Performance Report

Gabriella Munger, Justin Gomez, Sebastian Fernandez

1.0 Purpose

This program was written to simulate three different variations of the gossip protocol (Push, Pull, and Push-Pull) for the purpose of comparing the efficiency of the methods to each other and finding the relationship between the number of nodes and runtime.

2.0 Requirements

There are no further requirements to run this experiment outside of what is typically required to run a Go program.

2.1 Background Knowledge

For all three variations of gossip protocol in the program there is an implementation of susceptible and infected nodes. Susceptible nodes do not know about the update/message while the infected nodes have the update/message and are actively trying to spread it. In the program the infected node is initialized with a status of true and the message to spread, and the rest of the nodes become susceptible nodes with a status of true and message: “waiting.” All the nodes are assigned to their own go-routine so the gossip protocol runs concurrently.

For push gossip, the infected nodes are active and are trying to update the rest of the susceptible nodes, changing their status to true and pushing the new message. In each round, the infected node will push the update to a susceptible node, and this process will continue each round until all the nodes are infected. Secondly, pull gossip and push-pull gossip work with the infected nodes being passive and the susceptible nodes are active. In pull gossip for each round, susceptible nodes will randomly pick another node to retrieve the status and message. The process continues until all nodes become infected. Push-Pull gossip utilized both of the previous gossip protocols. While less than half of the nodes are infected, the protocol will implement push gossip, and one half the nodes are infected the protocol will switch to pull protocol.

3.0 Instructions

To run this program, first clone the git repository. Then, type “go run main.go” into the terminal line. The program will then ask the user to type the desired message into the terminal. Next, choose the desired gossip protocol by typing the letter corresponding to that algorithm.

To gather enough data for analysis, run the program several times. The number of nodes in the system can be changed by editing the source code according to the instructions commented at the top of the main.go file.

4.0 Output

The program will output the total time it took for the message to spread to each node in the system.

4.1 Code Output

This example shows an input of "*systems*" as the message, and "*a*" to indicate the Push protocol in a system of 100 nodes. The output displays the time taken to complete the gossip protocol.

```
Hello! Type a word that you would like to send
systems
Enter the letter corresponding to the desired gossip algorithm:
a. Push
b. Pull
c. Push-Pull
a
Great! We will send <systems> using the Push algorithm on a system of 100 nodes.
Gossip Protocol is complete!
The protocol took  16.310402ms
```

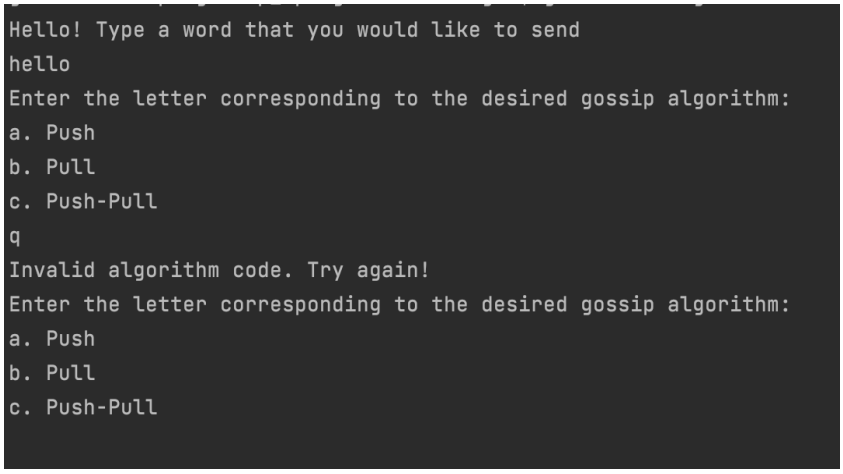
This example shows an input of "*message*" as the message, and "*b*" to indicate the Pull protocol in a system of 100 nodes. The output displays the time taken to complete the gossip protocol.

```
Hello! Type a word that you would like to send
message
Enter the letter corresponding to the desired gossip algorithm:
a. Push
b. Pull
c. Push-Pull
b
Great! We will send <message> using the Pull algorithm on a system of 100 nodes.
Gossip Protocol is complete!
The protocol took  16.592857ms
```

This example shows an input of "*hello*" as the message, and "*c*" to indicate the Push-Pull protocol in a system of 20 nodes. The output displays the time taken to complete the gossip protocol.

```
Hello! Type a word that you would like to send
hello
Enter the letter corresponding to the desired gossip algorithm:
a. Push
b. Pull
c. Push-Pull
c
Great! We will send <hello> using the Push-Pull algorithm on a system of 20 nodes.
Gossip Protocol is complete!
The protocol took  1.122857ms
```

If the input for the message includes a space, for example "*hello world*", then the program misinterprets the input and will only accept the first word. It reads the second word as input for the next question of which protocol to follow, which is invalid input. This does not result in an issue because the input function continues looping until a valid input is accepted.



