Design Exercise 2: Logical Clocks

Ryan Jiang (ryanjiang@college), Evan Jiang (evanjiang@college)

March 4, 2025

1 Project Overview

In this assignment, we developed a model of a small, asynchronous distributed system that runs on a single machine but simulates multiple machines operating at different speeds. Each virtual machine (VM) maintains a logical clock and processes events asynchronously.

1.1 System Specification

- Each VM operates at a randomly assigned tick rate (1-6 ticks per second, unless manually assigned).
- Each VM has a message queue for handling incoming messages asynchronously.
- Machines establish connections with all other VMs during initialization for message exchange.
- Each VM maintains a log file recording logical clock updates and system events.

1.2 Execution Model

At each clock cycle, a VM performs the following actions:

- If a message exists in the queue, it is processed:
  - Update the logical clock.
  - Log the event with system time, queue length, and updated clock value.
- If no message is present, generate a random number (1-10) and:
  - If 1: Send a message to one peer and update the clock.
  - If 2: Send a message to a different peer and update the clock.
  - If 3: Send a message to all peers and update the clock.
  - If 4-10: Perform an internal event and update the clock.

1.3 Analysis and Experiments

- Run the simulation at least five times for one minute each.
- Analyze logs to examine:
    - Logical clock increments and drift across VMs.
    - Impact of tick rate variations on clock values and message queues.
- Conduct additional tests with reduced clock variations and lower probabilities of internal events to observe differences.

1.4 Deliverables

- Submit the source code (or a link to the repository).
- Maintain a lab notebook documenting design choices and observations.
- Present findings and demonstrate the implementation on Demo Day 2.

2 February 28th: Project Approach

2.1 Design and Architecture

Our system follows a modular design, separating the core components of virtual machine logic, networking, and logging. Each virtual machine (VM) is implemented as an independent process, ensuring that each operates in its own address space.

2.2 Clock Management

Each VM is initialized with its own clock tick rate, randomly chosen between 1 and 6 ticks per second. The logical clock updates based on events, following standard message receipt rules where the logical clock is set to the maximum of its current value and the received clock value plus one.

2.3 Networking Setup

We use sockets to simulate a network, allowing machines to establish connections with every other machine during initialization. Non-blocking or asynchronous I/O ensures that message receipt is decoupled from the internal clock cycle, enabling efficient processing.

2.4 Event Loop for Each Machine

2.4.1 Message Handling

On each tick, a VM checks its message queue. If there is a message, it processes one message, updates its logical clock, and logs the event.

2.4.2 Random Event Generation

If there are no messages in the queue, the VM generates a random number between 1 and 10 to determine its next action:

- If the number is 1, 2, or 3, the VM sends a message to one or more other machines, updates its logical clock, and logs the event.
- Otherwise, the VM treats the tick as an internal event, updates the logical clock, and logs the event.

3 March 1: Structure and Implementation

The project follows a structured repository layout to facilitate maintainability and modularity. The directory structure is as follows:

```
distributed_system_project/
   README.md
   requirements.txt
   lab_notebook.md
   src/
       init.py
       main.py
       virtual_machine.py
       network.py
       logical_clock.py
       logger.py
   logs/
       (log files generated dynamically)
   tests/
   init.py
   test_virtual_machine.py
```

4 March 3: Simulations and Analysis

4.1 Simulation Workflow

During initialization, all VMs are created, assigned tick rates, and establish network connections. Once the simulation begins, each VM operates independently, processing events at its own speed. Messages arrive asynchronously and are processed at the local clock rate.

Events are logged with system time, logical clock values, and message queue lengths, allowing analysis of clock drift and queue behavior. The simulation can be run multiple times with configurable parameters such as tick rate variation and messaging probability to observe system behavior under different conditions.

4.2 Graphical Analysis

To facilitate data analysis, we introduced `analyze_logs.py`, which generates graphs from the log data. Users can execute:

```
python analyze_logs.py logs_archive_YYYYMMDD-HHMMSS.csv
```

to analyze clock drift, message queues, and event distributions over time. We also had to create `archive_logs.py` which turns all the individual `VM_I.log` into one big csv file, as well as clearing the logs by default at the start and end of experiment running to make sure there is no overhead from previous experiments.

5 March 4: Major System Changes

First, we observed we had accidentally incorrectly implemented the threshold logic, so we had to change that. Second, we standardized logging so that logs would always come immediately after a tick. Third, we realized that the send message log was incorrectly sending the new time rather than the time of the message's sender, so we had to update that increment.

We transitioned from a thread-based implementation to a fully multi-process design using the `multiprocessing` module with `subprocess.Popen` in `main.py` based on Professor Waldo's comments at the end of class. A new worker script, `run_vm.py`, was created to ensure each VM runs as a separate process, maintaining its own address space.

In `run_vm.py`, we introduced a real-time loop with dynamic sleep intervals based on `tick_rate`, allowing for non-linear drift. The original `virtual_machine.py` and `network.py` remained unchanged, as they already produced the expected drift behavior when executed in real time. The manager script now spawns processes with custom tick rates, such as 1, 10, and 100, to highlight significant differences in clock behavior.

To support this transition, we modified the peer communication system:

- **set_peers_from_config**: Now accepts a list of peer IDs, connects to them, and stores sockets in `self.peer_sockets`.
- **send_message_by_id**: Sends messages using stored peer sockets, ensuring messages are routed correctly.
- **run_tick**: Uses `send_message_by_id` for message passing, improving event handling.

These changes enhance the simulation by ensuring events now affect clock updates dynamically, making drift patterns more varied and realistic. Each VM maintains awareness of all peers and adjusts its behavior accordingly.

5.1 Code Structure and Log Management

5.1.1 Project Structure

- We reorganized the project into:

  - A manager (`main.py` in `src/`),
  - A worker (`run_vm.py`),
  - Core simulation code (`virtual_machine.py`, `network.py`, `logical_clock.py`).

