A Broadcast-Only Communication Model Based on Replicated Append-Only Logs

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 protocol described in the paper *A Broadcast-Only Communication Model Based on Replicated Append-Only Logs* [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 protocol graphically and to analyse its 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 versions of the protocol and the effects of particular choices in its configuration and in the network behaviour.

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 presented follows the steps of the original paper and adds some details which were not addressed there. A detailed description of the differences will be provided while presenting the analysis of the protocol which tries to point out some shortcomings from the original work and proposes some enhancements to it. Besides this, a more general reasoning is carried out to evaluate the benefits and limitations of this type of communication model based on broadcast and a replicated data structure.

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 `Relay` agent, which can receive and broadcast messages to other relays (relays are also addressed as *nodes* of the system). Each of these relays 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.

Relays are created at the beginning of the execution by the `BcastOnlyBuilder` class. This class is also responsible to delete them and create new ones in the case of a dynamic network environment where nodes can leave and join at any moment.

As anticipated, relays have two basic primitives: receive and broadcast. To understand these primitives is essential to introduce the network assumptions on which the system is

based. The broadcast-only communication model is an overlay network built on top of a physical network. The choice was to assume an underlying wireless network which fits quite well with the protocol behaviour, mainly because of its broadcast nature. In this model, nodes can broadcast messages to their neighbours in all directions in a given range (called *broadcast domain* in the next sections). This behaviour is implemented in the simulator using the `Space` and `Grid` objects which allows to position nodes in the space and compute their distance.

Relays are nodes of this network and can broadcast messages and can receive transmissions from their neighbours if they are within their broadcast domain. Relays broadcast messages by putting them in the receiving queue of all their neighbours. Once a relay receives a message, popping it from its receiving queue, it handles it based on three different versions of the protocol, based on a configuration parameter (ReceiveI, ReceiveII, ReceiveIII). All these versions of the protocol end with a broadcast of the message to propagate the information to the node neighbours. This behaviour generates the perturbation defined in the original paper. In the next sections, we will refer to a perturbation both as the actual perturbation of a single message to all the nodes in the network and as a single broadcast event from a node, following the definitions of local and global waves given by the author of [1].

As described in the original paper, relays offer other functionalities such as point-to-point messages, multicast messages, private messages and recovery of lost perturbations, all based only on the two primitives described before.

To allow for multiple evaluation scenarios, nodes, other than joining and leaving the network, can also simulate a partial crash or a temporal disconnection. Similarly, messages in the network can be lost or delayed by saturation in the available bandwidth. This aspect is implemented, arguably, as a global counter which refers to all the network. This is due to the difficulty to envision a simple way to represent local bandwidth for a wireless medium and by the assumption that, on average, the number of messages in the network is equally distributed. Relays keep a sending queue to tolerate these delays.

The architecture proposed allows respecting all the three properties stated in [1] for the broadcast model:

1. Each perturbation source has a globally unique identifier that is carried with each perturbation it triggers ⇒ Each relay is identified by its hash code, which is unique, based on the java implementation.

2. A perturbation and its value eventually reaches all anonymous observers ⇒ Guaranteed by the propagation of the message by all the relays in the network.

3. All observers sense subsequent perturbations coming from a specific source in the same order ⇒ Messages are delivered to other nodes through their receiving queue, which is intrinsically FIFO.

To perform the analysis, data are collected both from relays and from a support agent, defined in the `PerturbationAnalysis` class, which provides statistics for perturbations (intended as global propagations of messages to all 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, relays participating in the network are represented as points. Relays have different states which are mapped to different colours in the simulation:

- QUITE (blue): the relay is ready to send or receive messages but it is not currently working
- SENDER (red): the relay has started a global perturbation
- RELAY (green): the relay has received a message and it is forwarding it to its neighbours, acting as a relay
- ARQ_REQUEST (cyan): the relay has received an ARQ request, this message is not forwarded to neighbours
- DISCONNECTED (grey): the relay is temporarily disconnected from the network, during this time it doesn't perform any operation

Through the representation of the state of the relays in the network, it is possible to identify the propagation of the messages as waves, originating from the sending node (red) and propagating to other nodes which become green as they receive and forward the message.

Other events represented with graphical elements are the messages sent to specific destinations. This type of messages, just to recall from the previous discussion, is delivered exactly by the same mechanism described for global messages. Arrows represented in the network are simply a way to differentiate this type of deliveries from the ones that always happen when a node forwards a perturbation. Different types of messages are represented with different colours of the arrow:

- Point-to-point messages (black): sent from a source to a specific node using its unique identifier
- Multicast messages (orange): sent from a source to a group of nodes identified by a shared group id
- Private messages (magenta): sent from a source to a specific node using its public key to encrypt the message

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

- Delivery rate: represents the mean delivery rate of the last perturbations. The number of perturbations to average on depends on the actual rate on which new perturbations 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 delivery rate. Since propagation of perturbation is not immediate, when computing the average, not all perturbations might have had the possibility to complete (they eventually complete). Only a posterior analysis on the perturbation statistics could lead to a real representation of this measure. Also in the case of a dynamic network (total number of relays is instable), this plot is only indicative since it is impossible to determine how many nodes should have received the perturbation.