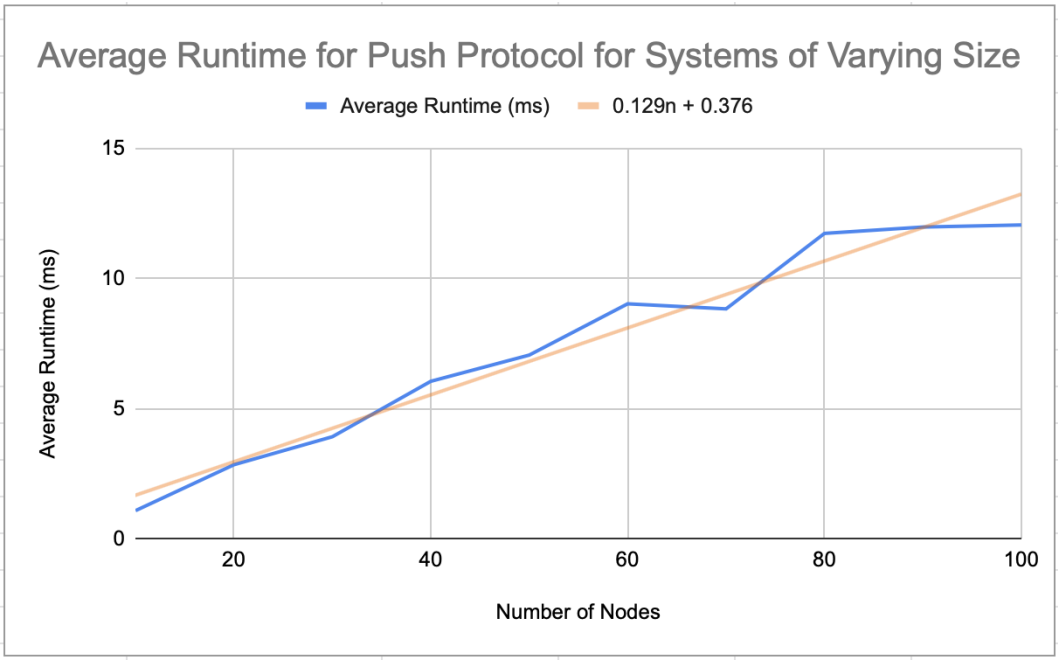
4.2 Data Analysis

This project investigates the difference in convergence time between the three gossip protocols developed in the program, and also the relationship between nodes. This was done by running the program nine times for each protocol for systems with ten nodes first. To obtain data over different system sizes, this process was repeated while incrementing the system size by ten until the test was completed on a 100 node system. To maintain consistency, the same message "*a*" was used for each test. This data was then put into a Google Sheet for creation of charts. The trendline function in Google Sheets was used to find the relationship between the number of nodes, *n*, and the convergence time (or runtime) for the protocol.

