Simulation of an M/M/1 Queue with OMNeT++

UGE: M2 SIA - MSQue Lab Report

Luca Uckermann Nikethan Nimalakumaran

2025-02-02

This report investigates the simulation of an M/M/1 queue using the INET framework within the OMNeT++ simulation environment. The focus is on analyzing queuing behavior under realistic conditions by configuring mean arrival and service rates with exponential probability distributions. The simulation generates output results in SQLite format, allowing advanced analysis and evaluation of queuing metrics such as queue length, queuing time and system performance.

The study uses both theoretical and simulation-based approaches. Source code modifications, including tracking packet arrival times and calculating queuing delays, enhance the analytical capabilities of the simulation. Metrics such as mean inter-arrival time, mean service time, system load and time-averaged queue length are examined through SQL queries, with results presented as comparative tables and graphical visualizations.

Table of contents

The goal of this report is to simulate an M/M/1 queue using the INET queuing library (INET Framework n.d.a) and OMNeT++ simulator (OMNeT++ n.d.a). The simulation is based on the INET queuing tutorial (INET Framework n.d.b) and includes configuring the mean arrival rate and service rate, generating output results as SQLite files and analyzing queuing time and queue length. The report also includes a comparative study of theoretical and simulation values, examination of queue length vector metrics and verification of Little's Law (Little and

Graves 2008). The source code modifications, SQL queries and simulation runs are detailed, providing insight into the simulation and analysis of M/M/1 queues.

The following commands were executed to install the INET framework and OMNeT++ simulator using the `opp_env` tool (OMNeT++ n.d.b):

```
pip install opp_env
opp_env install --init\
  -w inet-workspace inet-4.5.4 omnetpp-6.1.0
```

The `opp_env` tool simplifies the installation process by managing dependencies and environment variables. The INET framework version `4.5.4` and OMNeT++ version `6.1.0` have been installed in the `inet-workspace/` directory.

To run the OMNeT++ IDE, the following commands were executed:

```
cd inet-workspace
opp_env shell
omnetpp
```

The simulation was run using the IDE, which provides a graphical user interface (GUI) for configuring and running simulations. The `queuing` tutorial was selected and the simulation started (`/inet-4.5.4/tutorials/queuing/omnetpp.ini`).

1 Simulate an M/M/1 queue using the INET queuing library and OMNeT++ simulator. Use the queuing tutorial located at `inet/tutorials/queuing`.

To simulate the M/M/1 queue, the `PacketQueue` tutorial is used. After starting the simulation, it runs for 10 seconds and packets are sent from the `producer` to the `queue` and then to the `consumer`.

Figure 1 shows the simulation of the M/M/1 queue with the three components mentioned above. The GUI is used to start and stop the simulation, the results are displayed graphically and additional information is printed to the console.

Figure 2 shows the producer sending packets to the queue. Here `(Packet)producer-16` is being sent to the queue by the producer. The producer has already created *17* packets while the collector has only collected *8* packets.

Figure 3 shows the queue sending packets to the consumer. Here `(Packet)producer-4` is being sent from the queue to the consumer. The queue contains *4* packets and *5* packets have been sent to the consumer.

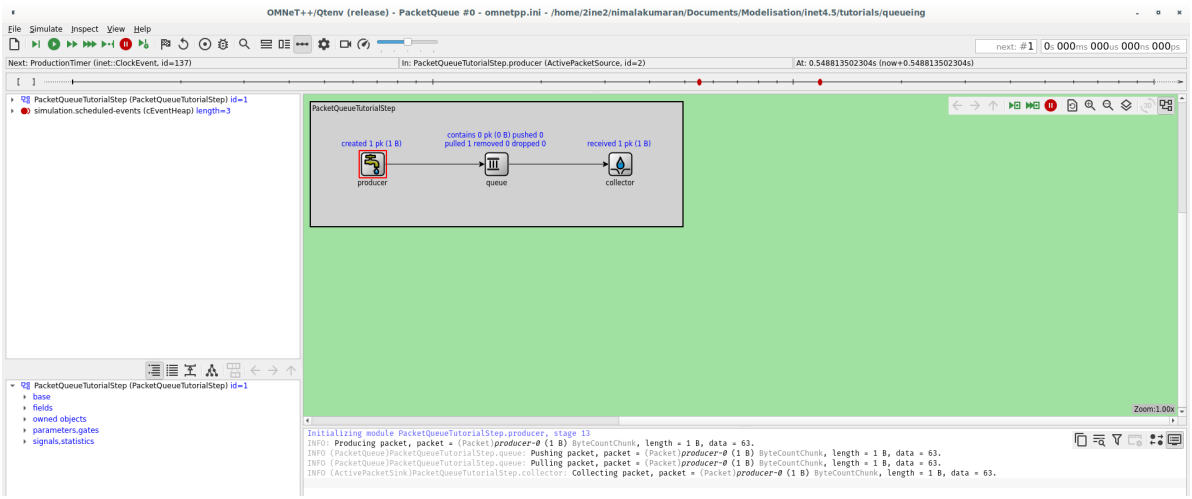

Figure 1: PacketQueue

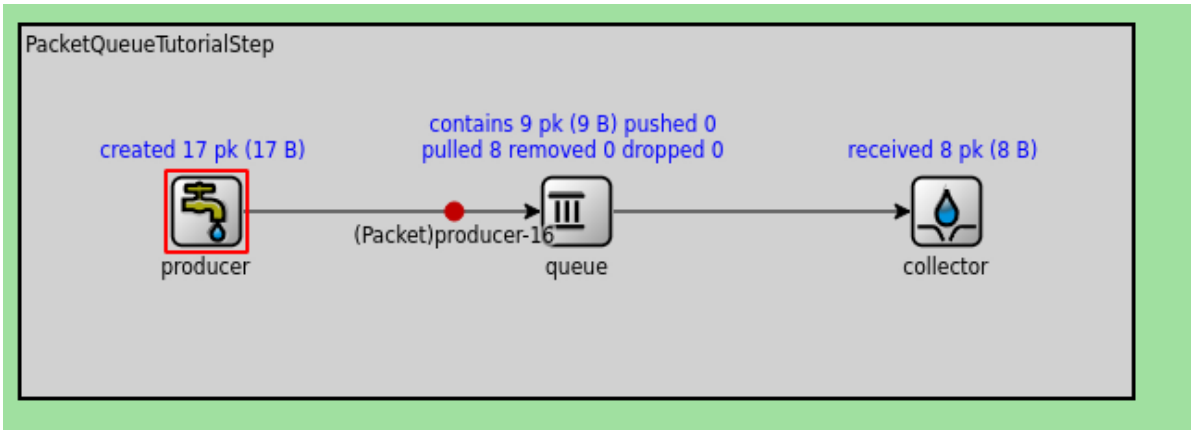

Figure 2: Producer sending packets to queue

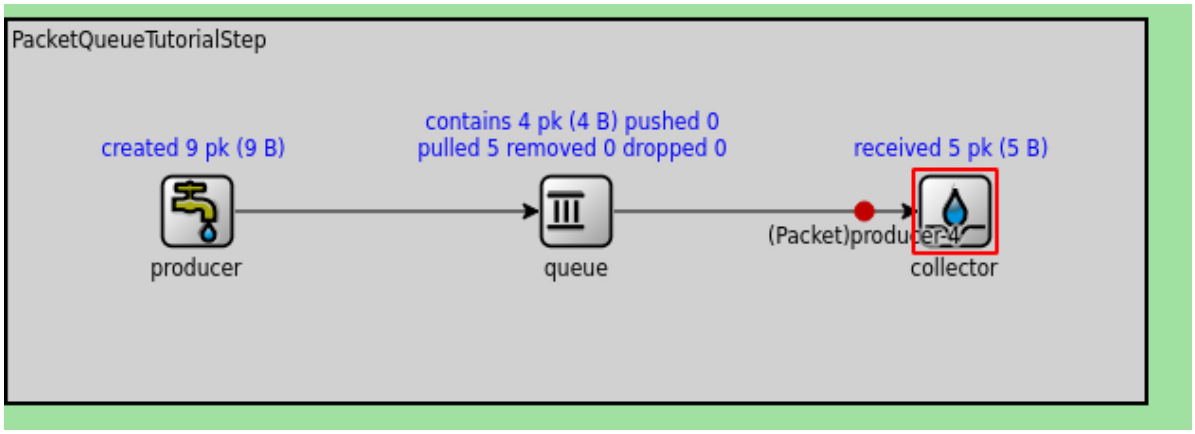

Figure 3: Queue sending packets to consumer

2 Configure the mean arrival rate of packets to `0.75` clients/s and the mean service rate to `0.79` clients/s with exponential probability distributions.

