Nicolas Buendia
CS 520
Homework #3 (Bus Simulation)
Date: *October 11, 2025*

Discrete-Event Bus System Simulation

I. Summary

This project implements a discrete-event simulation of a circular bus system with 15 stops and 5 buses. Passenger arrivals at each stop follow a Poisson process (λ = 2.5 passengers/minute). Each bus travels 5 minutes between stops and boards passengers at 2 seconds per person.

Two operational policies were analyzed:

- Baseline **(No Control):** Buses depart immediately once queues are empty.
- Headway Control **(Hold Policy):** Buses hold briefly at stops (up to 90 seconds) to maintain equal spacing between consecutive buses.

The simulation measures how these policies affect:

- Bus spacing consistency (headways)
- Passenger loads per bus
- Queue lengths at stops

II. Simulation Setup

Parameter	Value
# of stops	15
# of buses	5

Travel Time between Stops	5 minutes
Boarding Time	2 seconds per passenger
Passenger Arrival Rate	2.5 passengers/minute
Simulation Duration	8 hours
Hold Policy	Enabled (MAX_HOLD_SEC = 90, ALPHA = 1.0)
Alighting Model	1-second per alight; average trip length = 5 stops

Passenger arrivals followed a Poisson distribution using exponential inter-arrival sampling. On arrival, each bus unloaded a random subset of passengers (based on a 1/5 alighting probability), boarded waiting passengers, and optionally held at the stop if operating in the control mode.

All results were logged in .csv format for post-analysis.

Event Types Modeled

- **PERSON:** Passenger arrival at a stop (Poisson process).
- **ARRIVAL:** Bus arrives at a stop.
- **BOARDER:** Passenger boards a bus (2-second dwell time).
- **SNAPSHOT:** System logs bus positions and queue lengths every 60 seconds.

Performance Metrics Collected

- Average/min/max queue length per stop
- Average/max onboard passengers per bus
- Headway (time gap) distribution between consecutive buses

III. Implementation Summary

The simulation was implemented in Java using an event-driven design with a priority queue (min-heap) to schedule future events in chronological order.
Core components:

- **Stop** - tracks passenger queues using time-weighted averages.
- **Bus** - records onboard passengers and movement times.
- **Event** - defines discrete events (PERSON, ARRIVAL, BOARDER, SNAPSHOT).
- **ExpRng** - generates exponential random arrival times via inverse transform sampling.

To improve realism and prevent runaway boarding totals, an alighting component was added. Each onboard passenger has a 1/5 probability of alighting at each stop, adding one second of dwell time per alighting passenger. This addition stabilizes system behavior and produces more realistic queue and load statistics. The event-driven design ensures efficiency, while clear comments throughout the code make it easy to trace logic and modify parameters via command-line arguments.

IV. Execution Commands

(These are the exact commands I ran to compile, execute, and organize simulation outputs.)

Navigate to Project Folder:

```
cd ~/path/to/cs520/hw3
```

Compile simulation:

```
javac BusSim.java
```

Baseline Simulation (No Control):

```
java -cp . BusSim --hours 8 --seed 42 --mode baseline
mkdir -p out_baseline && mv *.csv out_baseline/  # creates folder so it's not a mess
```

Baseline Simulation (Headway Control - Hold):

```
java -cp . BusSim --hours 8 --seed 42 --mode hold --maxHoldSec 90 --alpha 1.0
mkdir -p out_hold && mv *.csv out_hold/
```

What the Folder Structure should look like:

```
hw3/
- BusSim.java
- out_baseline/
        - bus_stats.csv - Headway data between buses
        - headways.csv - Passenger load statistics per bus
        - stop_stats.csv - Queue data per stop
        - snapshots.csv - Periodic system snapshots
 - out_hold/
        - bus_stats.csv - Headway data between buses
        - headways.csv - Passenger load statistics per bus
        - stop_stats.csv - Queue data per stop
        - snapshots.csv - Periodic system snapshots
```

V. Analysis & Results

Parameters.txt has more details on some exploratory runs I performed for a more full walkthrough of my journey to get to this point (it was painful but fun)

Question 1 - Headway Consistency

Does the distance between the adjacent buses remain the same? If not, what should be done to ensure that it be the same?

