Study on network message propagation: Anti-Entropy Epidemic Algorithms

Flávio Silva
up201505646@edu.fe.up.pt

Miguel Cardoso
up201106812@edu.fe.up.pt

Manuel Hess
up202102422@edu.fe.up.pt

***Abstract*—This paper addresses the study of epidemic algorithms for information dissemination in a network, a scope of the CU (curricular unit) of Distributed Systems. Only anti entropy type algorithms were considered in this study for simulation, that was developed with python. The objective is to observe the behavior of infection and message transmission on different conditions, and conclude on what is the best algorithm.**

1. Introduction

An important part of an integrated system is the communication, that is, the exchange of information between the different devices connected that form a network. One of those is multicast communication, which objective is to disseminate information from one sender to multiple receivers. Epidemic algorithms is a way of achieving multicast communication, by the means of propagating a message through a network the same way a disease spreads through a group of people, with the important distinction that in the case of a network we want it to spread as quickly as possible as opposed to a disease.
Alan Demers was the first to write about epidemic algorithms in 1987 [2], with the goal of developing a multicast communication protocol that was efficient, robust and scalable. The main algorithms for disseminating information until then were based on setting up a tree on top of the actual layer (not easy to implement) or flooding the network with messages (required a lot of attention because of the resource waste). Demers then presented two different propagation models: anti-entropy and gossiping.
In the anti-entropy model, a given node chooses a random node from the list of neighbors (nodes that it's connected to), and exchanges information with it. The way the two nodes interact constitute the type of anti-entropy:

- **Push** - node P only pushes information to Q;
- **Pull** - node P only pulls information out of Q;
- **Push-Pull** - nodes P and Q exchange information between them.

In this paper, we focus on the anti-entropy models, namely its behavior on different network topologies. Demers presented an expression for each type, describing the probability of a given node not being infected after n rounds, that translates to the graph in figure 1.

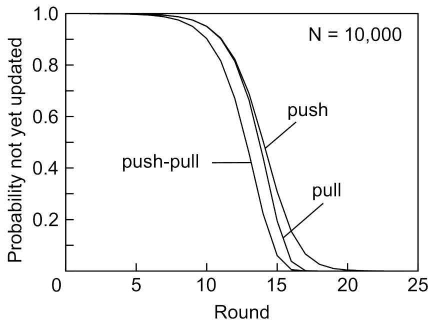

Figure 1. Expected behavior. Extracted from [1]

2. Solution description

In order to simulate the spreading of a message in a network and evaluate its performance, a program in a python environment was developed, as it is easy to develop with, and it's also the state-of-the-art programming language when it comes to data analysis. The goal is to analyze the behavior of the anti-entropy algorithms for a network under perfect conditions (no hardware or transmission failures), and then see what happens in more realistic conditions, i.e. with node crashes and message errors.
The program will generate as output the percentage of infected nodes by the number of rounds that were simulated (similar to the one in figure 1), the number of infected nodes per round and the number of messages transmitted per round. It's important to note that each round relates to a time unit, that is equal to the time it takes for a message to go from one node to another (arbitrated as equal for any pair of nodes in the system), which means the simulation developed is synchronous. The output is dependent on the set of parameters that the program receives as input, which adjust either the network size and/or shape (topology, number of nodes), or the way the system behaves (first infected node, anti-entropy type, probability of a node crash, probability of a message not arriving at the destination).
The input related to the network parameters are self-explanatory and are used combined with networkx (a "python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks" [3]) to generate the nodes and its connections. As for the other parameters, it's important to understand them as they

are vital to the simulation and the analysis that will be done afterwards:

- The **first infected node** is related to the node that has the message to be transmitted through the network, and can be a random node (selected by the user), the node closest to the center, the node that has the most connections (neighbors) or the node where the node density is higher;
- **Anti-entropy type** can be one of the three already mentioned, push, pull or push-pull;
- **Probability of a node crash** refers to the probability that a node will be removed from the network, and it's arbitrated that the node won't recover for the rest of the simulation. This may occur with several nodes;
- **Probability of a message not arriving at the destination** relates to transport problems, meaning the message is transmitted but never received by the other node. This may happen several times according to the specified probability.

