INTRO TO ROBOTICS JOURNAL
COMP 20170
GROUP 4:

TEAM MEMBERS:
- Mynah Bhattacharyya
- Luke Lynam
- Adam Lacey
- Jon Eckerth
- Ben McDowell

Assignment 4: ROS: Steps
- Create 2 nodes.
- The first node number_publisher will publish to a topic called /number with message type which is : std_msgs/Int64
- The second node number_counter contains both a subscriber and a publisher. It will subscribe to the topic number and it will publish to the /number_count
- update the publishers to be anonymous.
- With the commands you have learnt (rostopic, rosnode, isg_graph) and by increasing the number of publishers justify your comments on what you observe.

Tasks:	Status	Person	Date Finished
Download Virtual Box	Complete	Everyone	29/03
Install Ubuntu 20.04	Complete	Everyone	29/03
Install ROS	Complete	Everyone	30/03
Create first python & C++ node	Complete	Everyone	30/03
Create Subscriber with topic /number in ROS 2 python	Complete	Jon	06/04
Create Subscriber + Publisher with topic /number_count in ROS 2 python	Complete	Jon	06/04
Create Subscriber in ROS 2 python	Complete	Jon	06/04
Create Subscriber with topic /number in ROS python	Complete	Adam	09/04
Create Subscriber + Publisher with topic /number_count in ROS python	Complete	Adam	09/04
Create Subscriber in ROS python	Complete	Adam	09/04
Update the publishers to be anonymous	Complete	Everyone	06/04
Shoot Video	Complete	Adam	27/04
Journal	Complete	Jon	27/04

DAILY LOG 1:
Jon, Wed 29/3
Today, we started setting up the virtual environment for the project. It turned out to be more challenging than we anticipated. We encountered several issues, including limited storage on one of the laptops and incompatibility between Linux VM and ARM-based Macs. To resolve these issues, we researched solutions online, and in some cases, had to seek help from the . After working through these problems, we managed to have two PCs running Linux, one with Ubuntu 22 and one with Ubuntu 20.04.

Total work time - 4 hrs

DAILY LOG 2:
Jon, Thurs 30/3
We began setting up the ROS environment today. We discovered that ROS1 only works on Ubuntu 20, but ROS2 works on Ubuntu 22 and provides a legacy compatibility mode, allowing ROS1 and ROS2 to interact. The setup process was quite involved and took up most of our time. We had to ensure that we properly installed all the required packages and dependencies, and we also had to configure our systems for multiple users and understand the differences between ROS1 and ROS2.

Total work time - 4 hrs

DAILY LOG 3:
Jon, Wed 6/4
After finally getting ROS2 to work, we programmed the Subscriber, Publisher, counter, and the different topics. However, we realized that although ROS2 can interact with ROS1, the syntax is significantly different. The environment structure also differs greatly. We had to spend a considerable amount of time understanding the new syntax and adapting our code accordingly. We also had to refactor our code to ensure that it was modular and easy to maintain, as we anticipated further changes down the line.

