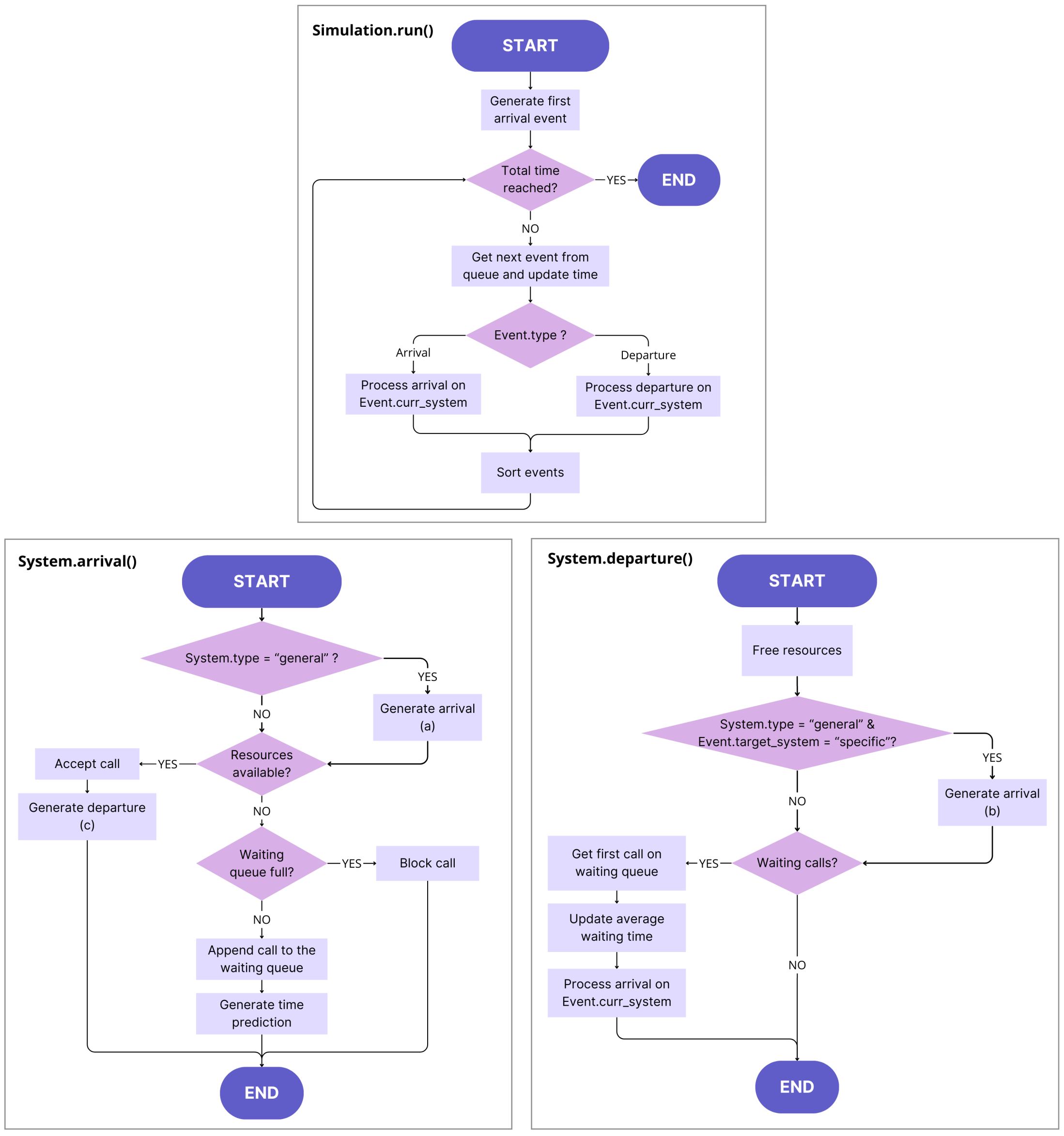

Simulation.run()
START
Generate first arrival event
Total time reached?
YES
END
NO
Get next event from queue and update time
Event.type ?
Arrival
Departure
Process arrival on Event.curr_system
Process departure on Event.curr_system
Sort events
System.arrival()
START
System.type = "general" ?
YES
Generate arrival (a)
NO
Resources available?
YES
Accept call
Generate departure (c)
NO
Waiting queue full?
YES
Block call
NO
Append call to the waiting queue
Generate time prediction
END
System.departure()
START
Free resources
System.type = "general" & Event.target_system = "specific"?
YES
Generate arrival (b)
NO
Waiting calls?
YES
Get first call on waiting queue
Update average waiting time
Process arrival on Event.curr_system
NO
END

Figure 2: Flow diagrams of Simulator functions.