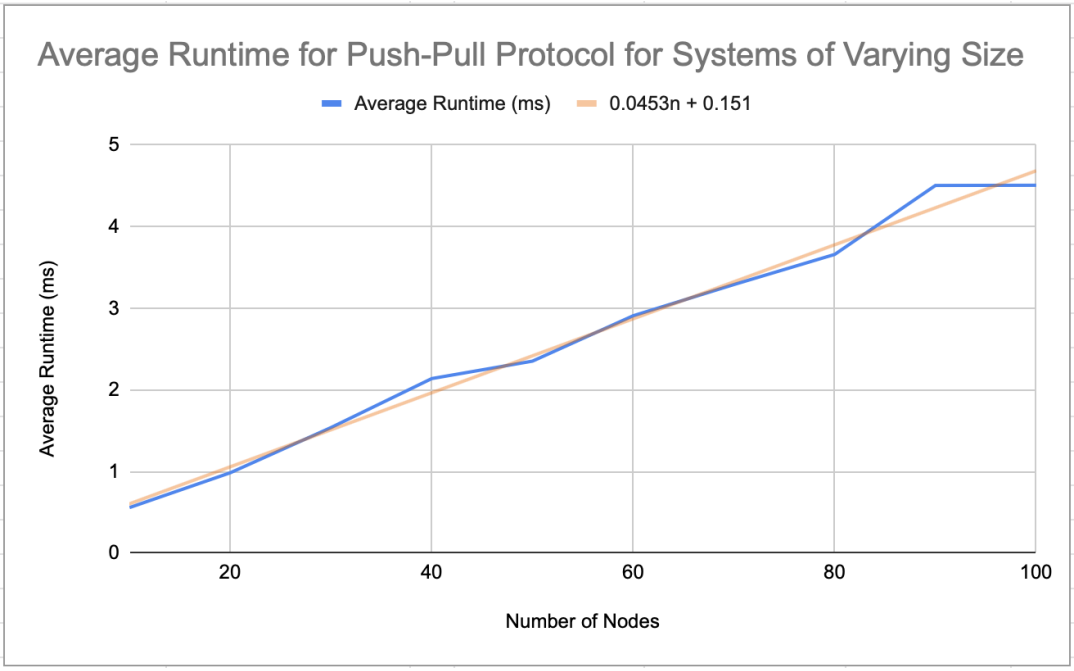
4.3 Discussion of Results

4.3.1 System Size vs. Convergence Time

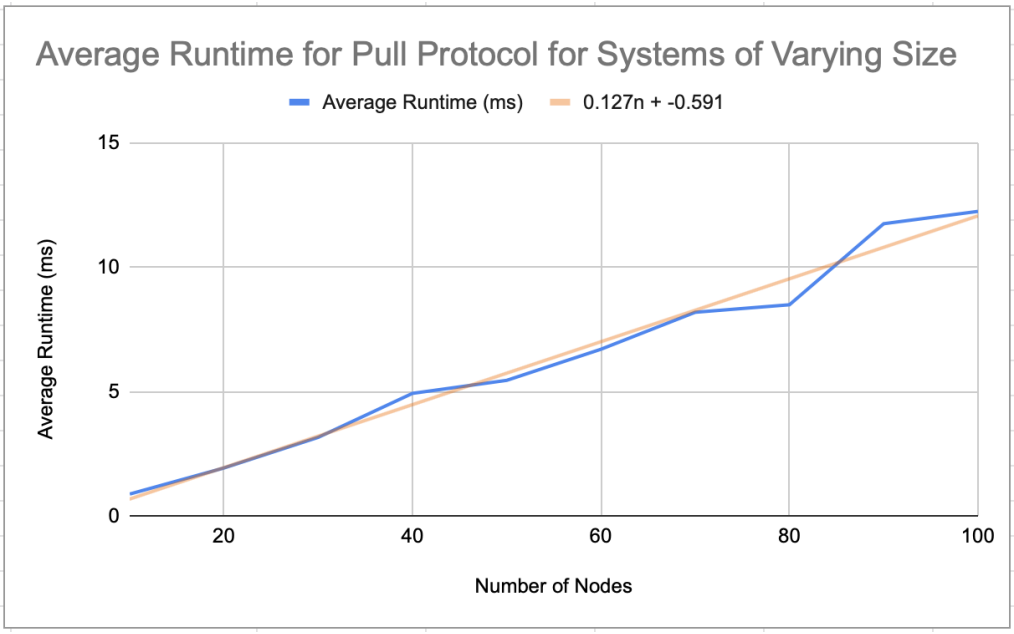
The first figure below shows the relationship between the number of nodes in a system and the average runtime in milliseconds for the push protocol. As shown in the figure, this relationship can be approximated by a function of *n* nodes where the average runtime in milliseconds is equal to 0.129n + 0.376.



The next figure below shows the relationship between the number of nodes in a system and the average runtime for the pull protocol. This relationship can be approximated by a function of *n* nodes where the average runtime in milliseconds is equal to 0.0435n + 0.151.

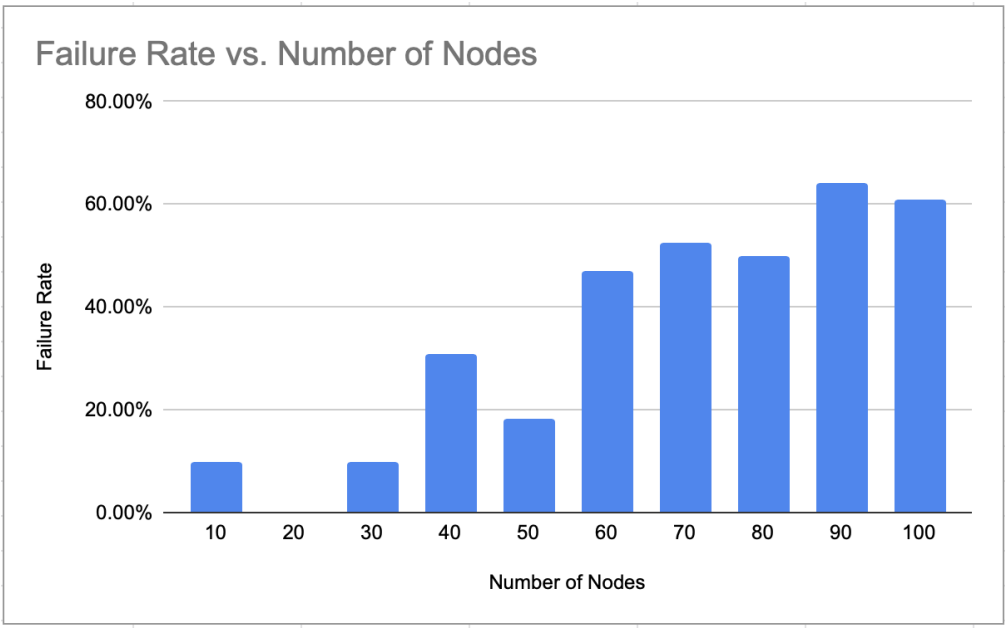


The next figure illustrates the relationship between the number of nodes in a system and the average runtime for the push-pull protocol. This relationship can be approximated by a function of *n* nodes where the average runtime in milliseconds is equal to 0.127n - 0.591.