To change the mean arrival rate and service rate, the following configuration are changed in the `omnetpp.ini` file:

```
[Config PacketQueue]
...
*.producer.productionInterval\
  = uniform(0s, 1s)
*.collector.collectionInterval\
  = uniform(0s, 2s)
```

to:

```
[Config PacketQueue]
...
*.producer.productionInterval\
  = exponential(0.75s)
*.collector.collectionInterval\
  = exponential(0.79s)
```

Where `productionInterval` is the mean arrival rate and `collectionInterval` is the mean service rate, both in clients (packets) per second.

2.1 Uniform (before):

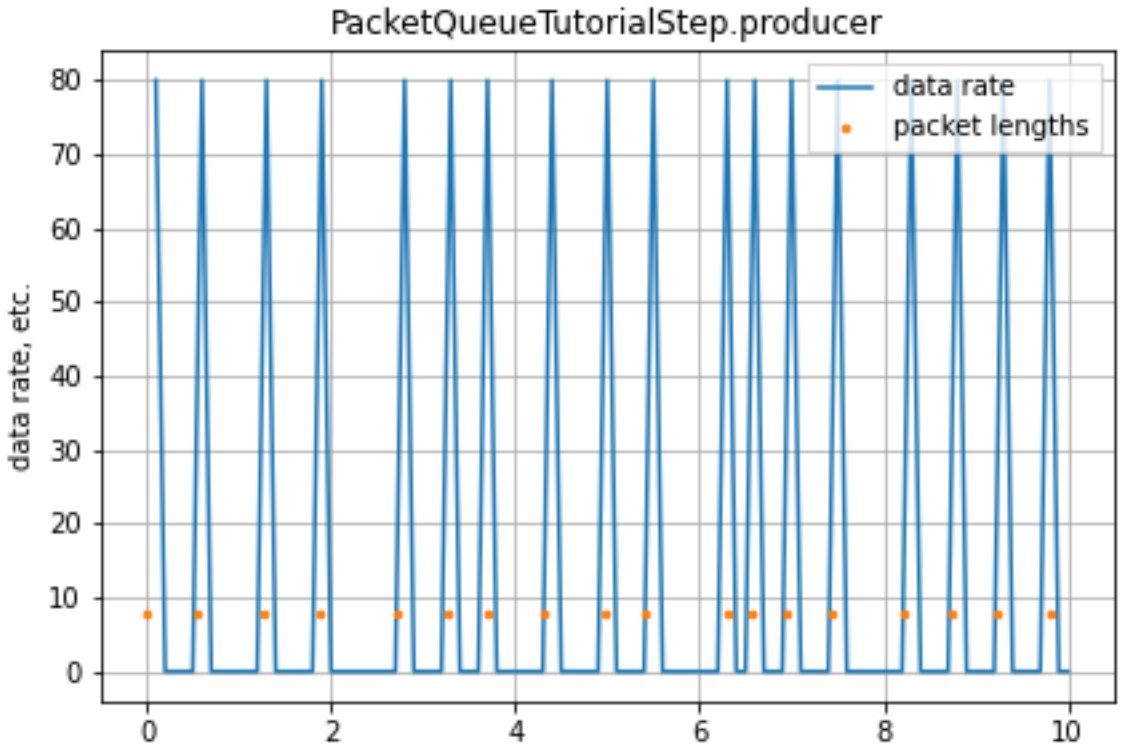

Figure 4: Producer with uniform distribution

Figure 4, Figure 5 and Figure 6 illustrate the behavior of the producer, collector and queue components under the default uniform distribution, before any modifications have been applied. Figure 4 highlights the output of the producer, showing the relationship between data rate and packet lengths over time. The consistent and periodic behavior aligns with the expected uniform distribution. Figure 5 shows the activity of the collector, capturing metrics such as data rate, dropped data rate and packet-related statistics like delay, jitter and length. The interplay of these variables shows the performance of the collector under uniform input conditions. Figure 6 illustrates the dynamics within the queue, showing incoming and outgoing data rates, packet lengths and queue lengths, along with the evolution of queuing times and bit-level operations. The smooth rise and stabilization of queue lengths and related parameters illustrate the ability of the queue to handle uniform traffic inputs. Together, these figures provide a detailed snapshot of the system's performance under a uniform input distribution and serve as a baseline for analyzing subsequent changes.