All the information regarding the simulation (like infected nodes, number of messages, infected nodes, etc) is kept on lists, that act just like a data structure. The simulation is based on threads, in which each thread is a representation of a node. They run ”simultaneously”, meaning all nodes will process its task at the exact same time (simulation time). Each thread will start knowing its 'id', and the list of neighbors that it's connected to. Then, the set of operations that will run is dependent on the anti-entropy type, and is described by the following diagrams.

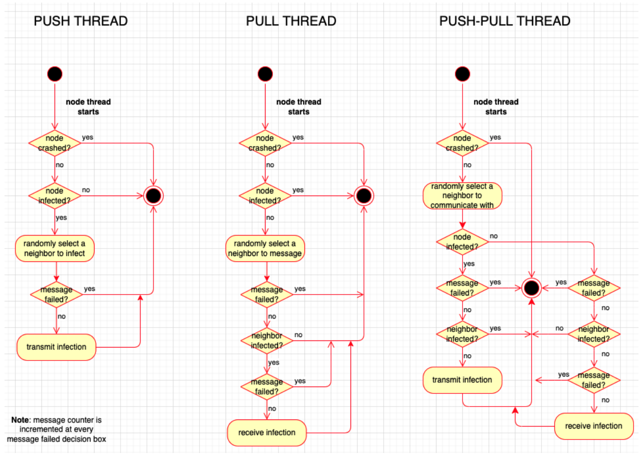
PUSH THREAD
PULL THREAD
PUSH-PULL THREAD
node thread starts
node crashed?
yes
no
node infected?
randomly select a neighbor to infect
message failed?
transmit infection
randomly select a neighbor to message
neighbor infected?
receive infection
randomly select a neighbor to communicate with
Note: message counter is incremented at every message failed decision box

Figure 2. Diagram of the algorithms implemented

Each time a simulation is run (it runs until all nodes available are infected), it does the exact same simulation 30 times in order to get a representative and relevant number of samples.
It should also be noted that for the push-pull algorithm, we considered each node is independent, meaning all of them will contact a neighbor. Regarding messages, this means each node will send 2 messages and also receive 2 messages, even if the neighbor has an outdated information (information exchange will always happen).

3. Conclusion

3.1. Results

Even though this implementation is very different from the implementation used in [4], it was used as a guide and to verify some of the results. Starting with the percentage of infected nodes by the number of rounds, the results are the same, with the exception of the tree (because we used a random tree instead of a binary tree) and the mesh topology, but even these last one shows the same tendencies on both. Given we had the same results, please refer to [4] for all the conclusions about node infection propagation.
The observation of these results were made using plots just like the one in figure 3, that contains the percentage of infected nodes, the number of infected nodes by round and the evolution of messages transmitted.
As for the message transmission, we considered it was important not only to consider the number of nodes in the network, but also the number of rounds to infect them all. The results are shown in the table below, and they were obtained with 200 nodes, no failures and the first infected node is the one in the center.

	Mesh	**Full Mesh**	**Tree**	**Star**	**Line**
Push	0.664	0.537	0.64	out	out
Pull	0.433	0.642	0.367	2	0.393
Push-Pull	0.433	1.769	1.956	4	1.982

TABLE 1. MESSAGE DENSITY - MESSAGE TRANSMITTED BY NODE BY ROUND. 'OUT' MEANS THAT IT COULDN'T BE SIMULATED BECAUSE IT TAKES TOO MUCH TIME TO RUN EVEN A SINGLE TIME SIMULATION.

For the purpose of simulating message failure and nodes crashing, we considered a given anti-entropy type and topology, because the study for every variable we have in the system would lead to too much data. Push-pull type was used since it is the fastest to achieve maximum infection, and mesh topology, because it is the most realistic one. The results for message failure are shown in figure 4, and for node crashing in figure 5.

3.2. Discussion

Regarding message propagation, the push-pull type is the fastest, but not necessarily the best. If the number of messages is an important factor for the design of the network, it may not be the best. Considering a mesh topology, and looking at the results we got, the results are acceptable, but not optimal. The best result in the table is for a tree topology using pull type so, in the design of a mesh topology that should be optimal, the designer may build a tree on top of the physical layer, and disseminate information based on the list of neighbors of that tree.
The results we got regarding message failure are pretty much the ones we expected. The higher the percentage of

