Gossiping with Append-Only Logs in Secure-Scuttlebutt

Cosimo Bortolan
Student ID 215025
cosimo.bortolan@studenti.unitn.it

1 Introduction

This work is intended to provide a possible implementation and an experimental analysis of the protocols described in the paper *Gossiping with Append-Only Logs in Secure-Scuttlebutt* [1]. To support our analysis and to provide a mechanism for future investigations we built a simulator using the Repast Simphony framework. Through this simulator is possible to follow the execution of the protocols graphically and to analyse their performance with the support of some charts which describe the main metrics of the system. Moreover, it is possible to configure the simulation through a set of parameters that allow exploring different configurations of the protocol and the effects of particular choices in the network topology.

In this document, after a short description of the architecture of the system and the technical details of the simulator, we will present some results based on the analysis of the protocol, toghether with some conclusions on its advantages and its limits. Finally, a brief guide is provided to install and run the simulator.

The implementation provided with the simulator follows the steps of the original paper refferring to two possible gossiping algorithms, one in which all participants store and replicate all the messages exchanged in the network (Open gossip), and one in which participants can choose which peers they want to follow and the ones they are not interested in (Transitive-Interest gossip). After the description of the implementation we will analyse the two protocols togheter with some possible configurations of them.

2 Architecture of the system and implementation details

Repast Simphony is an agent-based framework. Agents have proper behaviour and can interact with other agents to obtain system-wide results. Agents are added and deleted by the system and their actions are scheduled by the system at a given rate.

Our implementation is based on the `Participant` agent, which can receive and send messages to other participants (participants are also addressed as *nodes* of the system). Each of these nodes is a fully autonomous entity which behaves according to the same rules. Through the interaction of a number of these agents, we can simulate possible executions of the protocol.

Participants are created at the beginning of the execution by the `GossipBuilder` class.

Participants can join the protocol by receiving messages and replying to them. Each node which wants to participate in the protocol needs to maintain a store of all the messages it received. This store is organized with many append-only logs, one for each other participant. Thanks to these logs, data are replicated over the network and can be retrieved by any other participant at any time. The replication mechanism, togheter with the chaining of the messages

and their cryptographic signature, provides a secure environment in which messages cannot be spoofed and modifed by participants different from the creator.

Nodes communicate through a gossip protocol, by wich they exchange newly created messages and retrieves older ones, which they were not aware of or were sent before they joined the network. The update strategy consists of tho phases: in the first one a node computes the frontier of its store, collecting the pointer to the last message for each log, in the second phase it asks another participant to send to it all the messages that are beyond its frontier. In the implementation, this mechanism is realized with a 3-message exchange between participants, as represented by figure 1.

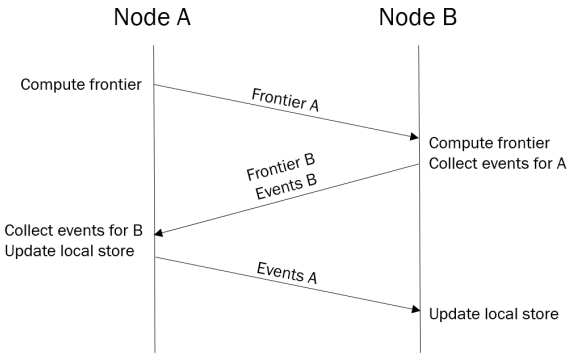

Figure 1: *Message exchange between participants in the gossip protocol, needed to update local stores.*

This exchange works for a node that only updates logs of participants it already knows. This is the case of the second protocol described in the original paper, in which each participant decides wich nodes to follow. In the case of the open gossip, instead, that aims at total replication between all nodes in the network, an additional step is required to make participants aware of peers they do not know. This is obtained with an additional message at the beginning of the exchange that contains the identifiers of known participants. With that information, nodes can create logs in their local store before computing their frontier.

As anticipated, in the second version of the protocol, each node defines a set of participants it wants to follow, and a set of participants it wants to block from being replicated in its local store. Before each exchange, it creates new logs for followed peers not present in its store, and deletes the ones he wants to block. This operation keeps the number of replicated logs under control of the single node and allows for networks of major size, as we will describe during the analysis.