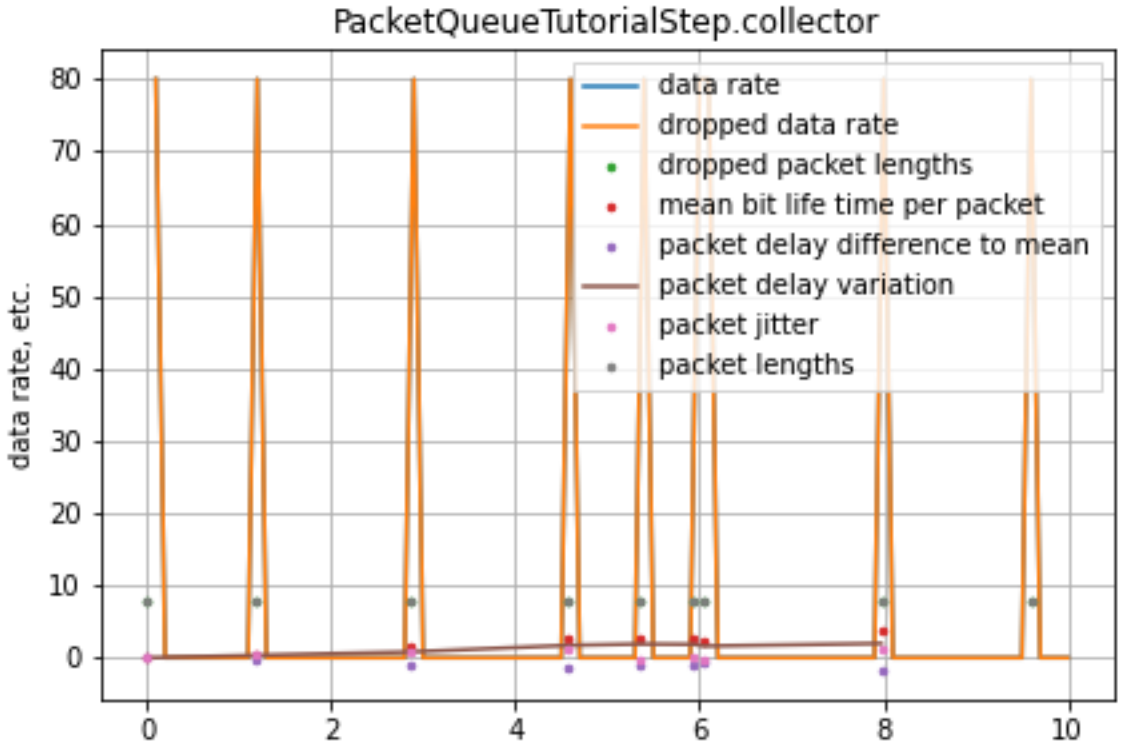

Figure 5: Collector with uniform distribution

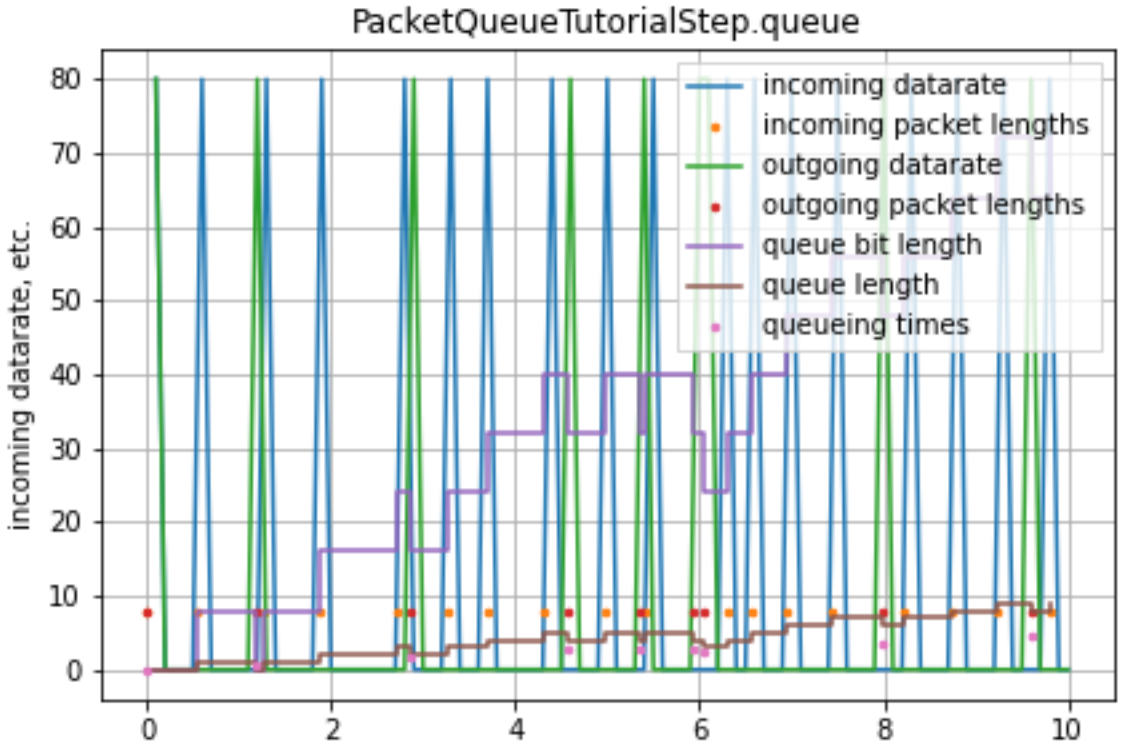

Figure 6: Queue with uniform distribution

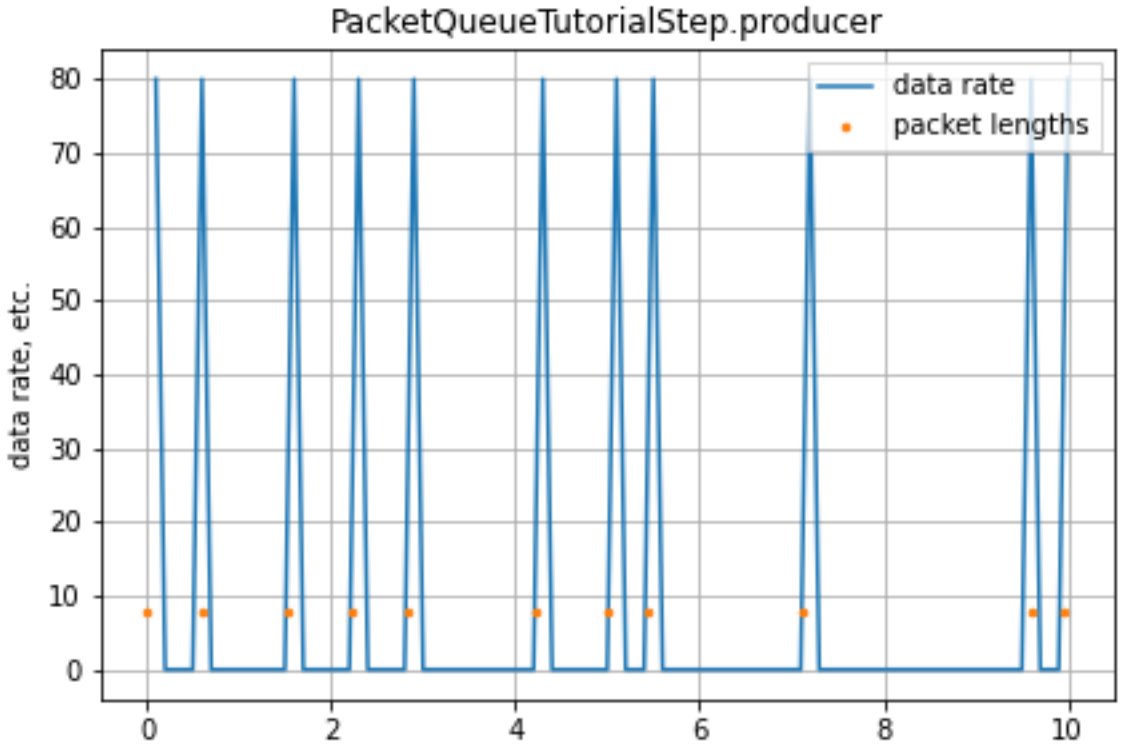

Figure 7: Producer with exponential distribution

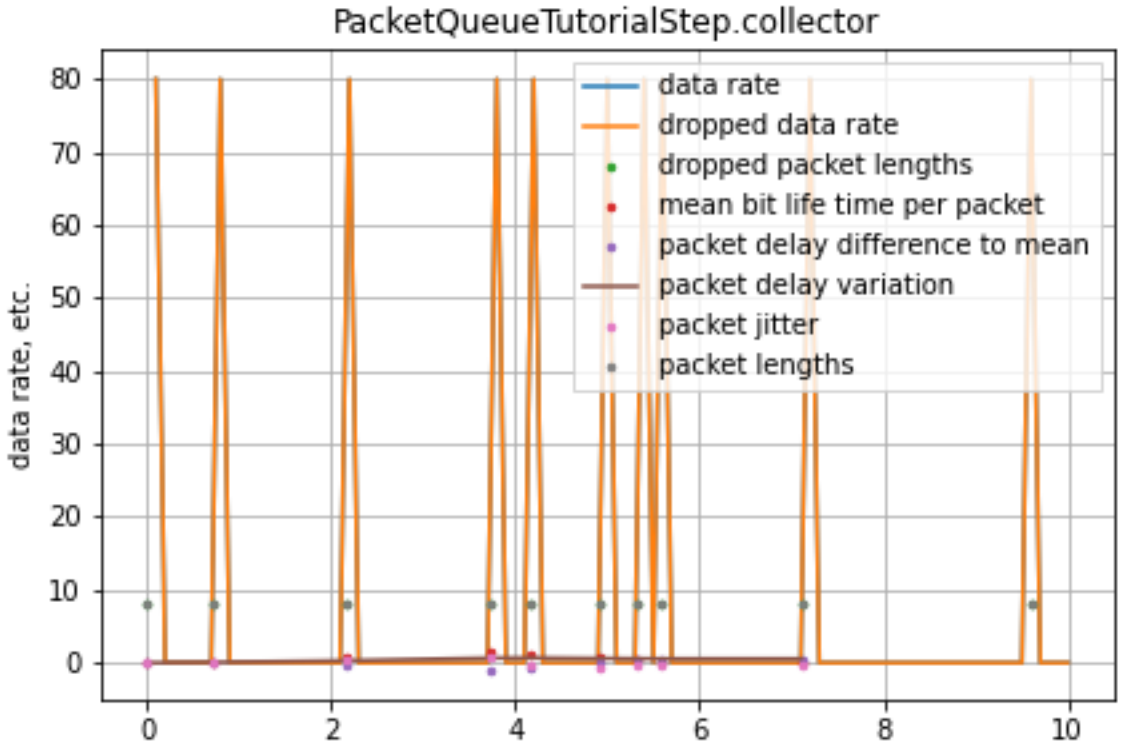

Figure 8: Collector with exponential distribution

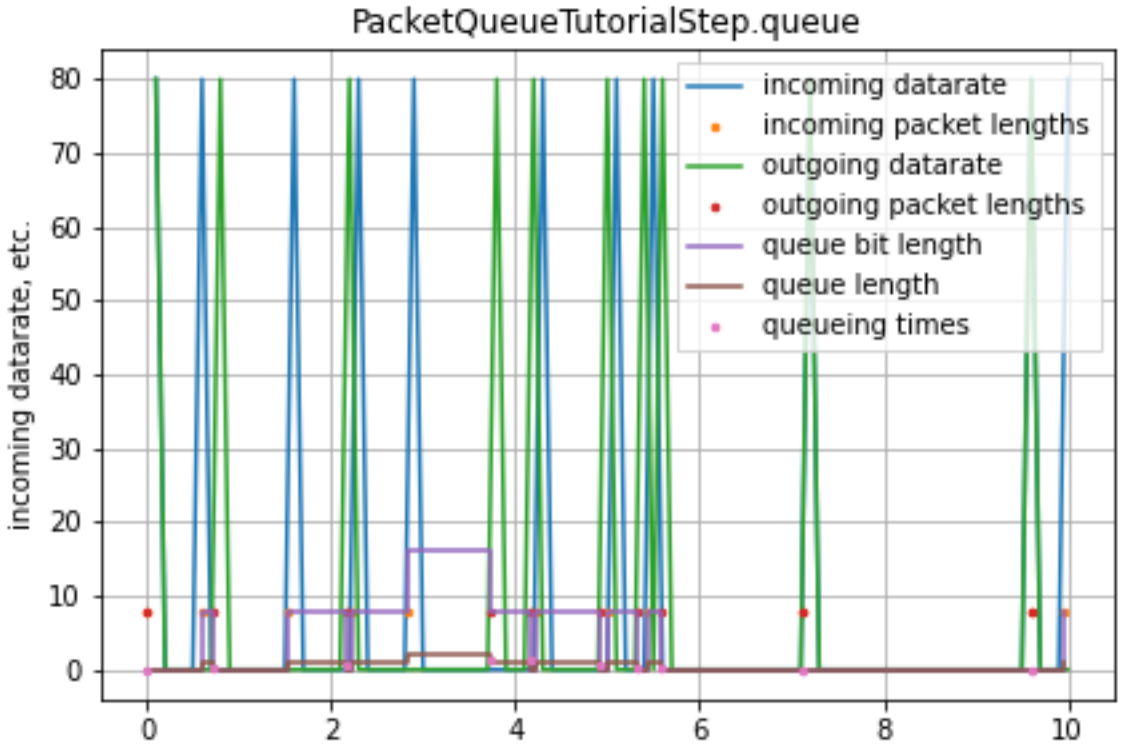

Figure 9: Queue with exponential distribution

2.2 Exponential (after):

Figure 7, Figure 8 and Figure 9 show the producer, collector and queue components with an exponential distribution after the changes were made. The exponential distribution is used to model the mean arrival rate and service rate of packets, providing a more realistic representation of system behavior. By configuring the mean arrival rate to *0.75 clients/s* and the mean service rate to *0.79 clients/s*, users can simulate packet arrivals and processing times based on these parameters. Figure 7 highlights the behavior of the producer, showing the data rate and packet lengths over time. Unlike the uniform distribution, the exponential distribution introduces more variability, resulting in irregular intervals between packet bursts. This reflects the non-uniform arrival of packets, a more realistic scenario in dynamic traffic systems. Figure 8 illustrates the performance of the collector. The variability in packet delays and occasional spikes in dropped data rate demonstrate the ability of the system to handle fluctuating input traffic, while revealing areas of congestion or inefficiency during bursts. Figure 9 shows the dynamics of the queue, including incoming and outgoing data rates, packet lengths, queue lengths and queuing times. The exponential distribution creates periods of congestion in the queue, as evidenced by spikes in queue lengths and queuing