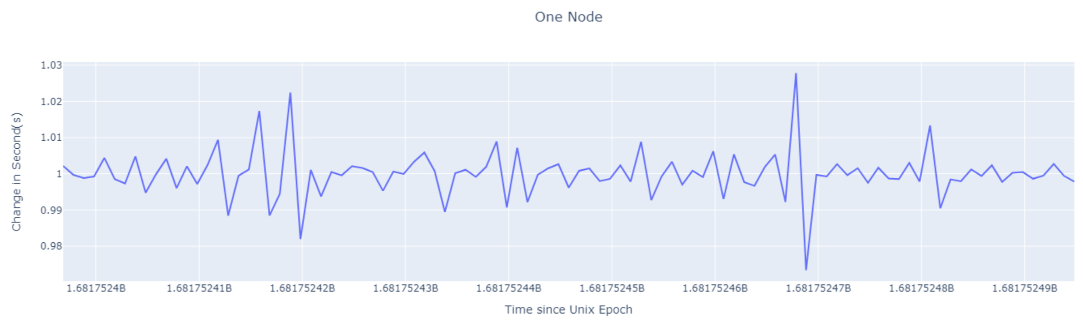
Total work time - 5 hrs

DAILY LOG 4:
Jon, Sat 9/4
We converted the ROS2 program to the correct ROS1 syntax, which required extensive research due to the differences in environment and program structure. Eventually, we managed to get it working perfectly.
We then were given the opportunity to explore ROS further, so we rewrote the code in C++ to test the interoperability of multiple languages within the ROS environment. We also experimented with spawning multiple identical nodes by generating custom XML scripts using a dedicated Python script.
We then started to time the delay of nodes in the message queue depending on the total number of nodes spawned and discovered an unexpected pattern.We analyzed the pattern and discussed potential reasons for the observed behavior.

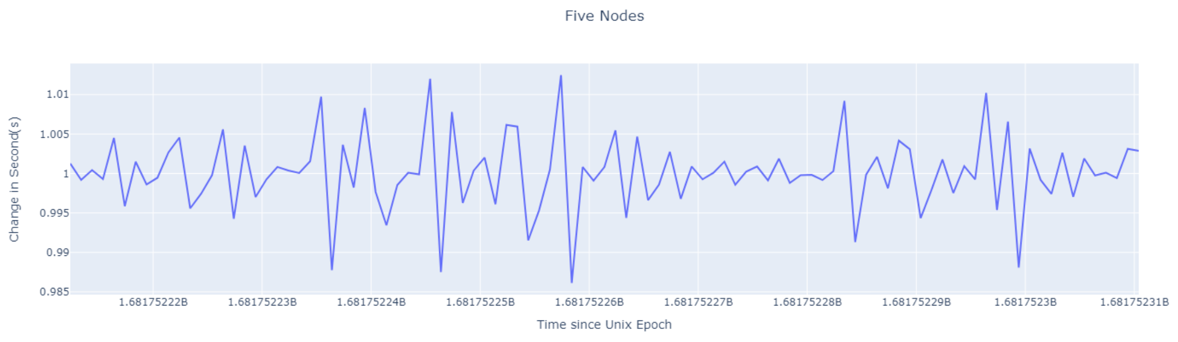
Total work time - 6 hrs

Analysis:

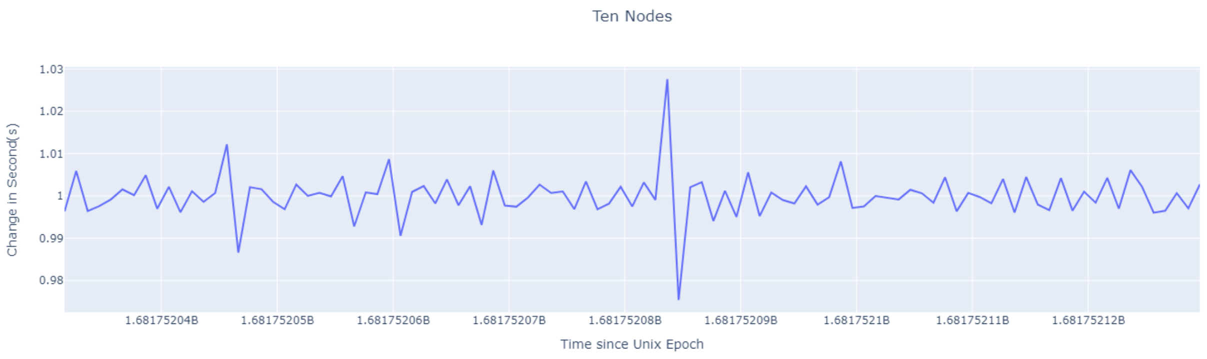
The following graphs show the timing in-between the messages of the same node, when more and more nodes are added to the topic.
So if there is a single node, we measure how long it takes between the messages, but if there are a hundred nodes, we let them all send messages to the same topic, but mark a single one of these nodes as special and measure the timing in between the messages of that single node while ignoring the messages of the other nodes.