In the simulator, a periodic function adds and removes nodes from the followed ones, and another call block peers with a certain probability. Nodes added to the followed set of a node are choosen from the participants currently followed by the peers it follows.

As a possible strategy to maintain the network free of undesired users, nodes can decide to block other participants when a number of other peers block them. In a real scenario this number can be choosen as a function of the trustness of the participants and with respct to the particular reasons adopted for blocking peers.

To allow for multiple evaluation scenarios, participants can also simulate offline operations. During these periods they continue to generate events locally but they do not sycnchronize with the rest of the network. As we will see from the analysis, the protocol itself allows for a fast and efficient recovery of past events from other nodes.

The underlying network is supposed to be reliable and information about participants' adressees are available to all the nodes. To simulate propagation delays in the simulator each message is assigned a delivery time, based on the distance from the source to the destination.

This function is linear with the distance and can be customized by appling a user-defined coefficient from the parameters of the simulator. Another network property simulated by the sistem is the available bandwidth, which depends on the number of messages exchanged by the network at each tick of the simulator.

To perform the analysis, data are collected both from participants agents and from a support agent, defined in the `EventAnalysis` class, which provides statistics for events generated locally by nodes (and their distribution in the network).

3 Simulator and data visualization

The advantages of using an interactive simulator, like the one provided by the Repast Simphony framework, are that it allows to customize different executions through parameters and it gives real-time feedback to the user about what is happening in the system.

The simulator used for this project, differently from other frameworks, provides a set of tools to have a better understanding of the operations of the protocol and its performance. This is mainly possible thanks to data visualization, available through representations of the agents in the space and charts, which are populated with real-time data from the running simulation.

In our case, we used these functionalities to give the user a graphical view on the network and its operations. In the main display of the simulator, called *Message network* nodes participating in the network are represented as points. Participants have different states which are mapped to different colours in the simulation:

- READY (magenta): the participant is partecipating actively to the gossip protocol
- DISCONNECTED (grey): the participant is temporarily disconnected from the network, during this time it continue to generate events locally.

Other events represented in the first display are the messages sent during the operation of the protocol. These messages, represented as directed arrows highlights the three (or four) message exchange described in the previous section.

Another display (*Social network*) represents the social relations between participants. In this graph nodes are represented togherther with arrows which indicates follow and block relations between them. These relations are randomly set at the beginning of the simulation and are periodically updated. Follow links are displayed in blue, while block pointers are displayed with red arrows. This representation is available only if the second algorithm is choosen.

Together with this graphical representation of the network, we defined also some charts in the simulator. We describe here the most relevant ones:

- Bandwidth: represents the number of messages sent for each tick in the network
- Maximum Latency: represents the maximum latency for a locally generated message to reach all the other nodes. The number of events to average on depends on the actual rate on which new events are generated, to rapidly adapt to network and configuration changes. This graph is indicative of the real-time performance of the protocol but is not an accurate representation of the actual latency. Since propagation of events is not immediate, when computing the average, not all events might have had the possibility to reach all nodes. Only a posterior analysis on the events statistics could lead to a real representation of this measure.
- Events Exchanged: represents the average and the maximum number of events exchanged within each message.

- Participant cumulative: represents the sum of generated events and sent messages for all the participants
- Total events: represents the total number of events generated by all the participants (green), the average number of events each participant has stored locally (blue) and the minimum number of events stored locally (red).

3.1. Parameters of the simulator

As anticipated, many configuration parameters of the protocol, as well as the network characteristics are configurable through the simulator.

As a way to make the simulation more realistic and understandable, we mapped each tick to a second in the real word and expressed parameters of the protocol with this unit.

Many of them, out of the ones used at the start-up of the simulation (marked with *s*), are modifiable by the user at run time. This allows users to see changes in performance without the need for running multiple simulations but directly tuning the running one. This is very useful during the evaluation phase and enables a fully configurable run.