times. This variability highlights the behavior of the queue under non-uniform traffic and provides insight into resource utilization and potential bottlenecks. These figures collectively show the performance of the system under an exponential input distribution, allowing a detailed evaluation of its robustness and efficiency in handling dynamic traffic conditions.

From here on, the exponential distribution is used for the mean arrival rate and the service rate.

3 Modify omnetpp.ini to generate output results as SQLite files:

```
[General]
...
**.scalar-recording = true
**.vector-recording = true
outputvectormanager-class =\
  "omnetpp::envir::SqliteOutputVectorManager"
outputscalarmanager-class =\
  "omnetpp::envir::SqliteOutputScalarManager"
```

Figure 10 shows the changed `Type` column of the output files (`.sca` and `.vec`), it is now `SQLite3 database` for the scalar and vector files. The "DB Browser for SQLite" is used to open the output files and analyze the simulation results. Generating output files as SQLite

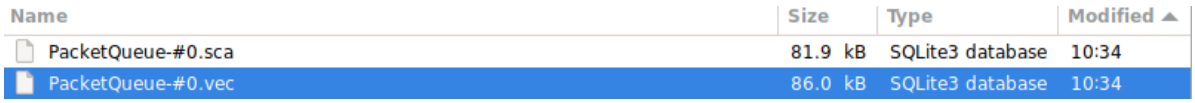

Figure 10: SQLite output files

databases allows to store, retrieve and analyze simulation data using SQL queries. This approach improves data management, facilitates interpretation and supports advanced analysis for performance evaluation and optimization.

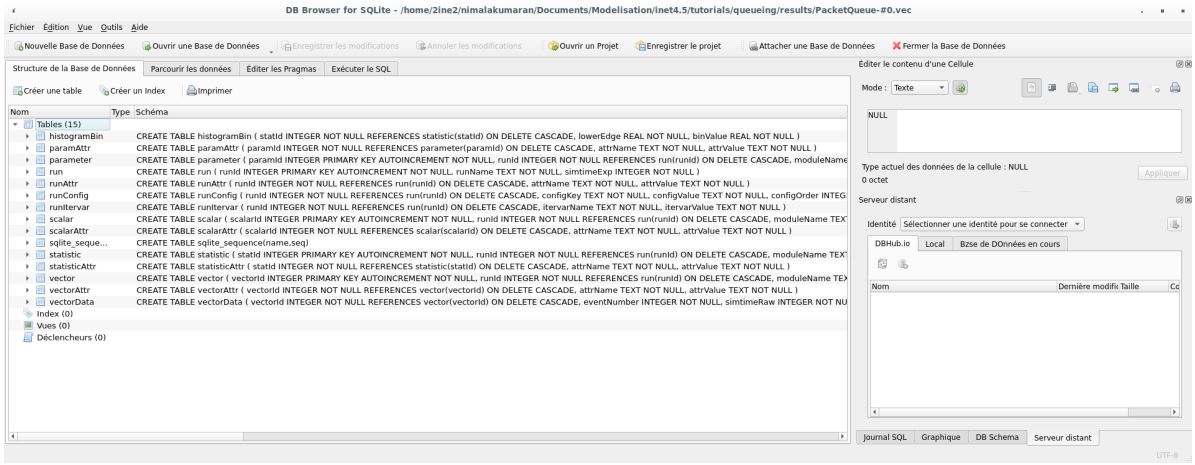

Figure 11: DB Browser for SQLite

Figure 11 shows the "DB Browser for SQLite" application with an opened vector file. It contains *15* tables with the simulation results. The overview is similar to the one provided by the OMNeT++ IDE, but the SQLite format allows the use of SQL queries, which will be used later.

4 Run the simulation and report the time-averaged queue length and queuing time. Please plot the curve of the scalar values of the queuing times.