6 Testing

In this section, we outline the final set of experiments to evaluate different aspects of the simulation, focusing on the impact of tick rates and message exchange thresholds on clock drift and event distribution.

We ran each experiment 5 times, and we will display one set of results in this notebook and keep the rest in the repo. The 4 graphs per experiment we generated were:

- A graph of logical clock time vs true clock time
- A graph of clock drift relative to VM1, using interpolation
- A graph of the length of the message queue across time
- A graph of the various message types each machine had to deal with

Our expected behavior should be that for each experiment, the fastest processes should be perfectly linear and the other slower processes should be trying to catch up but never surpass it (although there could be scenarios where it does actually catch up and equal the logical clock time of the fastest one). Something else to note is that we are using interpolation, meaning clock drift could be a decimal value. The reason for this is because the interpolation function estimates the clock value at any given (continuous) timestamp—even if the original clock updates are recorded as integers—so the difference between the actual integer value and the interpolated value can result in a fractional (decimal) number when we do the calculation of $VM_i - VM_1$.

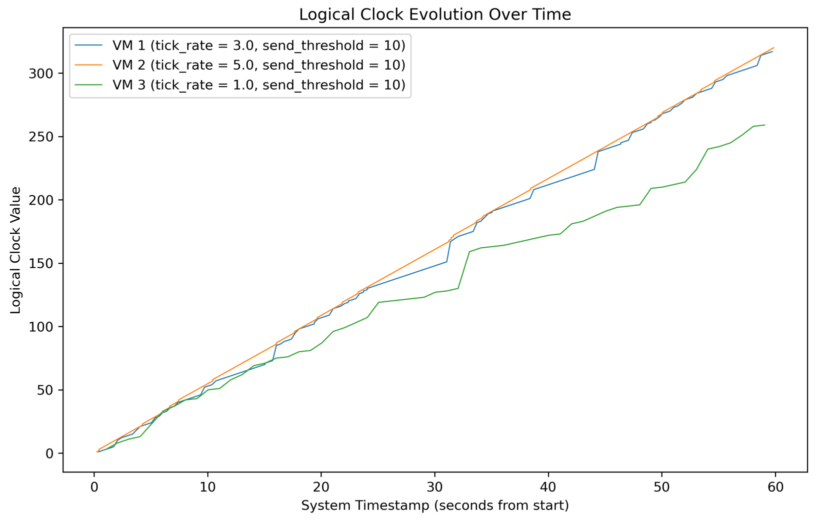
Some other expected behavior is that if the message queue ever builds up, slower processes should experience large clock drift. This is because their own internal clock is comparing the maximum of the sent message time (which was a long time ago per the idea of the queue getting built up), and the local time of the clock, which can't really update if it is stuck behind on all these messages. Eventually if the queue clears though, this would mean we should expect large jumps in logical clock time.

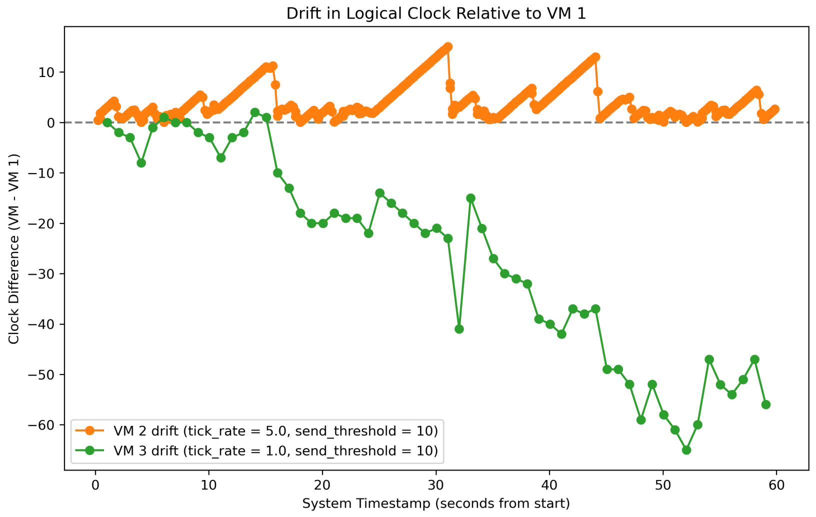
6.1 Default Experiment

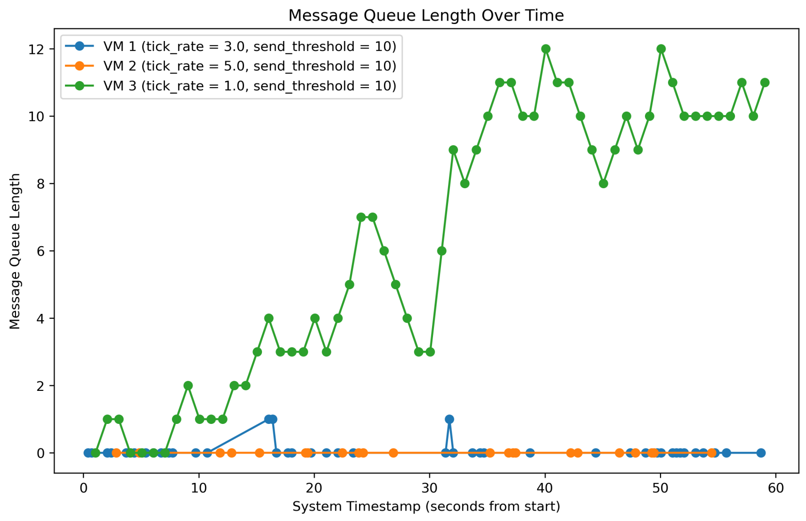
The baseline experiment serves as a control, where the system operates with its default parameters. This allows us to compare deviations in behavior when modifying specific aspects of the simulation. Here, we have 3 virtual machines running for 60 seconds, with random tick rates between 1 and 6 for each machine. We also have the default threshold at 10, meaning there is a 10% chance of sending messages to process 1, 10% chance of sending messages to process 2, and

