Data Dissemination in Distributed Systems

By Greg

CSC330-Fundamentals of Distributed Systems

07/04/2025

1. Introduction

This report looks at how data is disseminated across distributed networks through the use of what are known as epidemic, or gossip, algorithms. Their name comes from their design being inspired by how real epidemic diseases and gossip spread, and their suitability for distributed systems comes from the fact that they allow for "decentralized peer-to-peer communication" [1], and also they do not require awareness over the state of the whole network, only that the entities in the network are aware of their neighbors and can communicate with them. The goal of these algorithms is to help the distributed network reach a consistent state, or in other words, to get all the nodes within it to have the same information. This concept is referred to as convergence. As a sidenote, these algorithms are strictly for spreading data, not for deciding which data is the most accurate and up to date. The task of deciding on the most valid state to have depending on the data, is the job of consensus algorithms like Raft and Paxos, which we will not be going over. Moreover, there are various categories of epidemic/gossip algorithms, with some of them being deterministic, probabilistic, symmetric, and asymmetric. So, we will be looking at the Anti-Entropy and Rumor Spreading algorithms, both of which have versions belonging to each of the categories mentioned [2].

NOTE: The versions of the algorithms that are compared in this paper are the probabilistic **Push** implementation for Rumor Spreading, and the deterministic **Push, Pull,** and **Push-Pull** implementations for Anti-Entropy. The reason these ones were chosen is because they alone are covered in the week 3 presentation on distributed system communication.

Deterministic vs Probabilistic: Algorithms whose behavior is fixed or completely predictable are called deterministic. They do have any elements of randomness in them, so their output or behavior will always be dependent on the state of the system and the input taken. Probabilistic algorithms, on the other hand, will have randomness in at least some part of their decision-making process. This does not mean that they are completely unpredictable, for example, statistics can be used to guess the behavior that will be displayed most often, but since that randomness exists, they cannot be considered as reliable as deterministic ones.

Symmetric vs Asymmetric: Algorithms which involve entities interacting with each other in some way, and where all of them have the same role are called symmetric. As an example, in

the push-pull implementation of the Anti-Entropy algorithm when two nodes interact, they both exchange information, i.e., they push their own state to the other, and at the same time pull the other's state to themselves. So, when the interacting entities take up distinct roles or behave differently, it is considered an asymmetric algorithm [3, p. 269].

2. Rumor Spreading

This algorithm is the fastest at spreading data out of the two that we will be seeing, and this is due to its proactive nature. Moreover, it functions in a probabilistic manner because it has an element of randomness in one of its stopping conditions, making it so that each node has a non-zero chance of remaining uncontacted, in which case the dissemination process will have ended prematurely.

Idle Stage

When a node is in this stage, it is either "susceptible" or removed. When susceptible, it simply waits to receive a message from another node, and until then it does nothing else. When it is removed, it means it has already received the rumor and it stopped spreading it [4, p. 9], [5, 171]. Only in the pull and push-pull implementations does it actively contact neighboring nodes to check if they have new data that it can receive.

Spreading Stage

A node enters this stage when it becomes "infected", meaning it receives the message/rumor carrying new data, and it immediately starts spreading it to its neighbors randomly without any other events needing to happen and only stops when forced to. Going a bit further into the mechanism, when the node becomes infected it initializes a variable that represents the probability it has of stopping the spread of the rumor upon contacting an already infected neighbor. The initial value is relatively low. Moreover, each node keeps track of the neighbors it has already contacted, so when it chooses a random neighbor to message it does so from its list of uncontacted neighbors. If the node's list of uncontacted neighbors is empty, then it stops and becomes idle again by setting its status to "removed". Additionally, as mentioned above, the node has a chance of stopping the spread and going back to the idle stage when it sends the rumor to a neighbor that has already received it. This happens when another node that has the same neighbor got to it faster. Also, if it does not stop on this occurrence of the event, then it will increase, by some factor, the value of the variable representing the chance to stop. The probability of stopping like this can eventually reach 100%. With both these stop conditions, the node cannot accidentally get stuck spreading the same information indefinitely [4, p. 9], [5, 171].