Figure 12 shows the value of the time-averaged queue length, which is about *0.477*. This value is extracted with the following SQL query:

```
SELECT * FROM scalar WHERE scalarName =\
  'queueLength:timeavg';
```

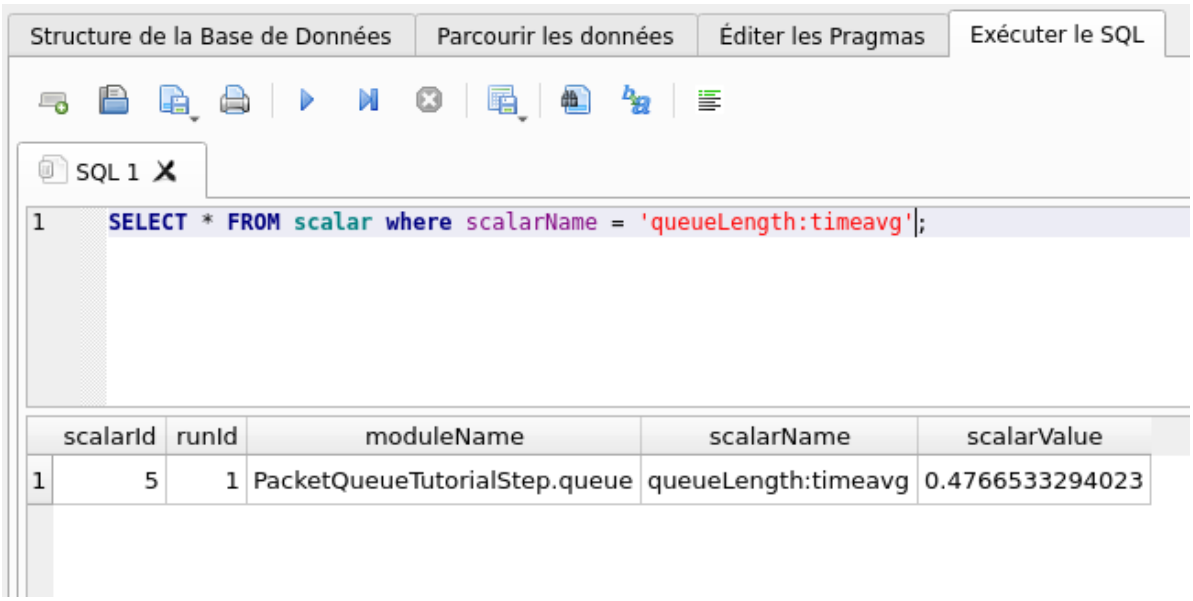

Figure 12: Time-averaged queue length

This allows to extract and analyze values of interest without having to search through the entire output file.

Figure 13 shows the queuing time curve with the following statistics generated from the scalar output:

- **Count:** 10
- **Mean:** $0.472422s$
- **StdDev:** $0.556602s$
- **Variance:** $0.309805s^2$

The curve illustrates the temporal behavior of packet delays within the queue, with key phases of low initial delay, a peak congestion period and eventual stabilization. Initially, queuing times are minimal, reflecting low traffic or under-utilization of the server. As the simulation progresses, increasing traffic intensity leads to longer queuing delays, which peak when the arrival rate approaches or exceeds the service rate. This phase highlights the congestion and reduced efficiency of the system. Subsequently, queuing times gradually decrease, indicating the server's ability to handle the backlog or a reduction in packet arrivals. Finally, the system stabilizes, with queuing times flattening to near zero, indicating the achievement of steady-state performance.

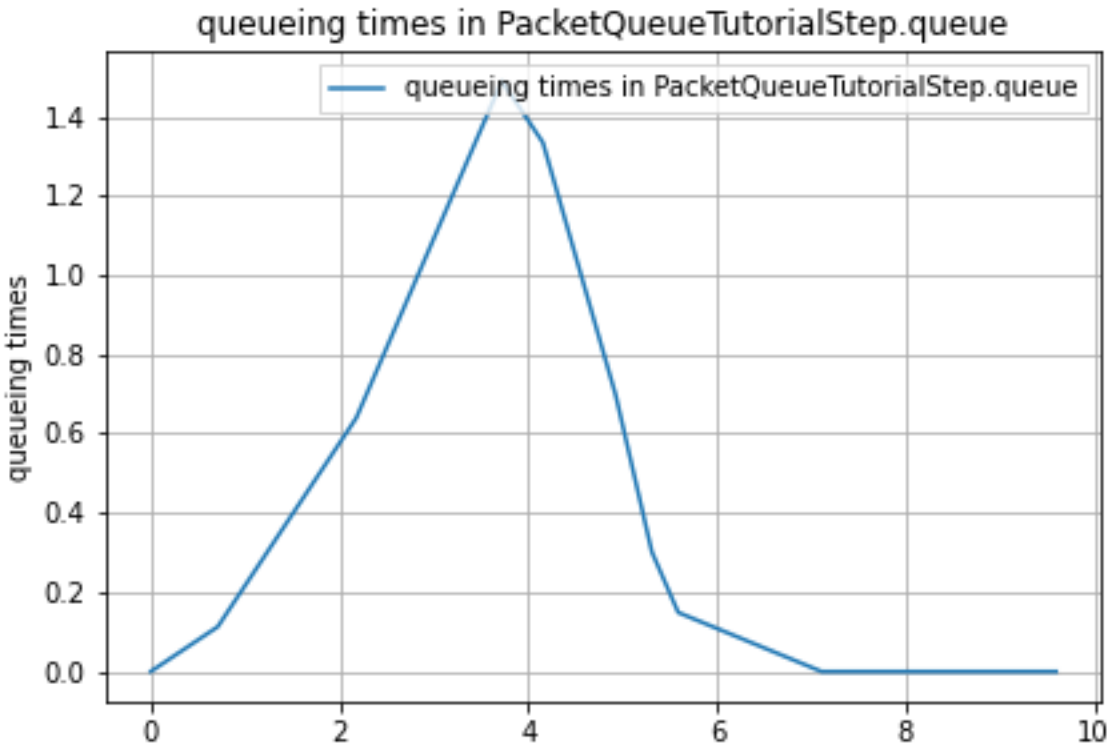

Figure 13: queuing time

5 Modify the source code (`PacketQueue.cc`) to measure queuing time by adding the packet arrival time. Explain the reasons for this modification.

To log packet arrival times during simulation, the `/inet-4.5.4/src/inet/queuing/queue/PacketQueue.cc` file is modified:

```
void PacketQueue::pushPacket(
  Packet *packet,
  cGate *gate
) {
    Enter_Method("pushPacket");
    take(packet);
    packet->\
      addTagIfAbsent<CreationTimeTag>()->\
        setCreationTime(simTime());
    ...
}
```

The `pushPacket()` method is modified to add a `CreationTimeTag` to the packet, which stores the arrival time of the packet. This modification allows the arrival time of the packet to be logged, providing insight into the queuing time packets experienced in the system. By tracking packet arrival times, queuing delays can be measured, analyzed and optimized to improve system performance and user experience.

```
Packet *PacketQueue::pullPacket(
  cGate *gate
) {
    Enter_Method("pullPacket");
    auto packet =\
      check_and_cast<Packet *>(queue.front());
    EV_INFO <<\
      "Pulling packet" <<\
        EV_FIELD(packet) << EV_ENDL;

    auto arrivalTimeTag =\
      packet->findTag<CreationTimeTag>();
    if (arrivalTimeTag != nullptr) {
        simtime_t arrivalTime =\
          arrivalTimeTag->getCreationTime();
        EV_INFO <<\
```

```
            "Arrival time for packet: "\
              << arrivalTime << EV_ENDL;
    }
    ...
}
```

The `pullPacket()` method is modified to retrieve the `CreationTimeTag` of the packet and extract the arrival time of the packet. This information is then logged to the console, providing visibility into the queuing time for each packet.