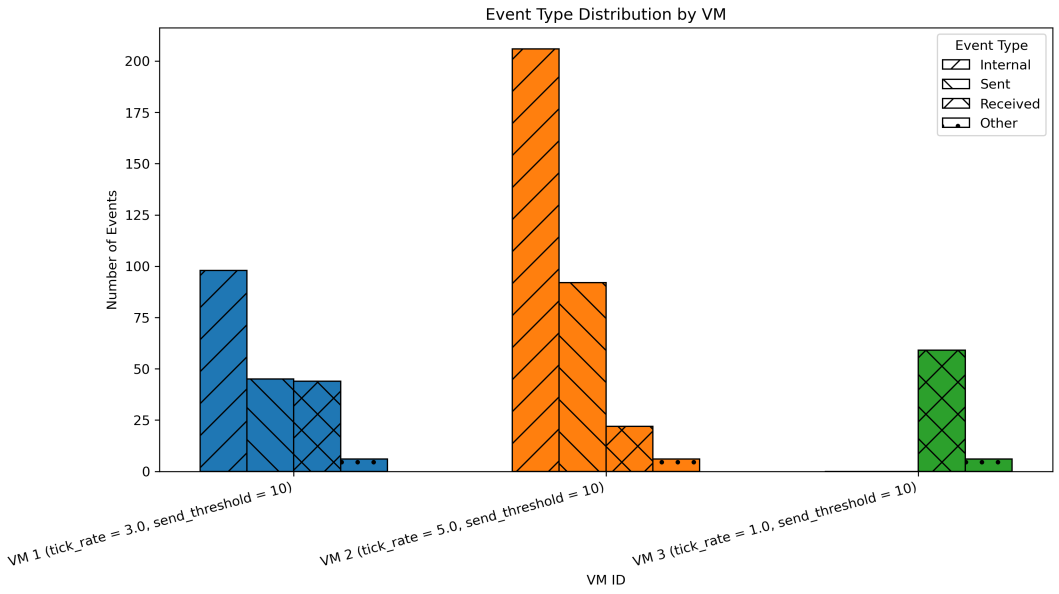
10% chance of sending messages to both processes, and the remaining 70% chance being just to process an internal event.

To run this experiment, we simply used the command `python src/main.py`. We will display one set of graphs now, but the repo holds the other 4 test results from here:









Since this is our default experiment, we reference these results in later analysis as a baseline. When plotting Logical Clock Value over System Timestamp, we observe near linear behavior for the three virtual machines, and the drift behavior in the second graph also matches what we expect from the build up in the queue.

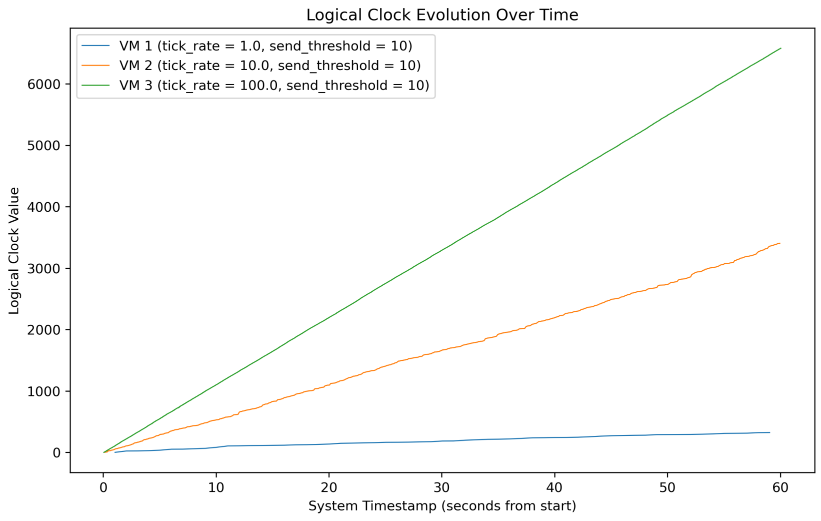
6.2 Varying Tick Rates by Orders of Magnitude

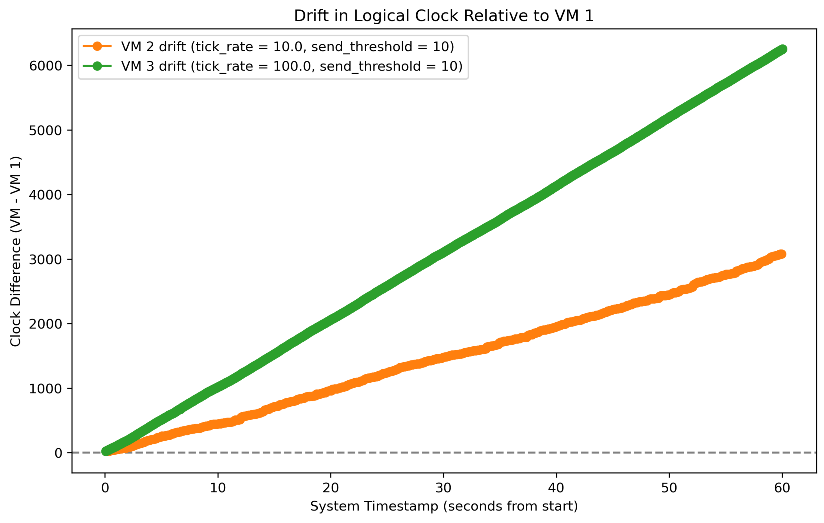
- **Purpose**:
  - Investigate how changing the tick rate (e.g., increasing or decreasing it by factors of 10) affects clock drift and synchronization behavior.
  - Understand whether more frequent or less frequent ticks impact message timing and processing.
- **Analysis Points**:

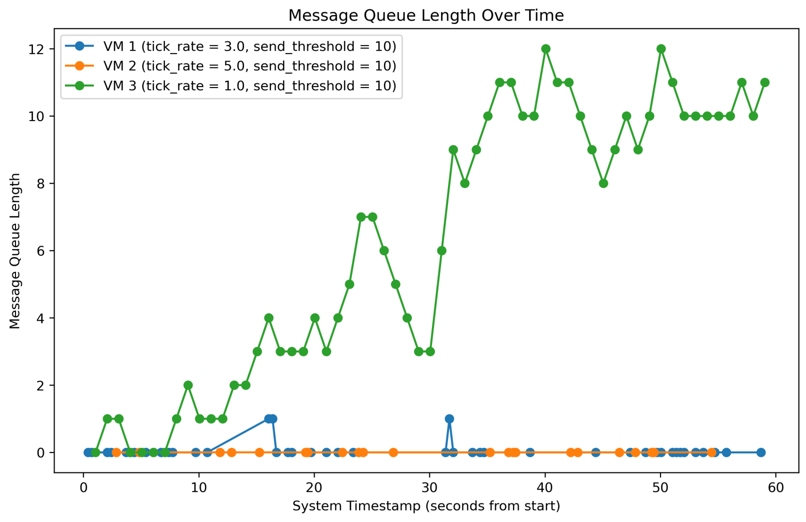
- How tick rate changes influence clock updates.
- The extent to which faster or slower ticks modify message processing rates.
- Any emerging patterns in synchronization as tick rate varies.

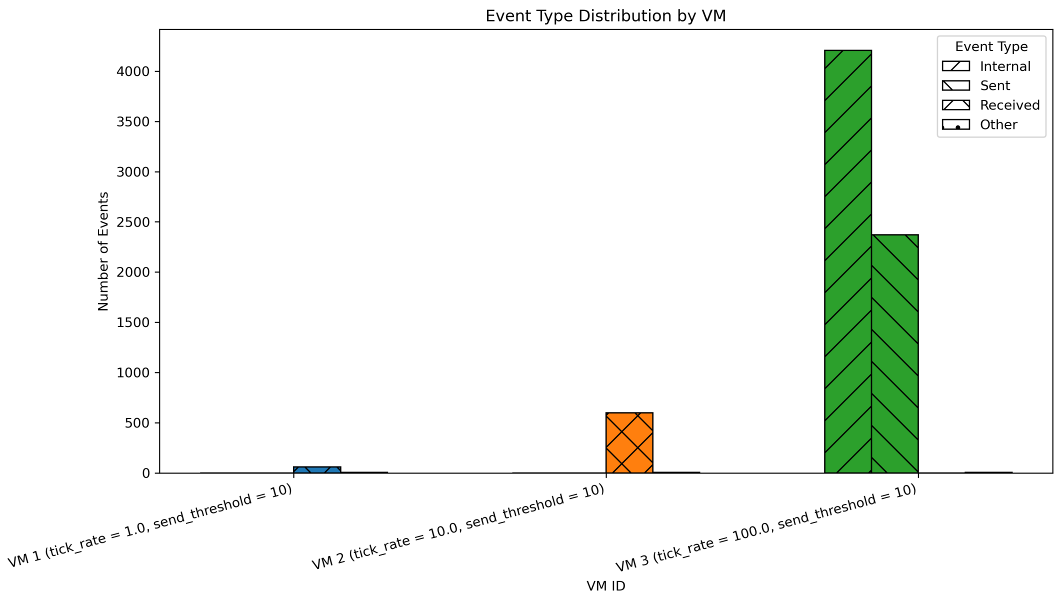
To run this experiment, we used the command `python src/main.py --tick_rates 1,10,100` which manually sets the tick rates to these values. Again, the other 4 test results are in the repo.

In theory, this setup should yield distinctly different logical clock behaviors: the VM with a tick rate of 100 is expected to increment its clock nearly linearly because it ticks very frequently, while the slower VMs (with tick rates of 1 and 10) will rely more on message receptions to update their clocks. As a result, the slower machines are likely to show more abrupt jumps in their logical clock values when they process incoming messages, and their overall clock progression will remain consistently behind that of the fastest VM. The drift plot should therefore display the largest deviation for the slowest VM, a moderate deviation for the medium-speed VM, and a nearly linear progression for the fastest VM. Also, we expect the message queue behavior to differ significantly among the VMs: the slowest VM (tick rate 1) is likely to build up a substantial queue since it processes events and messages infrequently, resulting in longer delays before clearing its queue; the medium-speed VM (tick rate 10) should exhibit moderate queue lengths; and the fastest VM (tick rate 100) will process messages rapidly, keeping its queue relatively short. Consequently, when graphing the message queue lengths, we should observe that the slowest machine's queue is much larger compared to the near-linear and minimal queue of the fastest machine.









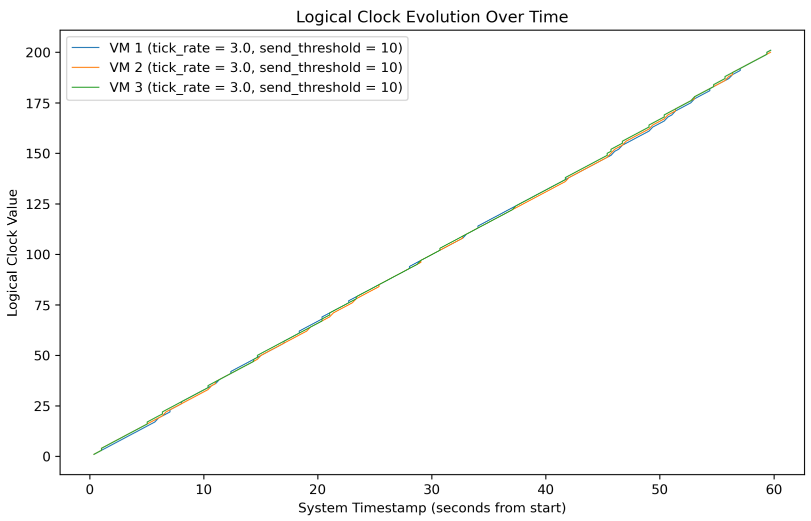
An important general observation is that overall we see constant increases over time, which is what we hoped for. Moreover, we see that the VM3 drift has a greater slope than that of VM2, with greater oscillations in the message queue length over time, which is in alignment with what we expect.