They are described here in detail, for ease of use:

- Algorithm: the algorithm to use during the simulation (1 - Open gossip, 2 - Transitive-Interest gossip) - *s*
- Bandwidth (msg/sec): the upper limit to the network bandwidth, expressed as the number of messages accepted by the network at each second
- Block factor: number of participants needed to transitively block another peer
- Block probability: the probability by which a participant block another node
- Disconnection probability: the probability by which a participant enter offline operation
- Disconnection time: the maximum time by which a disconnected node needs to connect again
- Event probability: the probability by which a participnt generates a local event
- Initial follow: the number of participat to follow at the beginning of the simulation - *s*
- Neighbors: the size of the set of neighbors a participant can connect with during the operation of the protocol (0 stands for connect to anyone). Low values enables local connections, while higher values implies connections with distant nodes.
- Network Delay Coefficient: the coefficient for the function that computes the time needed for a message to reach its destination
- Participants: the number of participants in the network - *s*
- Update followed frequency (sec): the fequency by which each participant updates its followed set
- Update frequency (sec): the frequency by which each participant starts a new message exchange with another node to update its local store.

4 Analysis

Data collection required for the analysis was performed with the tools offered by the Repast Simphony simulator. Charts presented in this section were generated with a python script to organize data and with the help of the matplotlib library to plot figures.

4.1. Open and Transitive-Interest Gossip Models

The first analysis performed with the simulator was to figure out the main differencies between the two protocols described in the original paper. Our interest was about some major metrics which describe the most important characteristics of the two algorithms. These metrics were choosen because of their relevance for a possible application in a real environment.

The parameters used to run the two experimets were the same, except for the algorithm choice. In the netwrok were present 100 nodes and each of them followed 5 nodes in the second algorithm.

Results are reported in table 1 and discussed in the next paragraph.

Table 1: *Differences between Open and Transitive-Interest gossip models*

Metric	Open gossip	T-I gossip
Generated events	24774	25693
Average store size	23554	1324
Avg Delivery Rate	1	0,05
Avg Latency	243	731
Max Latency	714	4417
Avg Events exchanged	107	2,7

Strating from the number of locally generated events, we can immediatly observe that is very similar form the two approaches. This is straightforward because it depends only on the probability by which events are generated. Differences start when we analyse the number of events stored locally at each node. This is the major difference between the two protocols and have a great impact on the storage at each node and, therfore, on the number of participants the protocol can accomodate. While in the first case all the messages are stored locally, without distinctions between participants, in the second case only events generated by followed peers are stored, allowing each node to choose what to save and what to ingore. Since in the simulation each participant followed 5% of the nodes it is reasonable to find that the store size correspond to $\sim 5\%$ of the generated messages on average.

The previous reasoning is confirmed by the average delivery rate of events. Again, we find that on average, each event is stored in 5% of the participants' logs when the follow approach is taken into consideration.

A more interesting result, was the one related to the average and maximum latency. We first try to explain better the intended meaning of these two metrics: the average latency is the average time needed by a local event to reach another node. The maximum latency is the worst case time needed for an event to reach all the interested nodes. Both these metrics are quite higher for the second algorithm with respect to the first. We can find the explanation for this difference in the behaviour of the gossip protocol. Since nodes exchange information with random peers in the network and, for the transitive-interest model, they store only some events, the probability to find the right updates when contacting another participant is lower in the second case. This means that, due to a lower rate of replication in the network, the time needed to find information is greater when no information on its location is known.

Finally, the last metric measure how many events are exchanged on average inside each message. This, due to the selection on the information done by participants, is significantly lower in the second version of the protocol. We can also take these values as an insteresting metric for the network load, which indicates clearly that the traffic in the second case is much less.

4.2. Recovery of past events

Analysing the benefits of the implemented gossip protocols we can surely take into consideration the high grade of replication of the system. To measure this aspect we show one relevant example and then generalize to different situations.

The expertiment consists in letting the protocol executing for a period of time and then disconnecting only one node to see how fast is the recovery of past events. To analize results we report here a chart 3 which plots three different counts for events: *generated*, that is the sum of locally generated events, *received avg*, that reports the average number of events stored by participants, and *received min*, which tracks the same metric in the case of the store with the smaller number of events.