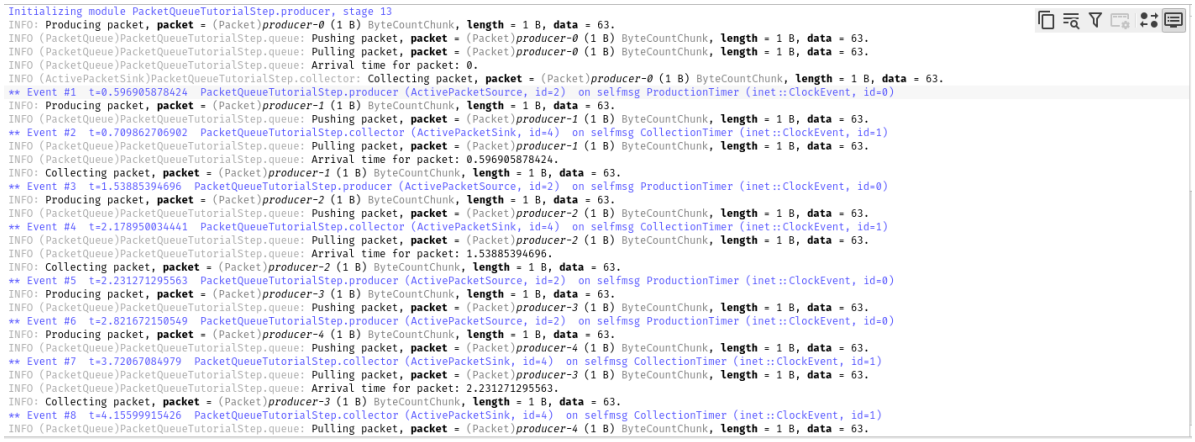

Figure 14: Arrival time outputs in console

Figure 14 displays the arrival time outputs in the console, showing the arrival time for each packet as it is pulled from the queue. Modifying the source code to measure queuing time by adding packet arrival time enhances the analytical capabilities of the simulation, allowing evaluation of queuing delays, system performance and resource utilization.

6 Provide the mathematical formulas for:

6.1 Mean inter-arrival time

$$\lambda = \frac{1}{\text{mean arrival rate}} \tag{1}$$

Equation Equation 1 represents the mean inter-arrival time, which is the reciprocal of the mean arrival rate.

6.2 Mean service time

$$\mu = \frac{1}{\text{mean service rate}} \tag{2}$$

Equation Equation 2 represents the mean service time, which is the reciprocal of the mean service rate.

6.3 System load

$$\rho = \frac{\lambda}{\mu} \tag{3}$$

Equation Equation 3 represents the system load, which is the ratio of the mean arrival rate to the mean service rate.

6.4 Mean queue length (time-averaged)

$$L_q = \frac{\lambda^2}{\mu(\mu - \lambda)} \tag{4}$$

Equation Equation 4 represents the mean queue length, which is the ratio of the square of the mean arrival rate to the product of the mean service rate and the difference between the mean service rate and the mean arrival rate.

6.5 Mean waiting time in the queue(client-based averaged)

$$W_f = \frac{\lambda}{\mu(\mu - \lambda)} \tag{5}$$

Equation Equation 5 represents the mean waiting time in the queue, which is the ratio of the mean arrival rate to the product of the mean service rate and the difference between the mean service rate and the mean arrival rate.

Table 1: Comparison of theoretical and simulation values for M/M/1 queue parameters

[ht]

Parameter	Theoretical Value	Simulation Value
Mean Inter-Arrival Time	1.333	0.959
Mean Service Time	1.266	1.266
System Load	0.949	0.758
Mean Queue Length	17.801	0.477
Mean Waiting Time in Queue	23.734	0.472

7 Compare simulation results with theoretical values. Please do comparative study using a table.

Theoretical values are calculated from the equations above. The simulation results are obtained from the output files generated during the simulation. The comparison is shown in the following table:

Table 1 shows the comparison of theoretical and simulation values for the most important M/M/1 queue parameters. The theoretical values are calculated using Equation Equation 1 to Equation Equation 5, while the simulation values are obtained from the output files generated during the simulation. The mean inter-arrival time is printed due to the modifications made to the source code in the previous section.

8 Perform 10 simulation runs with a simulation time of *10,000* seconds per run. Add the following configuration to `omnetpp.ini`:

To perform *10* simulation runs with a simulation time of *10,000 seconds* per run, the following configuration is added to the `omnetpp.ini` file:

```
repeat = 10
seed-set = ${runnumber}

sim-time-limit = 10000s
```

This configuration allows the simulation to be repeated for `10` runs, each with a simulation time of `10000s`. The `seed-set` parameter ensures that each run uses a different random seed, increasing variability of results. By performing multiple simulation runs with different seeds, a more comprehensive analysis of system performance, queuing behavior and resource utilization can be achieved.

```
** Running in Express mode from event #10  t=4.927834231111 ...
<!> Simulation time limit reached -- at t=10000s, event #26124
** Calling finish() methods of modules
```

Figure 15: Extended simulation runs

Table 2: Time-averaged queue length and average queuing time for 10 simulation runs

[ht]

Run	Time-Averaged Queue Length	Average Queuing Time
0	388	290
1	360	270
2	380	280
3	460	343
4	344	261
5	137	104
6	341	259
7	389	292
8	363	273
9	538	404
Avg	370	278

Figure 15 shows one of the ten executed simulation runs for `10000s` up to event `#26124`. The simulation is repeated ten times with different random seeds to ensure result variability and robustness. The following sections analyze the results.

9 Examine queue length vector average and time-average metrics. Please plot the results in the form of bars.

Table 2 shows the time-averaged queue length and average queuing time for 10 simulation runs. The results illustrate the variability in queue length and queuing time across different simulation runs and highlight the impact of random seed selection on system performance and behavior.

Figure 16 shows the average queuing time for each run, providing insight into the queuing delays experienced by packets in the system. The line graph visualizes the variation in queuing time over multiple simulation runs. Most of the runs' values are close to each other, but runs `5` and `9` show significant differences in lower and higher queuing times, respectively. These can also be seen in the table above, as they provide the lowest and highest values for queuing time.