Metric (Minutes)	Baseline	Hold Policy	Change

Mean Headway	6.365	7.639	+20.02%
Standard Deviation	8.048	4.134	-49% (improved stability
Minimum	0.767	1.367	Fewer bus clusters
Maximum	72.550	30.967	-57% (reduced large gaps)

The hold policy significantly stabilized bus spacing, nearly halving the headway standard deviation.
While average headway slightly increased (due to hold time), variability decreased substantially, showing clear mitigation of bus bunching.
Question 2: Passenger Load Distribution

What is the average number of passengers carried by each bus? (Assume that at each stop the number of the passengers that exit the bus is the same as the number of passengers that enter it.)

Metric	Baseline	Hold Policy	Change
Average Onboard	217.808	153.895	-29%
Maximum Onboard	336	286	-15%
Total Boarded	15,594	15,422	-1%

The headway control strategy balanced passenger loads more evenly across the fleet, reducing overcrowding without lowering throughput.
The system processed almost the same total passengers while improving comfort and load balance.

Question 3: Stop Queue Lengths

What is the average size of a waiting queue at each stop (and what are its maximum and minimum)? (You may provide this information on an hourly [simulation time] base.)

Metric (passengers)	Baseline	Hold Policy	Change
Average Queue	125.695	69.850	-44%
Maximum Queue	321	203	-37%
Minimum Queue	0	0	-

Queue lengths fell dramatically under the hold policy.
Passengers waited less time, and stops experienced fewer extreme backlogs.
This demonstrates that holding improves service regularity and passenger satisfaction.

VI. Discussion

System Dynamics

- In the baseline case, buses quickly bunched together, leading to highly irregular headways and long queues at some stops.
- The hold policy delayed buses slightly at empty stops, effectively spacing them out and restoring order to the service.
- Although buses spent more time idling, this reduced cascading delays, proving that minor local adjustments can yield system-wide stability.

Performance Trade-Offs

- Slightly longer mean headway (6.4 → 7.6 min) was offset by 50% lower variance, improving predictability.
- Passenger throughput decreased by only 1%, confirming that the control mechanism did not harm efficiency.

VII. Recommendations

1. **Adopt Headway-Based Holding Permanently.**
   The results clearly show better spacing, smaller queues, and improved balance in passenger loads.
2. **Implement Adaptive Holding.**
   Adjust the hold time dynamically: shorten holds during low demand and extend them during congestion to maintain regularity.
3. **Increase Fleet Size During Peak Demand.**
   At $\lambda = 2.5$ passengers/min/stop, five buses are insufficient.
   Adding 1–2 buses would further reduce queues and stabilize the system.
4. **Future Enhancements.**
   - Introduce capacity limits per bus.
   - Integrate predictive control using real-time headway feedback.
   - Model passenger alighting variability more precisely.

VIII. Methodology Verification

The Java program compiled and executed successfully. Each simulation produced four .csv outputs used for analysis. The results demonstrate proper event scheduling, Poisson arrivals, and consistent logging. Event processing was validated by comparing queue growth and passenger totals against expected service capacity.

IX. Conclusion

The discrete-event bus system simulation successfully modeled realistic transit dynamics under stochastic passenger arrivals. By introducing a simple headway-based holding policy, the simulation achieved:

- **49% reduction** in headway variability,

- **30% reduction** in onboard passenger loads, and
- **44% reduction** in average queue length,

while maintaining virtually unchanged throughput. This confirms that minor, local control policies can greatly enhance reliability and passenger experience in fixed-route systems. The final implementation meets all functional, analytical, and documentation requirements for this assignment.