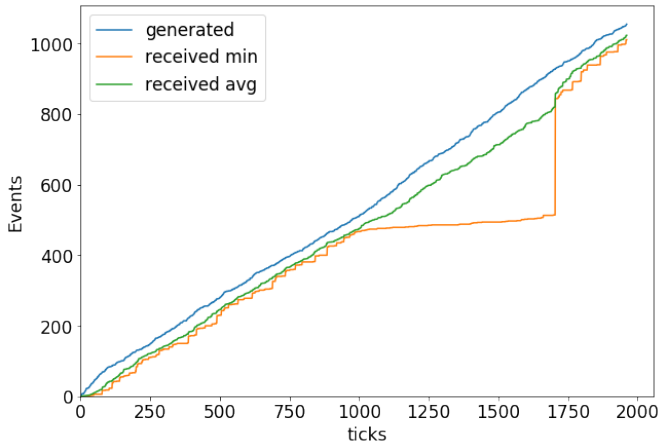

Figure 2: *Number of events in the case of a node with a period of offline operations*

Reading the graph, we can analyse the orange plot, which gives us information on the size of the smaller store between participants. While at the beginning this is very near to the mean value for this metric, around tick 1000 it diverges significantly, indicating the disconnection of a node. At this point, in fact, the selected participant stops to update its local store and to reply to other participants. Anyway, it continue to generate events locally, and we can confirm this behavior by noticing that the plot is not stricly orizontal but increases with the passing of time. The node stays disconnected for approximately 750 ticks, during which in the network ~ 300 new events were generated. At around tick 1750, the orange line does a quick jump towards the average value and starts back to follow it. From this point, the node is again connected and the gossip protocol continues its operations normally.

This behaviour can be explained analysing the characteristics of the gossip protocol, and the properties of the Open model in particular, under which this experiments was run. From the Open model, we know it aims at total replication between all participants in the network. From the gossip protocol, we know that, as soon as the nodes enters again the online mode, it will connect to a random node in the netwok to try to update its store. Given these two factors, we can conclude that the recovery of past events, is in this case very fast due to the

high probability of finding almost all missing events at the first peer contacted.

This aspect is highlighted by the graph 3 which plots the maximum number of events exchanged within a single message. At tick 1750 a peak in the chart confirm our expectations.

This recovery mechanism is therefore very efficient and does not require any additional layer on top of the implemented gossip prototcol. Moreover, it is very fast and scalable to a great number of nodes and long periods of offline operations.

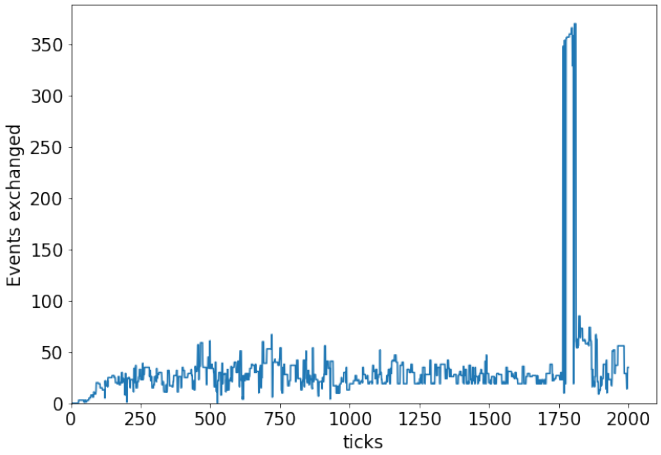

Figure 3: *Number of events exchanged for each message*

4.3. Frequency of updates and latency

Let's now focus on the frequency by which the protocol updates local sotres and try to understand which consequences it brings in terms of latency. To analyse this correlation, three different executions of the simulator were run with the same parameters except for the frequency of updates. Latency distributions of the three different executions were then reported in one single chart to compare them. The resulting graph is shown below 4 and gives evidence of the protocol behavior.