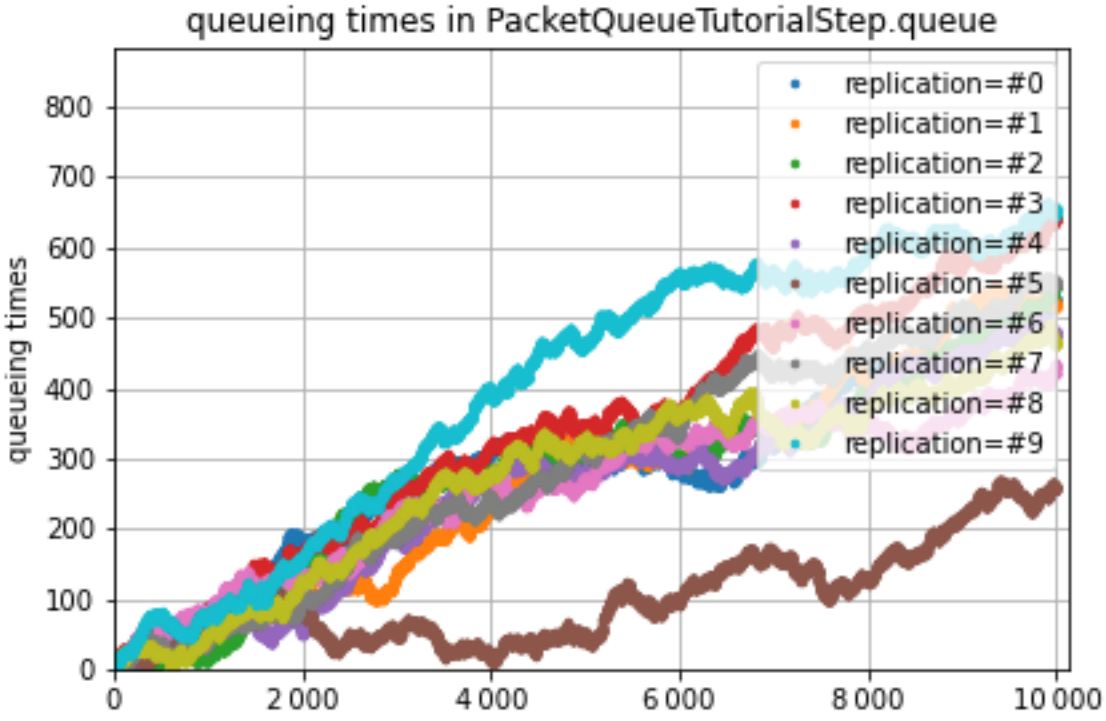

Figure 16: Queue time-average

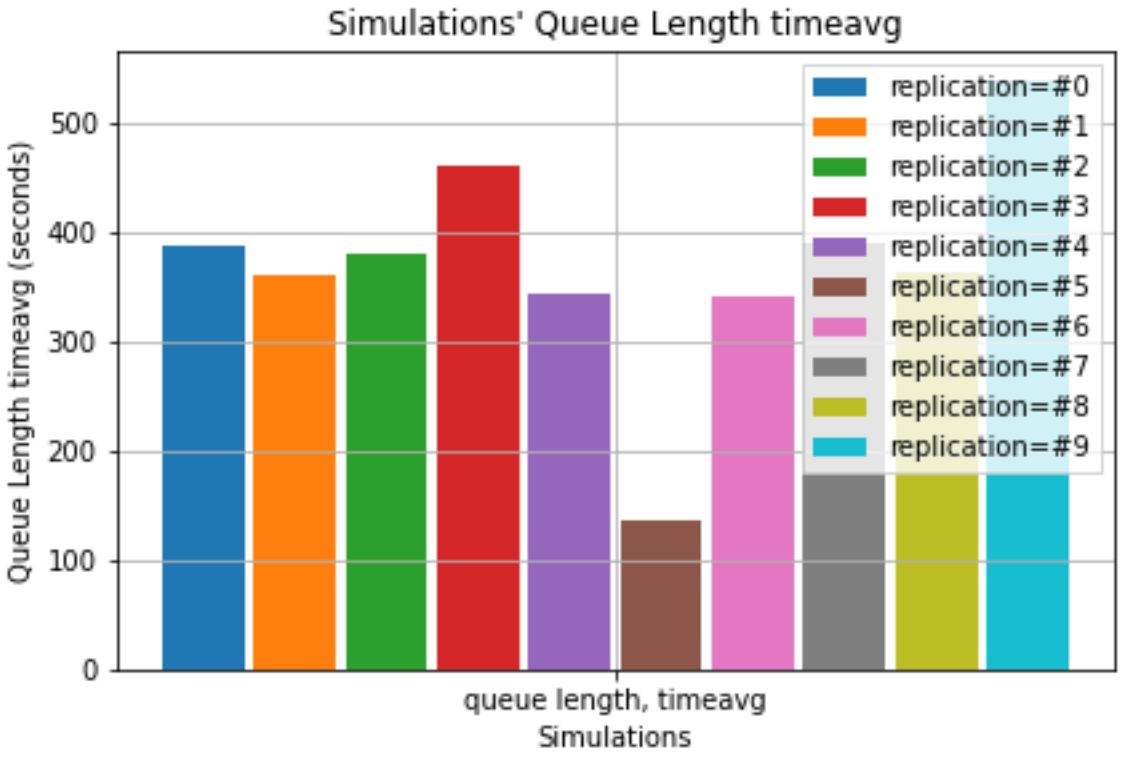

Figure 17: Queue length time-average

Figure 17 shows the time-averaged queue length for each run, indicating the average number of packets in the queue over time. The bar graph illustrates the variation in queue length between different simulation runs, with run `9` having the highest queue length and run `5` the lowest. The average queue length is around `370`, with most runs having values around this average.

10 Extract queuing time results to an SQLite file and analyze them using a SQL query.

The same configuration as in the previous section is used to generate the SQLite files. Since simulation run `8` is close to the average, the queuing time results are extracted from this run as an example. Both the scalar and vector output files are used for the analysis.

```
SELECT * FROM scalar WHERE scalarName =\
  'queueLength:timeavg';
```

This command is executed to retrieve the time-averaged queue length value from the `PacketQueue-#8.sca` file.

Figure 18 shows the results of the SQL query, as expected the time-averaged queue length for run `8` is about `363`.

```
SELECT * FROM vector WHERE vectorName =\
  'queuingTime:vector';
```

The same approach is used to extract information from the `PacketQueue-#8.vec` file, this time for the queuing time vector.

Figure 19 shows the results of the SQL query on the vector output file. In this case, it does not show the average queuing time, but information about the vector itself. The vector contains the queuing time values for each packet in the system and provides metadata such as the min (`0.0`), max (`491.85`) and sum (`3476632`) of the values.

11 Compute the average queuing time and verify Little's Law with time-averaged queue length. Compare results with theoretical formulas. Please include all the curves on a single plan to do comparison.

The computed average of the ten time-averaged queue lengths is `369.9` and the average of the ten average queuing times is `277.6`.