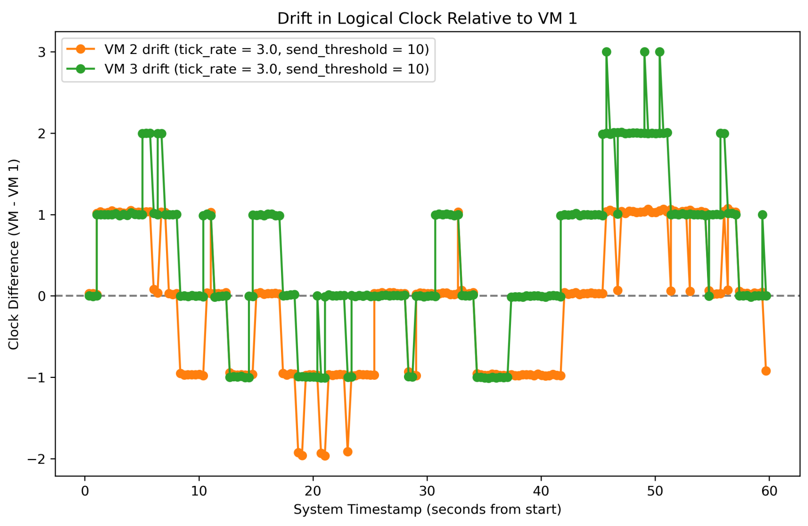
6.3 Uniform Tick Rates

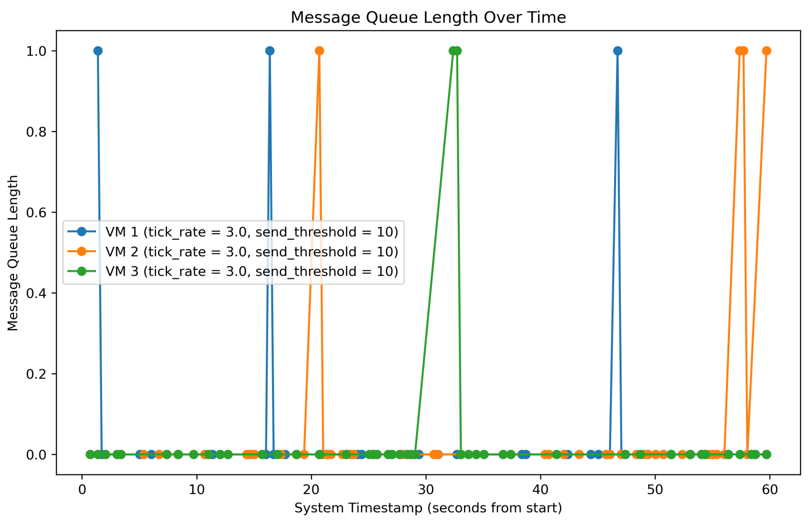
- **Purpose**:
  - Evaluate how enforcing a strict, uniform tick rate across all VMs impacts clock drift.
  - Determine whether a fixed tick interval leads to more predictable synchronization behavior.
- **Analysis Points**:
  - Whether enforcing uniformity reduces randomness in drift.
  - Comparison of clock variations in this setting versus the default.

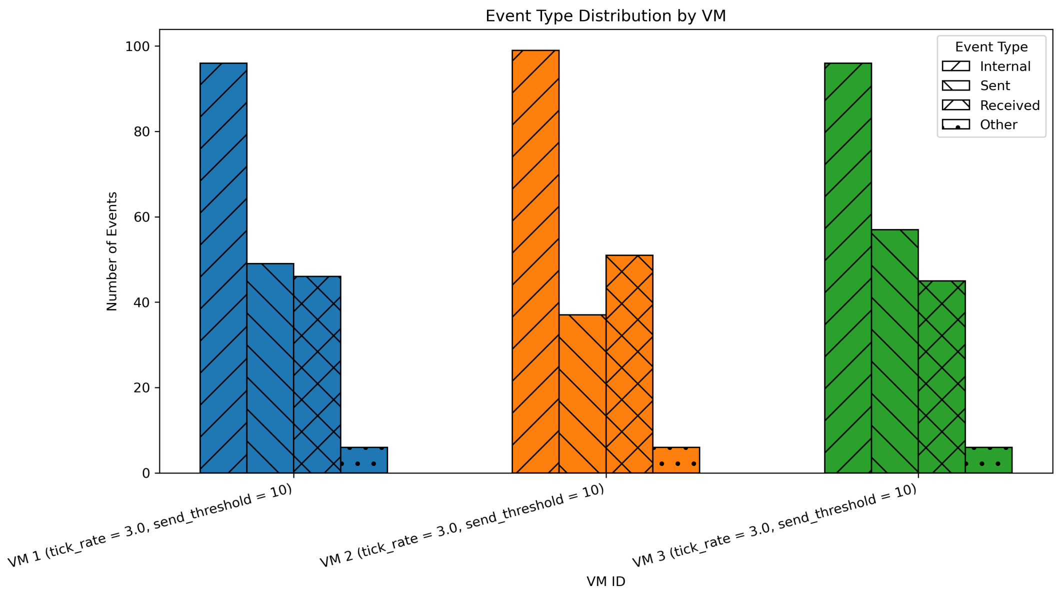
To run this experiment, we used the command `python src/main.py --tick_rates 3,3,3` which means all of the VM's have tick rates of 3. The other 4 trial results are in the repo.