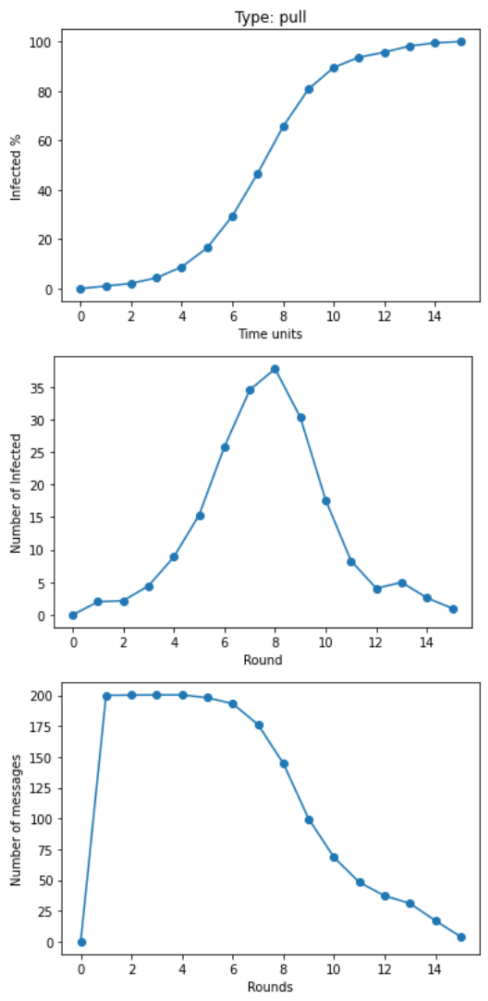

Figure 3. Example of an output after a simulation. The one present is extracted for pull type, 200 nodes and no failures, full mesh topology

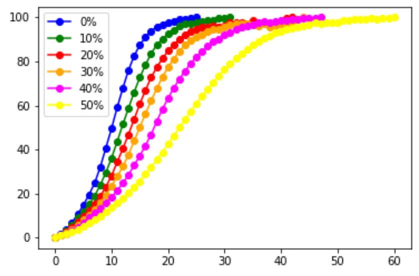

Figure 4. Percentage of infected nodes by round, comparing it for a different probability of message failure

messages not arriving at their destinations, the longer it takes to infect all of the network. It's also interesting to note that the curve maintains its characteristic, that is, its general shape, but flattens for higher values.
The node crashing analysis is probably the most interesting one. The first thing we noticed is that strange behavior in the end of the rounds. That happens because of the way we calculate the average of the the several simulations. A better algorithm, such as a moving average, would produce better results but, even with that weird behavior, the observation of the plot shows that the higher the node crash probability, the lesser nodes are in the network, leading to less rounds to achieve infection on all nodes. Another thing that we observe is that the results follow logarithmic functions, because the drop of the first 10% is much bigger than the rest, and it reduces every 'step'.
It was interesting to see that even with 50% of node crashing, the simulation could reach all available nodes. We were expecting that islands would be created, meaning the information would never reach those nodes. That was not observed, so we concluded that the way we formed the mesh topology (using networkx), for 200 nodes, has a low probability of forming said node islands. Even though it was a not a full mesh all the nodes had several connections.

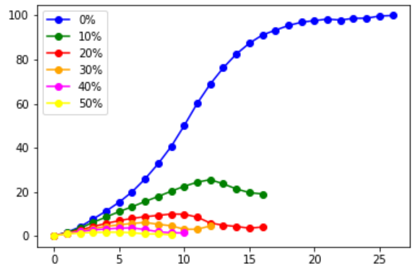

Figure 5. Percentage of infected nodes by round, comparing it for a different probability of node crashing

4. Members contribution

- Flávio Silva (45%) - Simulation construction, data extraction
- Miguel Cardoso (45%) - Simulation construction, algorithms diagram
- Manuel Hess (10%) - Project discussion

References

[1] Tanenbaum, Andrew S., and Maarten van Steen. 2007. Distributed systems: principles and paradigms. Upper Saddle River, NJ: Pearson Prentice Hall.

[2] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. 1987. Epidemic algorithms for replicated database maintenance. In Proceedings of the sixth annual ACM Symposium on Principles of distributed computing (PODC '87). Association for Computing Machinery, New York, NY, USA, 1–12. DOI:https://doi.org/10.1145/41840.41841

[3] "NetworkX documentation," NetworkX, 2014. [Online]. Available: https://networkx.org/. [Accessed: 12-Jan-2022].

[4] Hélder João Loureiro Pereira, José Martinho Oliveira Peres. Algoritmos epidémicos.