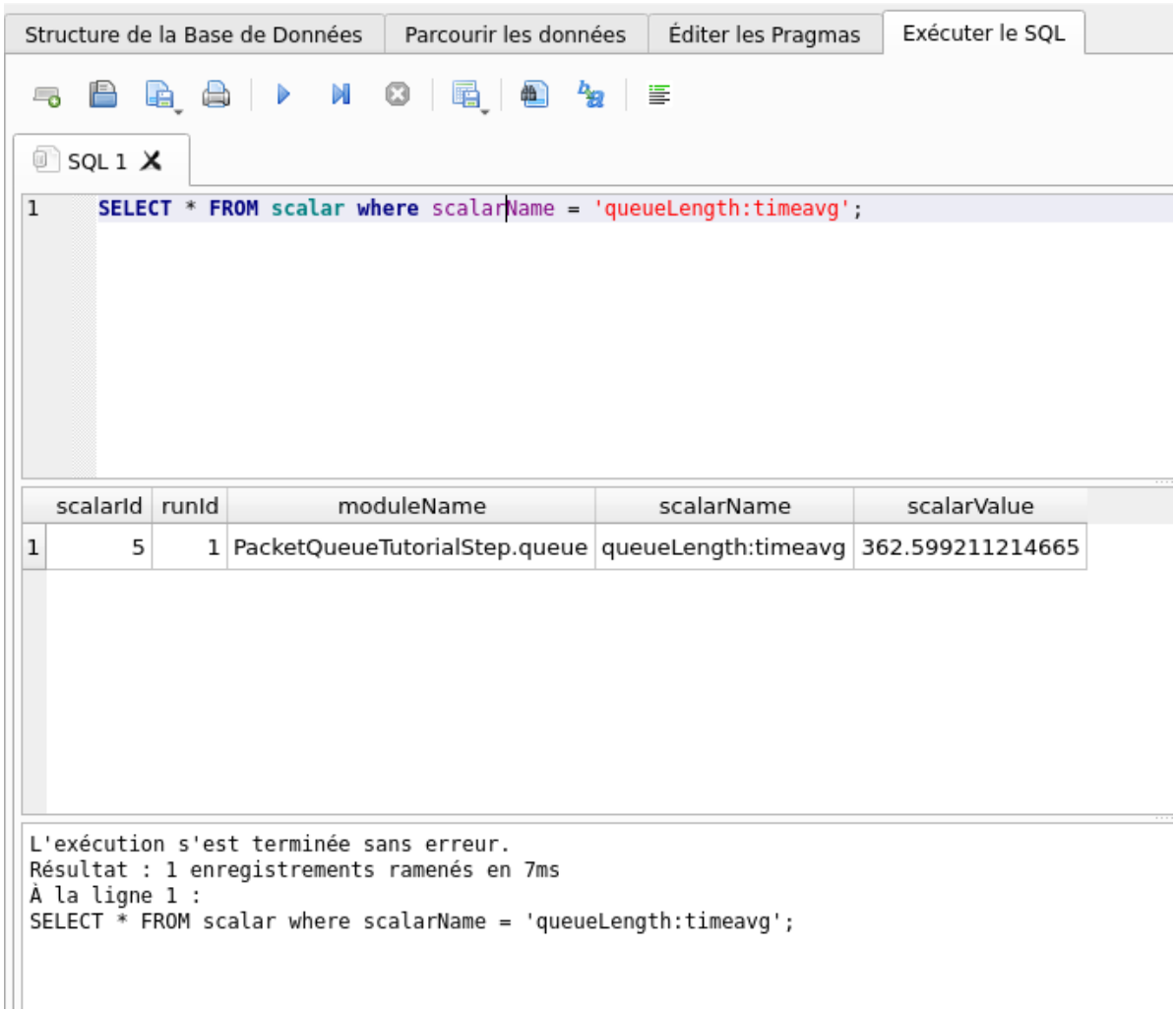

Figure 18: SQLite scalar query results

Figure 19: SQLite vector query results

$$W_q = \frac{p}{\mu(1-p)}, \text{ where } p = \frac{\lambda}{\mu} \tag{6}$$

$$N = \lambda W_q \tag{7}$$

Equations Equation 6 and Equation 7 represent Little's Law and the relationship between the time-averaged queue length and the average queuing time. By substituting the values for $\lambda = 0.75$ and $\mu = 0.79$ into the equations, the theoretical values are calculated as follows:

$$W_q = \frac{\frac{0.75}{0.79}}{0.79 \times (1 - \frac{0.75}{0.79})} = 23.734 \tag{8}$$

$$N = 0.75 \times 23.734 = 17.801 \tag{9}$$

The theoretical values are the same as those calculated in the previous sections (see Equations Equation 8 and Equation 9).

12 Comment the `PacketQueue.ned` file, explaining its statistics and signals sections.

The `PacketQueue.ned` file defines a flexible and configurable packet queue module that supports various features like packet dropping, custom sorting and queuing. Below is an explanation of the **signals** and **statistics** sections of this file.

12.1 Signals

Events triggered during packet processing:

- `packetPushStarted`: A packet begins entering the queue.
- `packetPushEnded`: A packet finishes entering the queue.
- `packetPulled`: A packet is removed for transmission.
- `packetRemoved`: A packet is removed for some other reason.
- `packetDropped`: A packet is dropped due to overflow or other conditions.

12.2 Statistics (Aggregated metrics to monitor the queue)

12.2.1 Queue state

- `queueLength`: Number of packets in the queue.
- `queueBitLength`: Total packet size in bits.

12.2.2 Packet timing

- `queuingTime`: Time packets spend in the queue.

12.2.3 Incoming packets

- `incomingPackets`: Count of pushed packets.
- `incomingDataRate`: Data rate of incoming packets.

12.2.4 Outgoing packets

- `outgoingPackets`: Count of pulled packets.
- `outgoingDataRate`: Data rate of outgoing packets.

12.2.5 Dropped packets

- `droppedPacketsQueueOverflow`: Count of packets dropped due to overflow.

12.2.6 Flow-specific

- `flowqueuingTime`: Time per flow in the queue.
- `flowIncomingDataRate` & `flowOutgoingDataRate`: Data rates per flow.

13 Comment on the source files involved in simulating the M/M/1 queue, detailing their functions and instructions.

13.1 `omnet.ini`

The `omnet.ini` file defines the simulation configuration for the M/M/1 queue. The network is set to `PacketQueueTutorialStep` and the simulation time is limited to `10,000s`. This ensures that the simulation runs long enough to collect meaningful results, while maintaining a finite time frame for analysis. The producer generates packets with an average inter-arrival time of `0.75` seconds, defined as an exponential distribution using the line:

```
*.producer.productionInterval =\
  exponential(0.75s)
```

Similarly, the server processes packets with an average service time of `0.79` seconds, also modeled as an exponential distribution:

```
*.collector.collectionInterval =\
  exponential(0.79s)
```

These settings ensure that the simulation adheres to the M/M/1 queue characteristics, where arrival and service times follow exponential distributions. Reproducibility is ensured by specifying multiple simulation runs using the `repeat` and `seed-set` parameters. The `repeat = 10` configuration allows the simulation to be run ten times, each time with a unique random seed determined by `${runnumber}`. This ensures statistical reliability by averaging the results over multiple runs.

Finally, the output configuration allows scalar and vector output in SQLite format, allowing efficient storage and analysis of simulation results. The settings are:

```
**.scalar-recording = true
**.vector-recording = true
outputvectormanager-class =\
  "omnetpp::envir::SqliteOutputVectorManager"
outputscalarmanager-class =\
  "omnetpp::envir::SqliteOutputScalarManager"
```

This ensures that both scalar (e.g., averages) and vector (e.g., time series) data are fully recorded for post-simulation analysis.

13.2 `PacketQueue.cc`

The `pushPacket()` function handles adding packets to the queue. It starts by claiming ownership of the packet with `take()`. A `CreationTimeTag` is added to record the packet's arrival time, which is important for calculating metrics like queuing time. The function then emits a `packetPushStartedSignal` and pushes the packet into the internal queue. If the queue exceeds capacity and a dropper function is configured, packets are removed until the queue is within limits. Finally, the function notifies the consumer when packets are ready for processing, emits a `packetPushEndedSignal` and updates the queue display.

The `pullPacket()` function manages the removal of packets from the queue. It retrieves the first packet (FIFO), logs its information and, if available, accesses its `CreationTimeTag` to calculate how long it has been in the queue. It then removes the packet and adds a `queuingTimeTag` to track its queuing duration. The function emits a `packetPulledSignal`, updates the display and optionally animates the removal of packets.

Together, these functions ensure efficient packet handling and accurate tracking of queuing metrics in the M/M/1 simulation.

14 Conclusion

This study has successfully demonstrated the simulation and analysis of an M/M/1 queue using the INET framework and OMNeT++ simulator. By configuring the mean arrival and service rates with exponential probability distributions, the simulation replicated realistic queuing behavior and provided insight into key performance metrics such as queuing time, queue length and system utilization.

The results demonstrated the value of combining theoretical models with simulation-based approaches. The comparison between theoretical values, derived using well-established queuing theory formulas and simulation results revealed deviations. These deviations were attributed to the finite runtime of the simulation and inherent randomness in packet arrivals and service processes. However, the results confirmed the applicability of Little's Law and validate the relationship between queue length, arrival rate and queuing time in the simulated environment.

The source code modifications enhanced the simulation's analytical capabilities by logging arrival times, which facilitated detailed performance evaluations. The SQLite-based output format and subsequent analysis using SQL queries highlighted the data extraction process and allowed for efficient visualization and interpretation of the results.

This project demonstrated the effectiveness of OMNeT++ and INET for modeling and analyzing queuing systems. Future work can extend this study to explore more complex queuing scenarios, such as priority-based queues or networks of queues. The methods and insights gained in this study provide a strong foundation for understanding and optimizing queuing

systems in diverse domains, including telecommunications, networking and resource management.

15 References

INET Framework. n.d.a. "INET Framework." Accessed January 14, 2025. https://inet.omnetpp.org.

———. n.d.b. "INET Framework - Queueing Tutorial." Accessed January 14, 2025. https://inet.omnetpp.org/docs/tutorials/queueing/doc/index.html.

Little, John D. C., and Stephen C. Graves. 2008. "Little's Law." In *Building Intuition*, edited by Dilip Chhajed and Timothy J. Lowe, 81–100. International Series in Operations Research & Management Science. Springer. https://doi.org/10.1007/978-0-387-73699-0.

OMNeT++. n.d.a. "OMNeT++." Accessed January 14, 2025. https://omnetpp.org.

———. n.d.b. "OMNeT++ Environment." Accessed January 14, 2025. https://omnetpp.org/opp_env.