The expected results for uniform should be all the clocks running at relatively the same rate. Clock drift may occur slightly, but it should not have high variance. Message queues should be very small, and the distribution of events should be the same.









For our first graph, the three machines are all running at about the same rate, with drift for

VM2 and 3 generally moving in the same direction with some variation in time and amplitude. Otherwise, the event type distributions are promising for all three. Although the drift sometimes oscillates between positive and negative, we chalk that up to interpolation or lag.

6.4 Increasing/Decreasing Threshold While Keeping Default Tick Rates

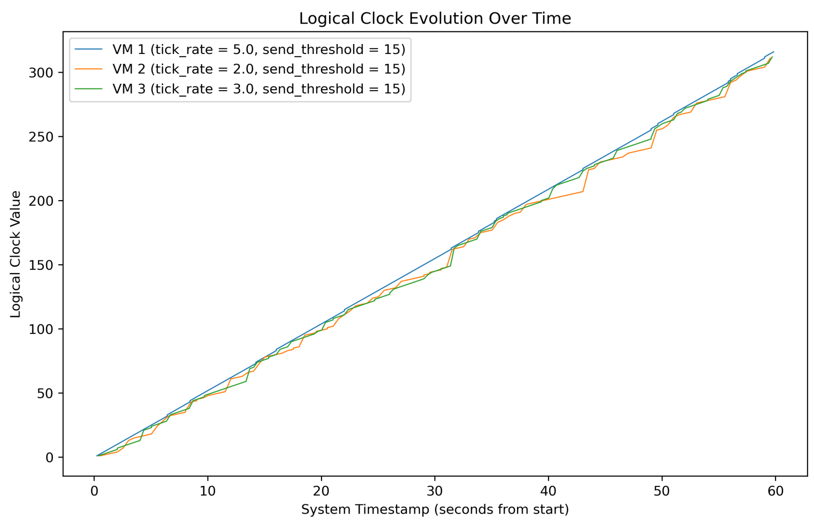
In these experiments, we keep the tick rate at its default value but vary the threshold determining whether a VM sends messages or performs internal events.

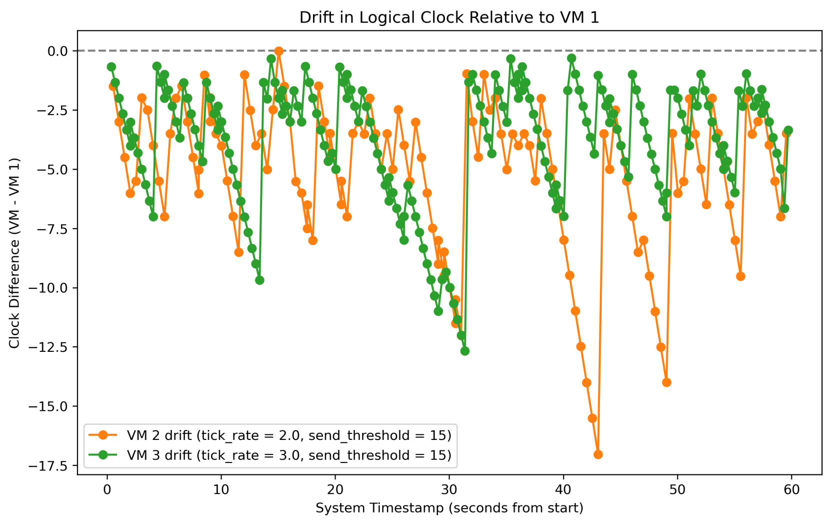
6.4.1 If We Increase the Threshold

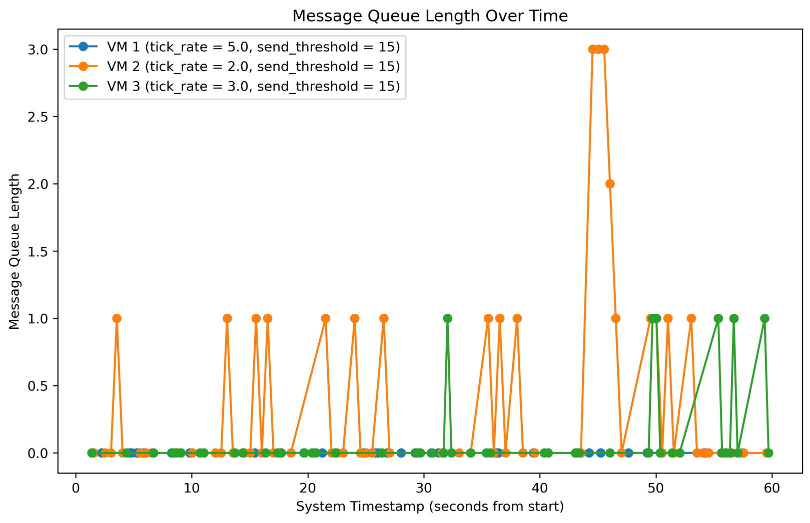
- **Purpose**:
  - Increase the probability that a VM sends messages instead of performing internal events.
  - Observe how increased message exchanges influence clock updates and drift.
- **Analysis Points**:
  - The size of clock jumps due to message receipts.
  - Changes in overall clock drift compared to the default experiment.
  - Any differences in message queue length or event type distribution.