Simulation.run() - The simulation starts by generating an initial event and runs a loop until the specified simulation time is reached. In each loop, the *type* and *current system* of the next event is checked and the event is processed accordingly. After processing, it is necessary to sort the events queue by the time of each event, preserving the chronological order.

System.arrival() - The function responsible for processing the arrival of an event e_1 starts by generating the next arrival event, e_2, if e_1 is arriving at the general system (a). It then accepts the call and generates the associated departure event if the system has resources available (c). If the system is full and its waiting queue is free, the event is appended to the waiting queue and a waiting prediction time is generated. Otherwise the call is blocked.

System.departure() - The function responsible for processing the departure of an event e_1 starts by freeing the resource used by e_1. If e_1 requires contact with an area-specific operator and it is departing from the general system, an arrival event associated with e_1 is generated, but this time directed to the specific system. Finally, it checks for waiting calls on the system and calls the arrival function for processing the first event on the waiting queue, if it exists.

Metrics such as delay probabilities, block probabilities, and average waiting times are collected during the simulation. This structure ensures efficient simulation of system dynamics and accurate computation of performance metrics.

2.3 Waiting Time Prediction

The waiting time prediction algorithm updates the expected waiting time for calls queued in the general-purpose system. Each call that is transferred from the queue to a general operator stores the respective waiting time on a list W, used for compute the average of the waiting time.

The prediction dynamically updates whenever a new call is added to the queue, based on:

- n_w - The number of calls currently waiting in the queue.
- avg_w - The historical average waiting time on queue.

Every time that a call enters the waiting queue, the expected waiting time is predicted by multiplying n_w by avg_w.

3 Simulation Results

3.1 System Parameters and Performance Goals

The simulations were conducted using a call arrival rate of $\lambda = 80$ calls/hour ≈ 0.0222 calls/s. The system design included varying the following simulation parameters: numbers of general-purpose, n_g, and area-specific operators, n_s, as well as the waiting queue length, L, for the general-purpose system. Each simulation ran for a total duration of $T = 432,000$ seconds, equivalent to 5 days of operation, running every parameter combination at least 30 times.

The performance goals for the system were defined as follows: the delay probability, P_d, should not exceed 30%, the block probability, P_b, should remain below 2%, the average waiting time, avg_w in the general-purpose queue should not exceed 30 seconds, and the total average delay until a call was answered by a specific-purpose operator, avg_d, should stay under 90 seconds. These criteria were used to evaluate the effectiveness of the system configurations.

The simulation results were stored in a database, and analytical functions were developed to identify simulations that meet the desired requirements. Table 2 presents relevant simulation design results, with 90% of confidence interval. The parameters obtained for the system design where: $n_g = 4,\ n_s = 5,\ L = 2$. From Table 2 it is possible to conclude that P_d, P_b and avg_w are only affected by n_g and L, while n_s has a dominant influence on avg_d. Note that avg_d is presented as the sum of the average delays in both systems plus the average service time of the general system.

Table 2: System design results.

n_g	n_s	L	P_d (%)	P_b (%)	avg_w (s)	avg_d (s)
4	**5**	**2**	15.00 ± 00.22	01.84 ± 00.07	23.73 ± 0.17	77.44 ± 0.21
4	5	3	16.67 ± 00.18	00.65 ± 00.03	25.50 ± 0.18	78.76 ± 0.30
4	6	2	15.01 ± 00.19	01.85 ± 00.05	23.97 ± 0.19	70.50 ± 0.17
5	5	2	6.25 ± 00.12	00.57 ± 00.03	19.71 ± 0.25	75.70 ± 0.28

3.2 Delay Distribution and Prediction Error

The delay distribution of calls in the general-purpose system is depicted in Figure 3, showing the frequency of waiting times across different time intervals. Alongside this, the system's ability to predict waiting times was evaluated by comparing the predicted waiting times against the actual observed times. The histogram in Figure 4 illustrates the distribution of the prediction errors. On average, the absolute error in prediction was 16.38 seconds, representing a relative error of 68.8%. This results allow to evaluate the time prediction algorithm. Given the basic nature of the used algorithm, an average error of 16.38 seconds can be considered tolerable in this context. However, the relative error associated can be problematic if it stands the same for higher values of average waiting times.