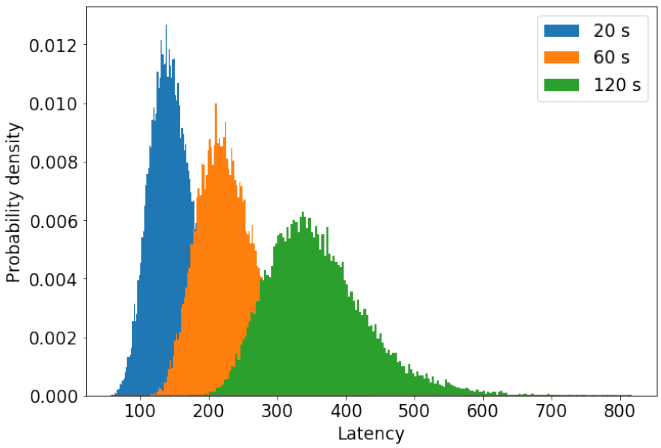

Figure 4: *Latency distributions for different update frequencies*

Table 2: *Messages exchanged for different update frequecies*

Update freq.	Genrated messages	Messages exchanged	Events/message
20 s	25.000	100.000	43
60 s	25.000	33.000	68
120 s	25.000	16.500	125

To understand better the impact of these choices on the network we can look also at table 2, where some statistics on messages and events exchanged in the network are reported.

Given these data, we can observe that with a reduction of the frequency of update the relative latency and the number of events exchanged within each message increases linearly by a factor smaller than 1. At the same time, the number of messages needed to keep the stores updated decreases by the same factor of the frequency. Moreover, as visible from the graph, reducing the update frequency has an impact both on the average latency and on the standard deviation of this measure. These considerations can be taken into account when designing an application which uses this protocol, to minimize the overhead generated by a huge number of messages while keeping the latency under an accepted value.

4.4. Impact of network topologies

The implementation provided with the simulator allows to change same parameters in the way nodes communicate between each others and in the network behaviour. Let's explain these two parameters and try to see their impact on the protocol execution.

The first parameter allows to choose how participants decide to what peers they want to connect to update their store. The choice is between choosing a node among all the possible ones, or selecting it among a set of neighbours of configurable size. In the second case, connections between participants become local and the variety of nodes they connect to decrease with respect to the first option. The difference between the two topologies is clearly visible in the simulator as reported in figure 5: these settings were obtained with a network of 50 nodes and a set of 4 neighbours in the second picture.

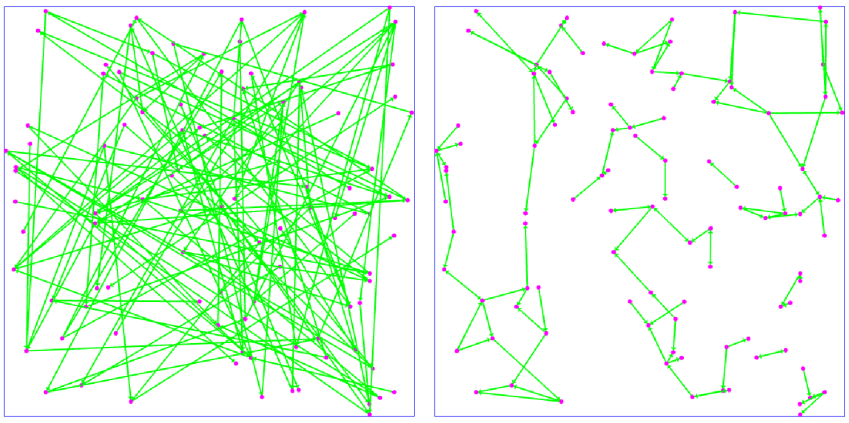

Figure 5: *Difference between global and local connections*

The second parameter set the delay caused by the network transmissions. As anticipated in the previous sections, network delay is a function of the distance between nodes and can be multiplied by a factor provided by the user. This value can be therfore appropriately choosen

as a function of the type of network used and the effective distance between participants.

The two graphs in figure 6 show respectively the distribution of the latency and of the distribution of the message size for two experiments, underling the difference between a global and a local topology.