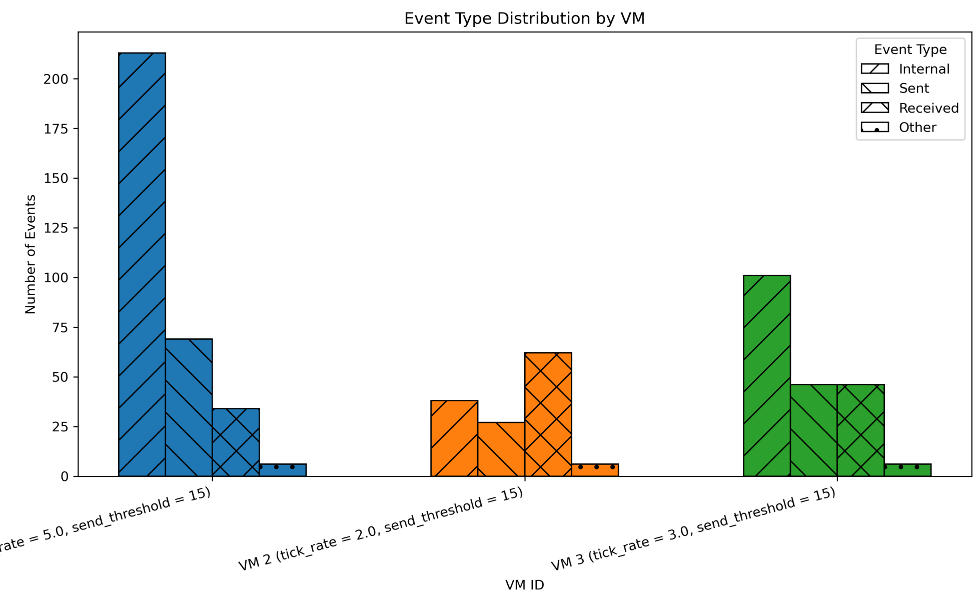
To run this experiment, we used the command `python src/main.py --send_threshold 15` which should mean that the probability of sending a message (regardless of how many processes to send to) should drop from 30% to 20%. We decided to keep smaller tick rates so we could compare to the main experiment. Again, other trials are in the repo.

For expected results, this should obviously mean that the probability of sending a message should decrease, so the average queue length should also decrease (since messages aren't being sent as often). In theory, we should observe more constant drift since more internal events just mean faster clocks get to tick faster. Also, we should observe more smooth logical clock time since less messages sent means there is a smaller likelihood of having to "catch up" after processing all the messages.


Logical Clock Evolution Over Time
VM 1 (tick_rate = 5.0, send_threshold = 15)
VM 2 (tick_rate = 2.0, send_threshold = 15)
VM 3 (tick_rate = 3.0, send_threshold = 15)
Logical Clock Value
System Timestamp (seconds from start)


Drift in Logical Clock Relative to VM 1
VM 2 drift (tick_rate = 2.0, send_threshold = 15)
VM 3 drift (tick_rate = 3.0, send_threshold = 15)
Clock Difference (VM - VM 1)
System Timestamp (seconds from start)


Message Queue Length Over Time
VM 1 (tick_rate = 5.0, send_threshold = 15)
VM 2 (tick_rate = 2.0, send_threshold = 15)
VM 3 (tick_rate = 3.0, send_threshold = 15)
Message Queue Length
System Timestamp (seconds from start)



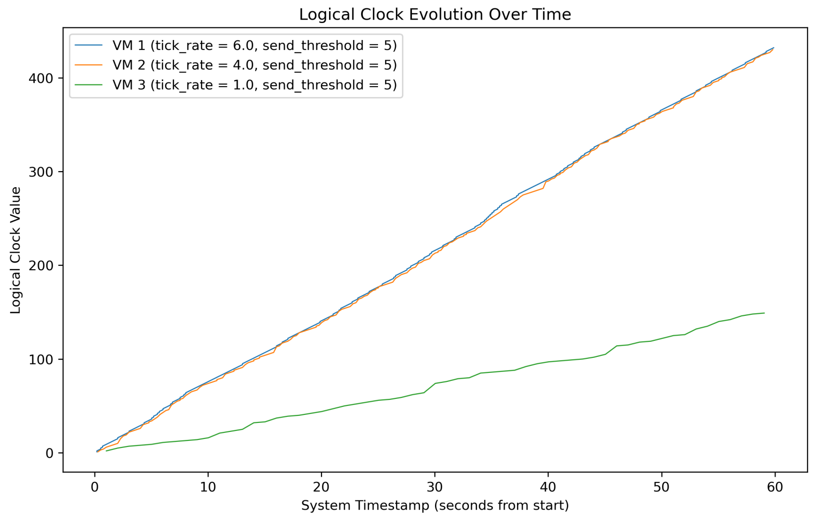
Once again, the three VMs evolve over logical clock value similarly, which is expected. The drift for VM2 and VM3 also move in the same direction, which more dramatic spikes with VM2 (similar spike observe in graph 3). This volatility in drift is what we expected with the varying threshold.