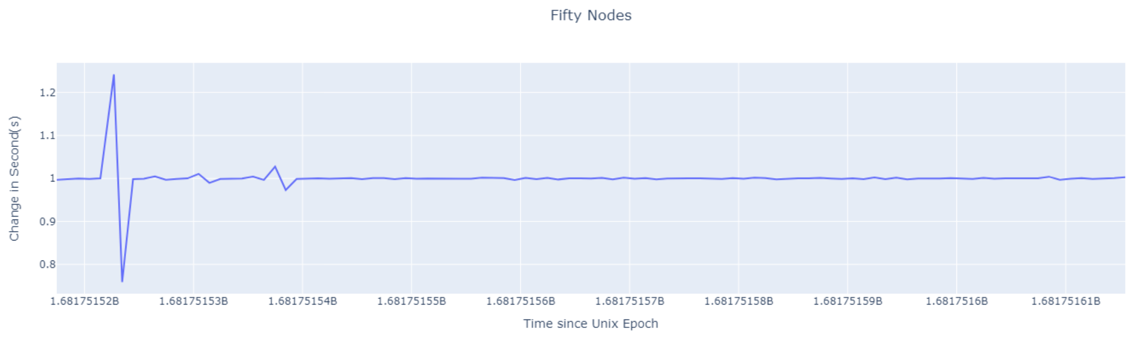
As we can see with a single node, most of the peaks stay below the 1.01 Line, with only three going beyond that. The big spike we can see at Unix time 1681752460 is possibly due to another process doing something in the background, causing a lag in the processing. We assume that, because it keeps happening periodically, for roughly the same amount of time at every occurrence. We tried to isolate as many factors as possible, this being the only user initiated process running at that point in time, so our hypotheses of what might be causing this is a low level operating system process.



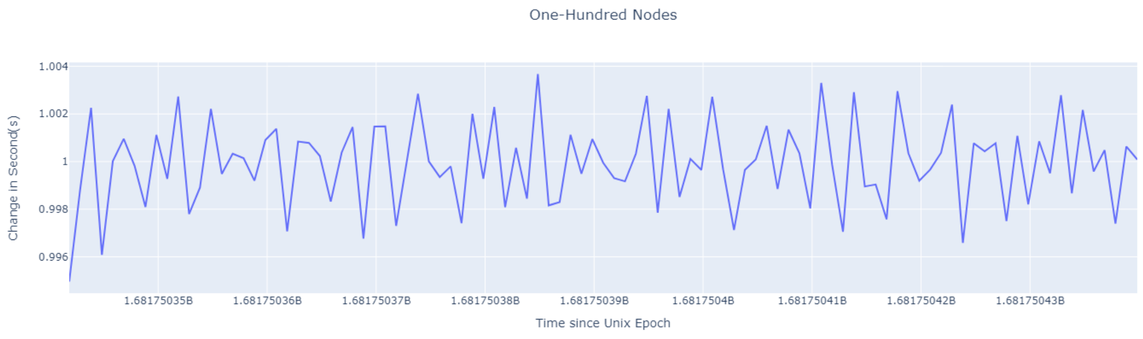
Here we can see the recurrence period of a single node, being influenced by 4 other nodes sending messages at the same time. Here we see little difference, having only 2 peaks reaching past the 1.01 and 4 touching it. This shows little to no difference to the previous measurement, except the large peak, of which we assume that it is caused by an external factor.



