Name: Yubo Wang

Email: yzw5825@psu.edu

Github repo: https://github.com/wap12358/CSE517-sim

Question 1: How to generalize your simulation to more arbitrary queueing networks. In particular, how to simulate more general renewal arrivals (easy) and how to simulate a Jackson network or a network with fixed paths between end-points, including a nice way for the user to specify: the network topology and paths, the inter-arrival time and service time distributions as chosen from the broader family of exponential phase-type, and the performance measures.

Answer:

Device Class for Nodes: I created a Device class, which serves as a model for nodes within the network. For each type of device in the network, we can define a subclass, where each subclass represents the specific functionality of that node type—such as forwarding rules, packet processing logic, and so on. In testing, these subclasses are instantiated to form models, and each device instance is connected to form a network. When each device operates, it generates events, such as processing a packet or forwarding it to the next node. These events are placed in an event stack and executed in chronological order. This structure allows for the simulation of arbitrary network topologies.

Source Device for Packet Generation: I included a source device that acts as the origin for packet arrivals. This device generates a series of packets and sends them into the network based on specified settings. By adjusting the logic of this source device, we can easily simulate various types of arrivals.

Customizable Inter-Arrival and Service Time Distributions: Both the inter-arrival and service time distributions are fully customizable as parameters. This flexibility enables the use of any distribution within the exponential phase-type family, allowing the simulation to reflect various queuing behaviors and performance measures.

Question 2: When the performance measures are mean network sojourn times, explain a role (if any) for Little's formula.

Answer:

Little's formula provides a straightforward method for cross-checking the consistency of simulation results, as any deviation from Little's formula might indicate potential inaccuracies in arrival rate, sojourn time measurements, or model assumptions.

In Little's formula, $L = \lambda W$, where L represents the expected number of packets in the system, λ is the packet arrival rate, and W is the average sojourn time of packets. In the final three figures of this report, λ is fixed at 10,000. The trend shows that L and W vary almost identically, demonstrating a proportional relationship, which is consistent with Little's Law.

However, in my implementation, I directly record the sojourn times for all packets and simply calculate their mean at the end. Therefore, Little's formula is not explicitly used in my code.

Question 3: Finally, one may partition the network and a have different event stack for each partition element. In this case, a stack X may receive an event from another stack that precedes events in stack X. Describe what needs to be done (if anything) and how that would change the operation of the stack - hint: rollback.

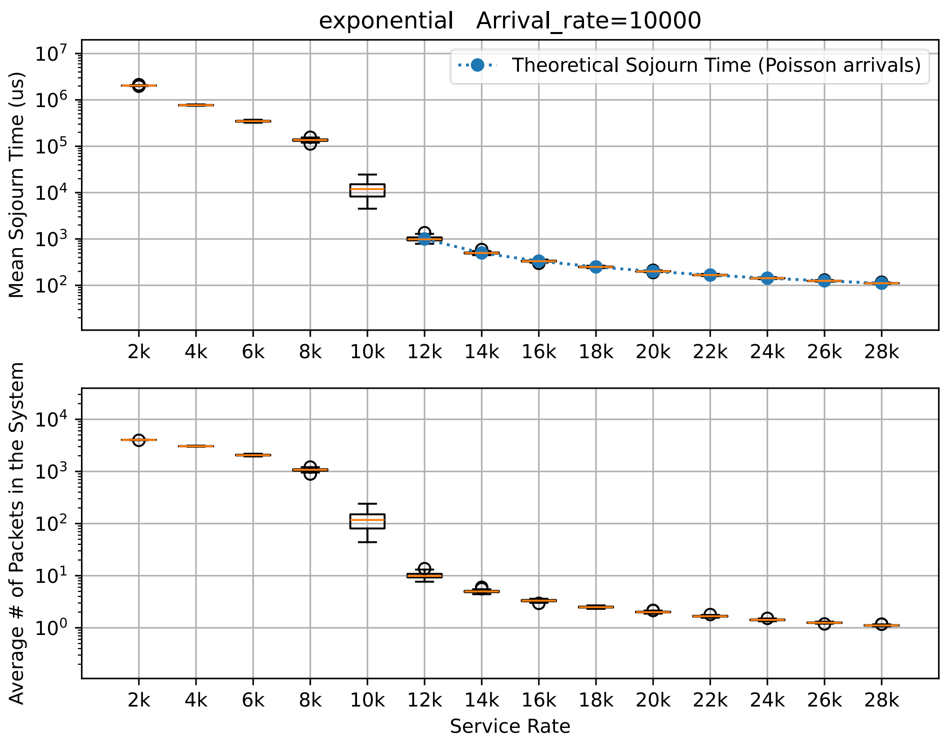
Answer:

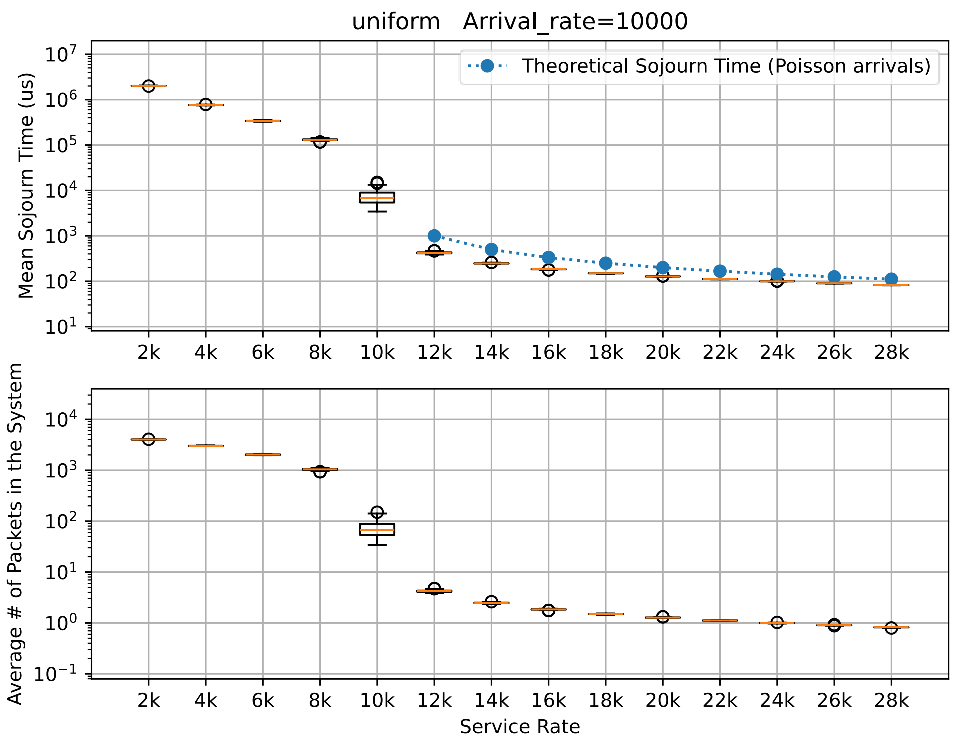
State Recording for Rollback: Each event in the queue should record the system state (differentially), enabling rollback if necessary. When an event is executed, the event enters a "pending commit" state until it is approved for finalization.

Commit Control via Centralized Synchronization Controller: A centralized synchronization controller manages the commit process. This controller tracks the latest event times in each stack and identifies the timestamp of the slowest-progressing stack. Only events occurring before this timestamp are allowed to commit across the network. This ensures that all stacks progress in sync, without premature commits.

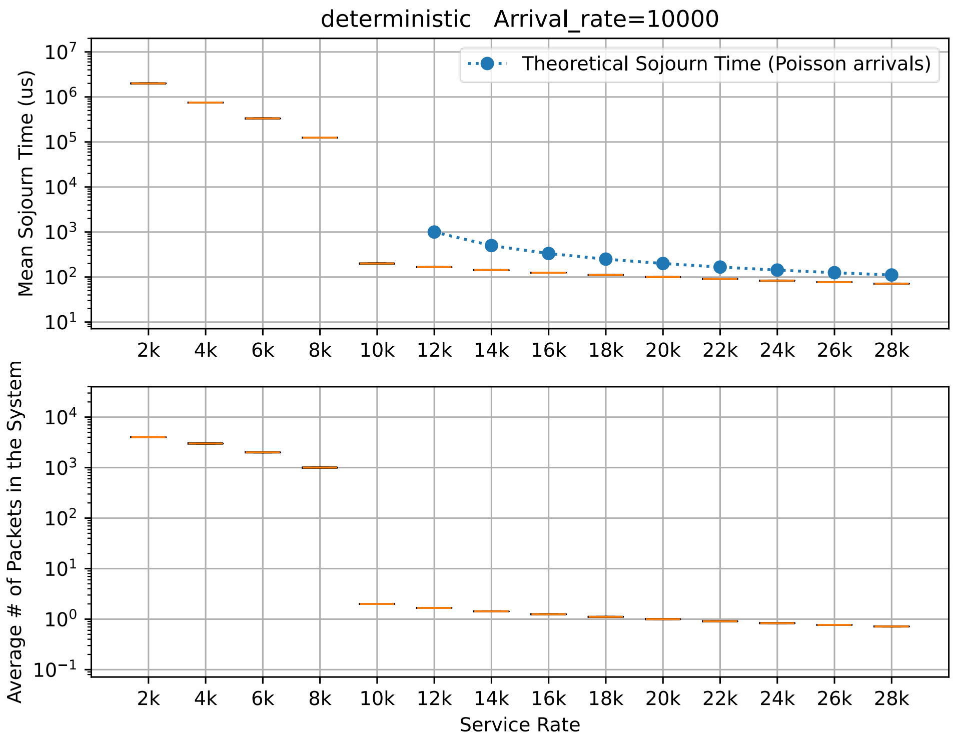
Handling Cross-Stack Events: When an event from one stack affects another stack and is inserted with a timestamp earlier than the latter stack's current progress, a rollback is triggered. This reverts the stack's progress to a point where it can correctly process the earlier event and maintain chronological order.

Results for different arrival types:





uniform   Arrival_rate=10000
Theoretical Sojourn Time (Poisson arrivals)
Mean Sojourn Time (us)
Average # of Packets in the System
Service Rate
2k
4k
6k
8k
10k
12k
14k
16k
18k
20k
22k
24k
26k
28k



deterministic   Arrival_rate=10000
Theoretical Sojourn Time (Poisson arrivals)
Mean Sojourn Time (us)
Average # of Packets in the System
Service Rate
2k
4k
6k
8k
10k
12k
14k
16k
18k
20k
22k
24k
26k
28k