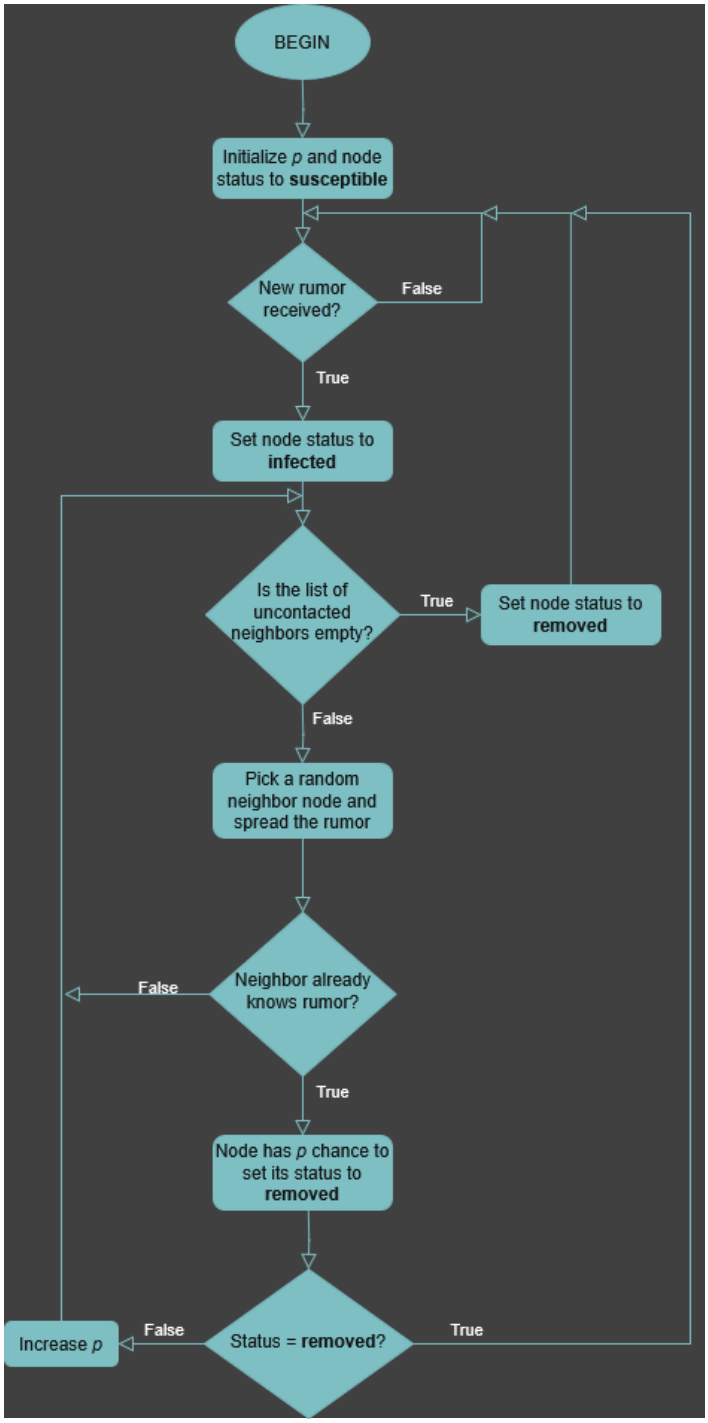

Figure 1: Node behaviour in the Push implementation of Rumor Spreading

3. Anti-Entropy

This algorithm is commonly implemented in a deterministic way, unlike Rumor Spreading. Additionally, it ensures eventual consistency/convergence in the network. The main principle behind its dissemination mechanism is the comparison of states between nodes and their

synchronization [4, p. 9]. Moreover, the comparison of states between nodes is done periodically, so nodes do not really have an idle state in the same sense that they would if Rumor Spreading was used.

Push-Based Implementation

Each node periodically contacts a random neighbor and pushes their total state, i.e., all the information they hold, onto them. The receiving node will then proceed to synchronize itself with the sender's data [5, p. 171], [6], [7, p. 1-2].

Pull-Based Implementation

Like in the push implementation, nodes contact random neighbors, but instead of pushing their total state they compare their state with their neighbor's, and if there is a discrepancy between them then they request only the new data they do not have and need in order to synchronize [5, p. 171], [6], [7, p. 1-2]..

Push-Pull-Based Implementation

This implementation uses a symmetric algorithm, as mentioned in the introduction, unlike the push-based and pull-based ones which are considered asymmetric. When nodes interact, after comparing states they let each other know which of their information is outdated so they can exchange only the data each needs to synchronize and become up-to-date [5, p. 171], [6], [7, p. 1-2].