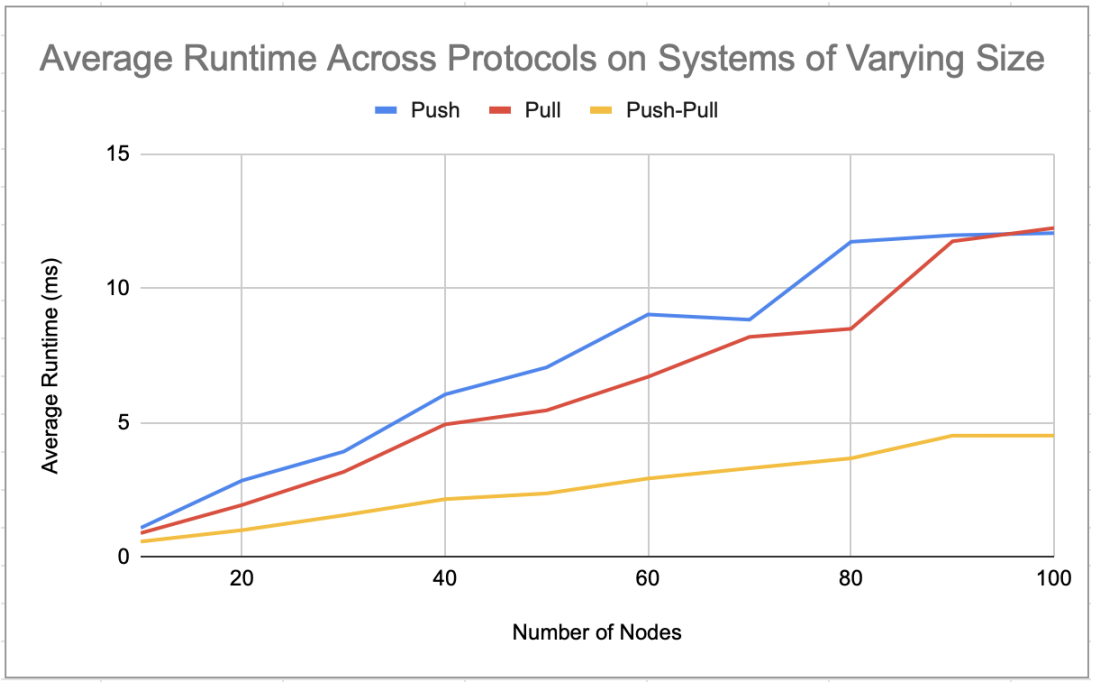
Overall, the results indicate a relatively linear relationship between system size and convergence time for all three protocols on systems with 10 to 100 nodes. Using nine repetitions to find the average runtime for each system size may result in error due to a small sample size, however it was the most feasible number of repetitions for this execution of the experiment as the data had to be manually transferred to a spreadsheet.

It is of note that execution of the program for the push-pull protocol sometimes results in a deadlock and “exit status 2” is printed in the terminal. This issue was not resolved as of completion of this report, however it seems that this issue arises more often as the system size increases. This seems logical, as there are more nodes and goroutines that could have a deadlock issue causing the program to stop executing. For this experiment, trials were repeated until there were nine successful executions of the push-pull protocol for each system size, however data was also gathered on the number of failures, as shown below.



4.3.1 Convergence Time vs. Protocol

The figure below displays the data collected for each of the three protocols on the same axes for the purpose of comparing the average runtime between the different protocols. In this implementation of the gossip protocols, it was observed that the push-pull protocol was consistently the fastest on average. While the push protocol typically took slightly longer than the pull algorithm, this was not the case once the system had 100 nodes. This was the expected result, since it is known that it is faster at first for infected nodes to push their message to a random node, as there is a higher probability that the node they choose to push to will be susceptible and thus the push will result in a successful infection. Once more than half of the nodes are infected, it is more efficient for susceptible nodes to try to pull the message from a random node, as the random node chosen is more likely to be infected and result in a successful infection.


Average Runtime Across Protocols on Systems of Varying Size
Push
Pull
Push-Pull
15
10
5
0
Average Runtime (ms)
20
40
60
80
100
Number of Nodes