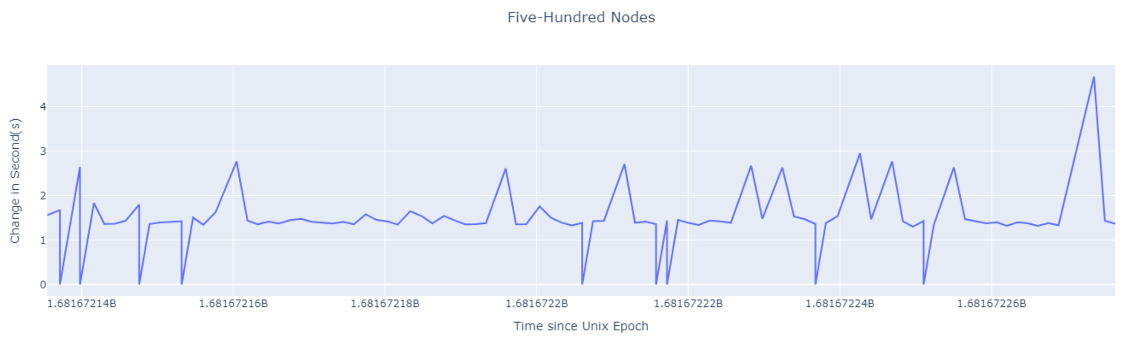
As we can see here, with ten nodes being active, the big spike is of the same size as it was back with one node running. Aside from that, we see little difference during the runtime.



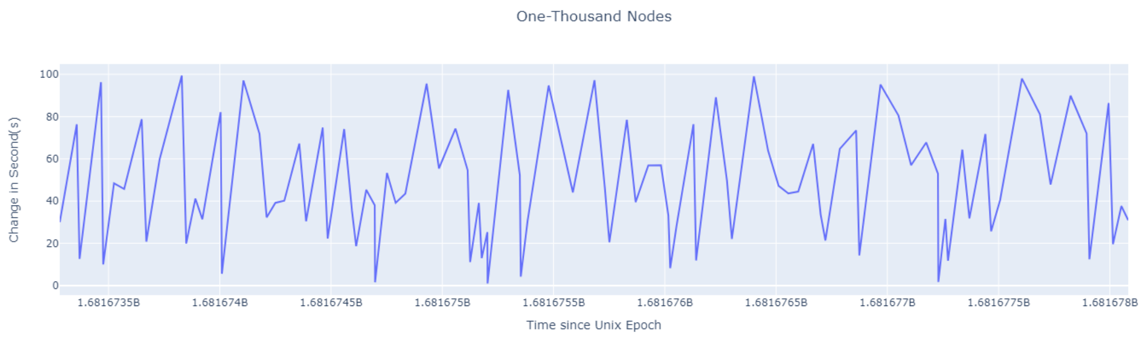
This is the first time that we can see a difference from the previously established pattern. Around 168175192 we can see a large spike of almost 0.06s and 4 peaks beyond the 1.1 mark, indicating that we are reaching some kind of breaking point or resource limitation in our virtual machine.



The one hundred node graphs somehow has a much lower latency, never even going beyond 1.004. We are not sure what might have caused this, possible explanations is the termination of background processes, but to actually confirm this hypotheses further research with a significantly deeper investigation into the concurrent processes of the operating system and automatic resource allocation. We tried to eliminate as many variables as possible for all these experiments by terminating as many non essential processes and ran the python thread with maximum scheduler priority further research is needed.



With five hundred nodes, we can definitely see that we are taking up a large part of the compute resources with other nodes. With at least 9 peaks beyond the 2.5 seconds line and the median response time more than a quarter second delayed.



At one thousand nodes, the virtual machine is barely responsive, utilizing the entirety of the limited ram and swap that was allocated. The period of response is now beyond a minute, showing a total inability to keep up.

In this examination of the messaging delays of a ROS1 node under load, we discovered that when there were only a few nodes, the system performed fairly well, with minor increases in message timing. However, when the number of nodes increased to 100, we saw an unexpected drop in latency, which requires further investigation. As the number of nodes continued to grow, the virtual machine struggled to keep up, with significant delays in message timing at 500 and 1,000 nodes. We attribute these struggles to resource limitations of the Virtual machine, and providing more would result in much better performance.

Summary of Project Journal:
Over the course of this project, our team encountered and overcame various challenges. We began by setting up the virtual environment and ROS environment, and then programmed the Subscriber, Publisher, and counter nodes. We converted our code to ROS1 syntax and explored ROS further by rewriting the code in C++ and spawning multiple identical nodes. By monitoring the performance of the nodes, we discovered an unexpected pattern in the message queue delay. Our team worked together effectively, dividing tasks and supporting each other to ensure the successful completion of the project.