4. Comparative Analysis

Convergence Speed

When talking about speed, Rumor Spreading is generally faster at dissemination information than Anti-Entropy. One of the main reasons for this, is the delay Anti-Entropy has because of the period checks nodes have to perform. This means that new updates do not start propagating as soon as they appear, and when they do start the propagation speed fairly affected by how frequently nodes interact with each other.

Scalability

Both of these algorithms are pretty scalable, which makes sense since they were designed for distributed systems. On the one hand, due to the higher propagation speed of Rumor Spreading dissemination rates can remain quite stable as a network becomes bigger but the load on it can increase significantly during active dissemination due to a high number of redundant transmissions caused by the nature of the protocol. On the other hand, Anti-Entropy's dissemination speed might suffer by size increases, but because of how efficient it is, especially its push-pull implementation, network load will remain consistently stable.

Robustness

In terms of robustness and adaptability to different network conditions, Anti-Entropy is better at Rumor Spreading, at least when comparing the versions discussed here. Rumor Spreading's effectiveness counts a lot on the density of the network, which basically means that depending on how many connections exist between the nodes, the better it performs. If, for example, some nodes do not have high connectivity because of their topology or because they are at the far edges of the network, the likelihood of them missing new data or some update is increased.

Fault Tolerance

The eventual consistency that Anti-Entropy entails, makes it the more tolerant protocol. In cases where we are presented with a high number of node failures, we know that when they come back online they will be able to "catch up" with the rest, whereas with Rumor Spreading, even though data dissemination might finish, if they came back online after it, they would have to wait until the next update to reach convergence.

5. Conclusion

Having gone though both algorithms, we can see that they each have some areas that they are better suited to. In a distributed network where conditions can vary and consistency is the most important aspect, Anti-Entropy would be more fitting, but in a dense network well connected network where we mostly care for the fastest propagation possible, Rumor Spreading would be more ideal. The best general solution though, is their combination. Making a hybrid protocol by implementing both algorithms we can exploit their strengths for the most optimal results. This is not some groundbreaking modern technique, as its use was even mentioned by Xerox researchers in an academic paper about database maintenance all the way back from 1989 [4, p. 20-21]. They recommended employing Rumor Spreading as the first step for data dissemination because it would be able to propagate the update to, at the very least, the majority of the network in very little time, and Anti-Entropy mechanisms, i.e., state comparisons, would be used afterwards just to make sure that there were not any nodes left without the update.

6. Bibliography

[1] N. Kim, "Gossip Protocol," *System Design*, Jul. 09, 2023. Accessed: Apr. 7, 2025. [Online]. Available: https://systemdesign.one/gossip-protocol/

[2] B. Haeupler, "Simple, Fast and Deterministic Gossip and Rumor Spreading," *Journal of the ACM*, vol. 62, no. 6, pp. 1–18, Dec. 2015, doi: 10.1145/2767126

[3] P. Poonpakdee, J. Koiwanit and C. Yuangyai, "Epidemic Algorithms on Distributed Systems Towards Industry 4.0," *2017 21st International Computer Science and Engineering Conference (ICSEC)*, Bangkok, Thailand, 2017, pp. 1-5, doi: 10.1109/ICSEC.2017.8443937

[4] A. Demers *et al.*, "Epidemic Algorithms for Replicated Database Maintenance," *Xerox Palo Alto Research* Cente, vol. 22, no. 1, pp. 8–32, Jan. 1988, doi: 10.1145/43921.43922

[5] A. S. Tanenbaum and M. V. Steen, *Distributed Systems : Principles and Paradigms*, 2nd ed. Upper Saddle River, N.J.: Prentice Hall, 2007. [Online]. Available: https://csc-knu.github.io/sys-prog/books/Andrew%20S.%20Tanenbaum%20-%20Distributed%20Systems.%20Principles%20and%20Paradigms.pdf

[6] "Anti-Entropy in Distributed Systems: Data Sync Guide," *Endgrate*, 2024. Accessed: Apr. 7, 2025. [Online]. Available: https://endgrate.com/blog/anti-entropy-in-distributed-systems-data-sync-guide

[7] R. V. Renesse, D. Dumitriu, V. Gough, and C. Thomas, "Efficient Reconciliation and Flow Control for Anti-Entropy Protocols," *LADIS '08: Conference on Large Scale Distributed Systems and Middleware*, New York USA, pp. 1–7, Sep. 2008, doi: 10.1145/1529974.1529983