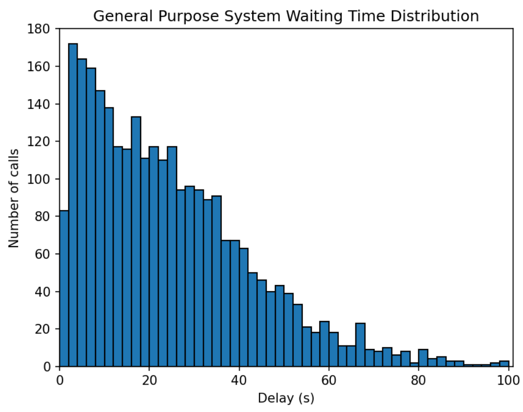


Figure 3: Distribution of waiting times on general-purpose system.

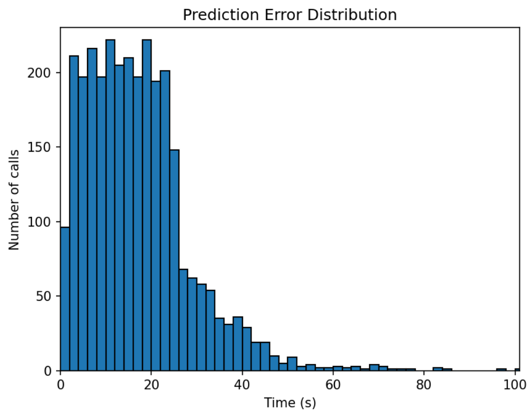


Figure 4: Prediction error for the waiting time in the input queue.

4 Sensitivity Analysis

For performing the system sensitivity analysis, the test variable considered was the average total delay of the calls, since they arrive at the general-purpose answering system until they are answered by the area-specific answering system, avg_d. The variation of the average total delay with the offered traffic is depicted in Figure 5. As the offered traffic increases, the average total delay exhibits a noticeable upward trend due to increased queuing and system congestion.

As described in Section 3.1, several simulations were run in order to get the confidence intervals for the simulation metrics. At the nominal rate of 80 calls/hour, the 90% confidence interval was the following:

$$\text{Confidence Interval} = 77.44 \pm 0.21\,\text{seconds}.$$

This confidence interval implies that, with 90% certainty, the true average total delay lies between 77.23 seconds and 77.65 seconds. The relatively narrow interval demonstrates the reliability of the estimator under the given simulation settings.

The analysis reveals that the system performs effectively at nominal traffic levels, meeting the performance goal of keeping the average total delay below 90 seconds. However, as the offered

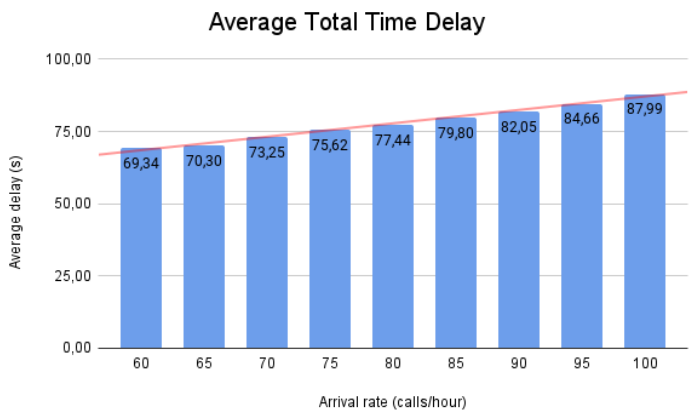


Figure 5: Variation of average total delay with offered traffic.

traffic increases beyond 80 calls/hour, the delay begins to reach this threshold, underscoring the importance of maintaining optimal operator allocation and queue management to prevent system congestion. This study highlights the system's sensitivity to traffic variations and provides actionable insights for resource planning and system optimization.

5 Conclusion

In conclusion, the traffic simulation model developed successfully meets the intended performance goals for a call center system, demonstrating its robustness under nominal traffic conditions. By implementing discrete event simulation and parameter optimization, the system achieves low blocking and delay probabilities, as well as acceptable average waiting times, ensuring efficient call handling. Sensitivity analysis further highlights the model's responsiveness to traffic variations, emphasizing the criticality of resource allocation and queue management in maintaining optimal performance. All source code and simulation data is present in [3].

References

[1] P. Duarte, A. Lopes. Discrete event traffic simulation - Laboratory Report 1. October 2024

[2] P. Duarte, A. Lopes. Discrete event traffic simulation - Laboratory Report 2. October 2024

[3] P. Duarte, A. Lopes. GitHub Repository: `https://github.com/Pektro/RTEL` . Last accessed: November 2024.