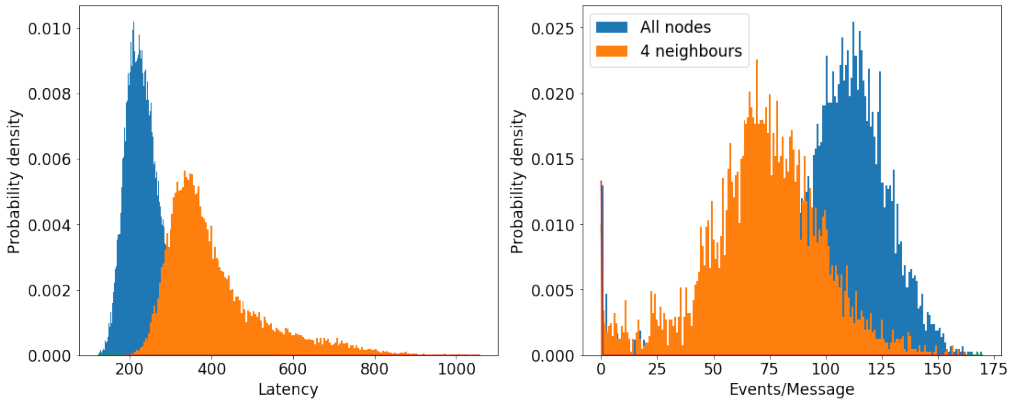

Figure 6: *Difference between global and local connections*

From these charts we understand that, under the assumptions by which the two experimet were run, having a local topology increases the average latency. This may be conterintuitive in a first place but some aspects must be considered here.

In the first place, the influence of netweork delays in the two execution is irrelevant: the update frequency is set to 40 seconds and the coefficient for the delay is set to 0.5 (which implies a maximum latency of 25 seconds): such values implies that the latency is dominated by the update frequency and local and global connections have no impact on it.

Second point, connecting only to a few nodes, rahter than choosing randomly between all the participants, increases the number of steps needed to propagate informations throught the network. This aspect is confirmed by the second chart, from which is evident that, in the case of local connections, each message contains on average less information because many messages exchange only local events.

Considering these aspects, can lead to better considerations when deciding to implement the analysed protocol in a real deployment and the simulator can surely help to find the better configuration given some environmental parameters.

5 Conclusions

The analysis carried out on the protocol and described in the previous section led us to some conclusions that we will briefly report here. The first and most important point is that the analysis was very useful to correctly understand and implement the protocol proposed in [1]. In the original paper, some aspects of the implementation were not detailed and no performance analysis was performed. Moreover, tring to understand the behaviour of the protocol during different simulations helped to provide a more complete implementation.

Second, from the analysis we can have some hints on which are the pros of the solution proposed in [1]. The main strength of this protocol is certainly the efficient way by which information is replicated throught the network. As measured, the latency to cover all the network in the average case is low with respect to the update frequency and the presence of the information in all the nodes allows for a quick recovery in the case of disconnected periods.

These considerations done, toghether with the simplicity of the algorithm implementation, this protocol well suits for a sistem with robustness as a priority requirment.

6 Installation guide

In order to run the simulator, ensure version 11 of Java RE is installed on your system, then follow this steps:

1. Download the installer `ssbgossip.jar` from Google Drive [1]
2. Run the installer by clicking on it or typing `java -jar setup ssbgossip.jar` in the terminal
3. Follow the proposed steps for the installation procedure
4. Launch the simulator by executing the `start_model.bat` or `start_model.command` file depending on your system
5. From the simulator, you can set each parameter in the dedicated tab
6. Run the simulator by clicking "Start Run"

References

[1] Anne-Marie Kermarrec, Erick Lavoie, and Christian Tschudin. (2020). Gossiping with Append-Only Logs in Secure-Scuttlebutt. In 1st International Workshop on Distributed Infrastructure for Common Good (DICG'20), December 7–11, 2020, Delft, Netherlands. ACM, New York, NY, USA, 6 pages. 10.1145/3428662.3428794.

[1] `https://drive.google.com/file/d/17WNvCdHn8mhOSpr6284Th_k81dirc33S/view?usp=sharing`