Finding a Minimum-Weight Spanning Tree Using Gallager, Humblet and Spira's Algorithm

Jens de Waard
4009215

Tim Wissel
4xxxxxx

January 8, 2019

Abstract

Enter a short summary here. What topic do you want to investigate and why? What experiment did you perform? What were your main results and conclusion?

1 Introduction

For exercise 3 of the course Distributed Algorithms we implemented the algorithm by Gallager, Humblet and Spira for determining a minimum-weight spanning tree of a distributed network. In this report we describe the details of our implementation in section 2 and the results of running the algorithm on two types of networks, manually and automatically created, are listed in section 3.

2 Implementation

We've chosen to implement the algorithm in Java RMI. This was a requirement for the previous assignments and as such we already had some code to work with. Had we chosen to implement the algorithm in Python, we would have had to start from scratch.

The implementation adds artificial delays when sending a message to simulate the latency that occurs on true networks. Due to this delay, it is possible that two messages send in quick succession over the same edge do not arrive in the proper order; the later message will overtake the first. This is a violation of the FIFO property of the network that the algorithm depends upon.

To counteract this, all messages are timestamped using a scalar clock that is mainted by every node for each of its outgoing edges. Each node also maintains for every incoming edge what the timestamp of the next message must be. Any message that has a higher timestamp than expected is deferred until all prior messages on the corresponding edge are received.

2.1 Generation of test cases

Creating large networks by hand would be very difficult and error-prone. Due to this we created a bash script to output valid graphs that could be read by the implementation. The script created graphs of sizes ranging from 5 to 100 nodes in steps of 5 with a chance p of an edge between each possible pair of nodes, which $p \in \{0.25, 0.5, 1\}$. For each combination of size and p, 5 test cases were created. However this didn't matter for when $p = 1$ as all fully-connected graphs of size n are equal. In those cases, there is still additional value in having multiple tests as there is a certain randomness in the implementation due to the delays and the specific node that starts the algorithm.

3 Results

Figure 1 shows the relationship between the number of nodes in a graph and the total number of messages sent by the algorithm. The blue line corresponds to the number of messages that our implementation of the algorithm sends, while the red line is the upper bound to the number of messages as described by Gallager, Humblet and Spira in their paper. This upper bound is $O(5 * n \log n + 2E)$.

For higher values of n, the implementation sends more messages than the expected upper bound would allow. This is especially noticable in the fully connected networks.

4 Conclusion

References

[1] K. Grove-Rasmussen og Jesper Nygård, *Kvantefænomener i Nanosystemer*. Niels Bohr Institute & Nano-Science Center, Københavns Universitet

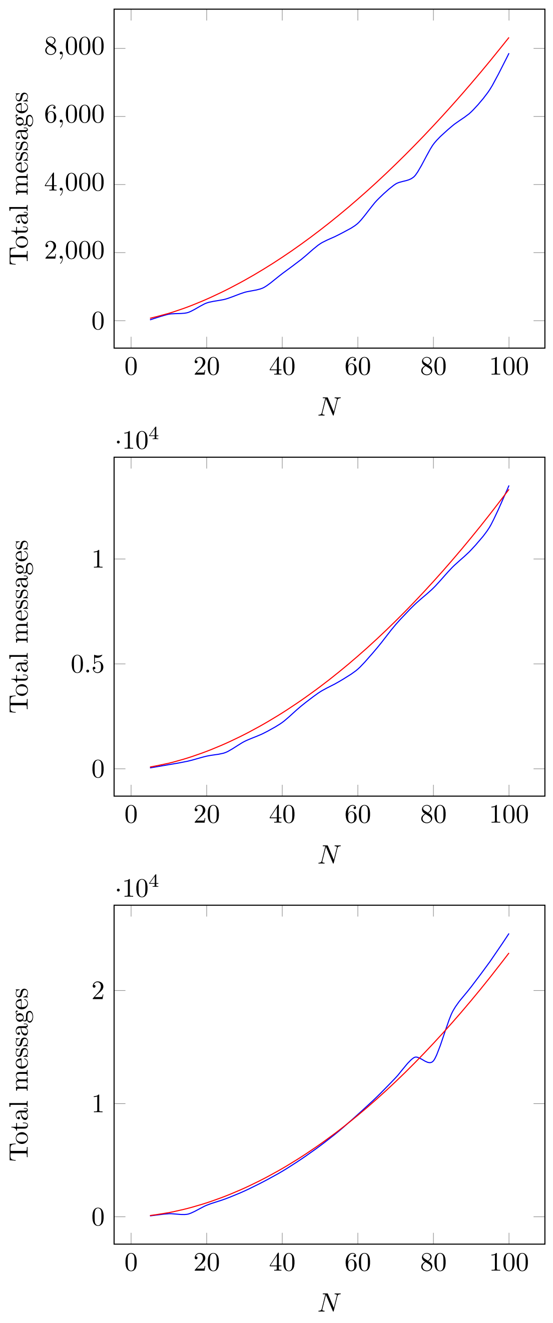

Figure 1: The total number of messages and the expected number of messages for $p = 0.25$, $p = 0.5$ and $p = 1$.