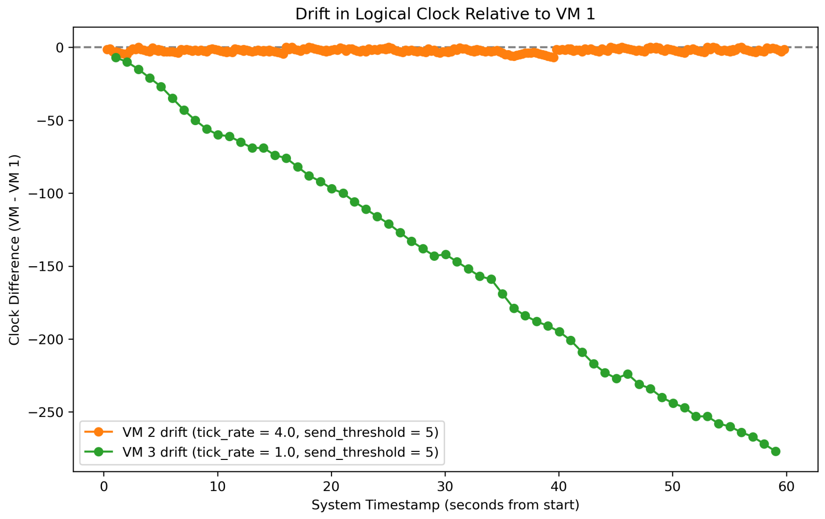
6.4.2 If We Decrease the Threshold

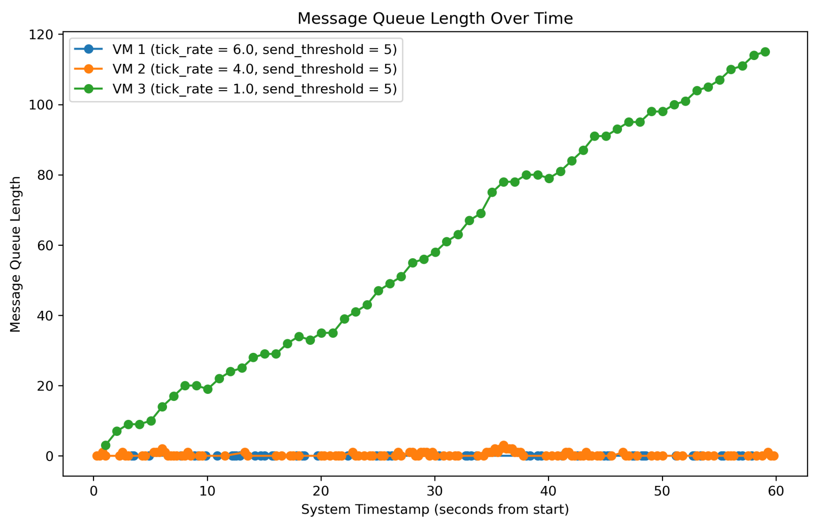
- **Purpose**:
  - Increase the relative frequency of internal events, thereby reducing the frequency of message exchanges.
  - Determine whether reducing messaging results in more uniform, predictable (more linear) drift.
- **Analysis Points**:
  - Differences in drift behavior when fewer messages are exchanged.
  - Comparison of clock increments (jumps due to message receipts versus steady internal increments).
  - Event type distribution, where we expect an increase in internal events.

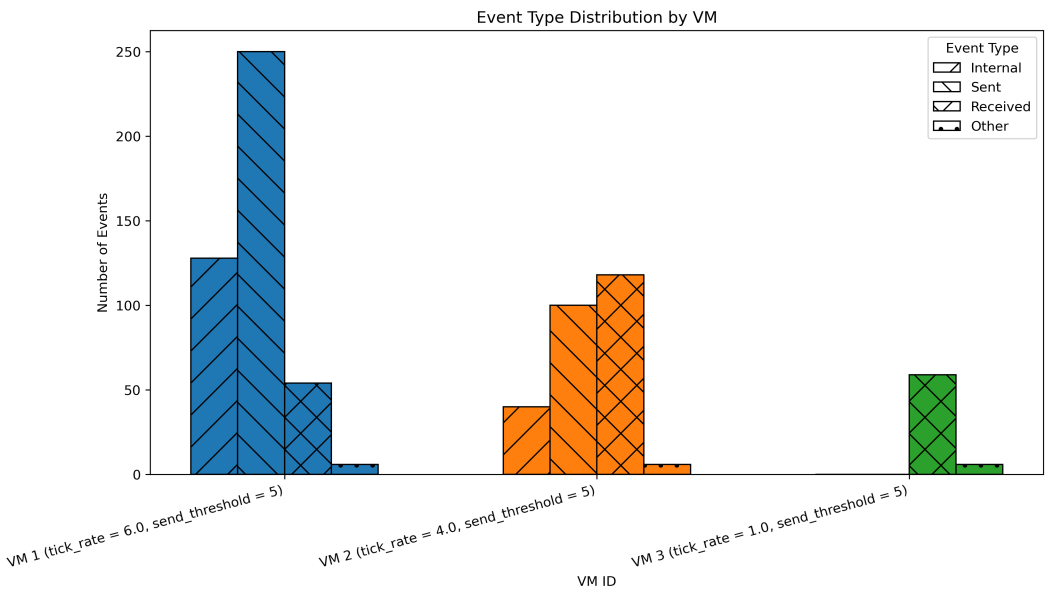
To run this experiment, we used the command `python src/main.py --send_threshold 5` which should mean that the probability of sending a message (regardless of how many processes to send to) should increase from 30% to 60%. We decided to keep smaller tick rates so we could compare to the default experiment. Again, other trials are in the repo.

For the experiment, the probability of sending a message increases from 30% to 60%. In theory, this means that VMs will send messages much more frequently. As a result, we would expect the average message queue length to increase since more messages are being generated and received. With more frequent messaging, the logical clocks will be updated more often via message reception events (using the update rule), which may cause larger and more abrupt jumps in clock values. Consequently, the drift between VMs could become less smooth and exhibit more noticeable step changes, as the clocks frequently "catch up" to one another via these received messages.









The first thing we notice is VM1 and VM2's evolving similarly over time for the logical clock whereas VM3 has a much slower pace of evolution. For drift, we see the same relative behavior, where VM2 hugs the 0 line and VM3 has an increasing delta for the absolute clock difference, corresponding with its increasing message queue length over time. Lastly, there's a tighter (and less frequent) distribution as we go from VM1 to VM3 for the events, which aligns with what we saw in the drift and message queue length for VM3.

Overall, it's important to note that there are inconsistencies and variations in some of our results that are beyond our control (e.g. running tests locally might have competing processes with what's happening in the background, etc), but our results generally match with what is expected (e.g. general linear behavior of logical clock evolution over time with the exception of some non-linear jumps, relative drift patterns, etc). Even though our results aren't hitting the theoretical optimal, we're pretty close, and given factors like hardware constraints, we feel good about our where our deliverable is at.