- Latency: represents the maximum latency of the last perturbations. Same considerations done for the previous chart applies here.
- Relays number: represents the current number of relays in the network
- Perturbations: represents the maximum and the minimum number of perturbations received at different nodes. This chart highlights delays in the delivery due to propagation, disconnections, lost messages or joining nodes.
- Relays load: represents the average load of each relay measured as the number of send and receive events
- Sent messages: represents the total amount of messages sent in the network
- Used Bandwidth: represents the number of messages sent for each tick in the network and the size of the out queue of the nodes

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.

All of them, out of the relay count, which is only used at the start-up of the network, 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:

- Bandwidth: the total amount of messages that the network can dispatch at each tick
- Broadcast domain: the range of a single broadcast transmission. This measure is relative to the dimension of the space.
- Increase and Decrease Factor: the number of nodes which are added and removed every 10 ticks
- Disconnection probability: the probability for each node to become disconnected at each tick
- Max Disconnection time: the maximum number of ticks for which a node remains disconnected
- Loss probability: the probability for a single message to be discarded by the network
- Moving factor: the distance by which each node can move at each tick
- Relay count: the initial number of nodes of the network
- Relay type: select the version of the protocol to use during the simulation as described in the original paper
- Send Probability: the probability for a node to start a perturbation.
- Send P2P Probability: the probability for a node to send a point-to-point message.
- Send P2M Probability: the probability for a node to send a multicast message.
- Send P2S Probability: the probability for a node to send a private message
- Workload: the available workload for each relay measured as the number of send and receive events for each tick

4 Analysis

We start the analysis by following the steps described in the original paper from Relay I to Relay III. While implementing the protocol, we realized that some minor aspects were not

considered in the description provided by the author and we tried to address them with possible solutions. Then we will address the protocol from a more general point of view, trying to find its strengths and weaknesses.

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 the help of the matplotlib library to plot figures.

4.1. From Relay I to Relay III

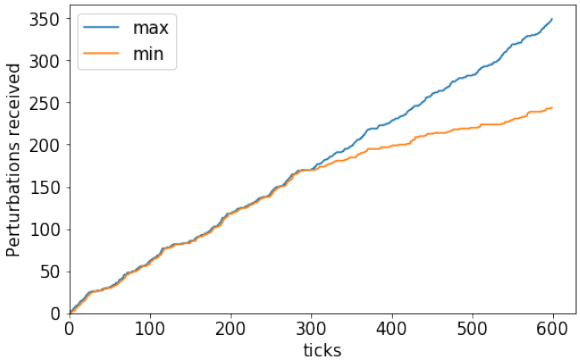

Figure 1: *Minumum and maximum number of perturbations received by different nodes in the case of Relay I and network losses*

Starting from Relay I, the protocol behaves well under the conditions described in the paper of a static network and constant propagation times. As expected, when we introduce some dynamicity in the network, both by moving relays or by adding some loss probability, nodes start to lose perturbations and are no more capable of accepting new perturbations from the same source. This behaviour is clear in figure 1, where we plot the maximum and the minimum number of perturbation received by different relays. As soon as perturbations start to be lost the two lines become divergent and no action can bring them to converge again.

Relay II is presented to be capable to work in a dynamic network case, with the addition of nodes and data mules with intermittent connectivity but two considerations must be done:

1. To allow for new nodes to join the network we have to initialize their frontier to the first perturbation they receive from each source, otherwise, they will not be able to accept new perturbations, as they will wait for older ones that will never arrive.

2. The problem of lost messages shown for Relay I is only partially resolved. For a static network with node addition, the bag structure presented in the paper could only lead to a small decrease in latency, since perturbations are not lost in this case. In the presence of data mules or moving nodes, the bag structure can help only in some particular situations, difficult to reproduce in the simulation.

An interesting improvement is carried by Relay III, which provides a mechanism to recover from lost perturbations. As shown by figure 2 and 3, which can be compared to figure 1, nodes with lost perturbations are able to come back to normal operation.

This mechanism also allows new nodes to join in the middle of the execution and acquire knowledge about all the previous perturbations. This is possible, not only because of the retransmission requests but also by the replicated nature of the protocol. But, how do this two factors affect performances?

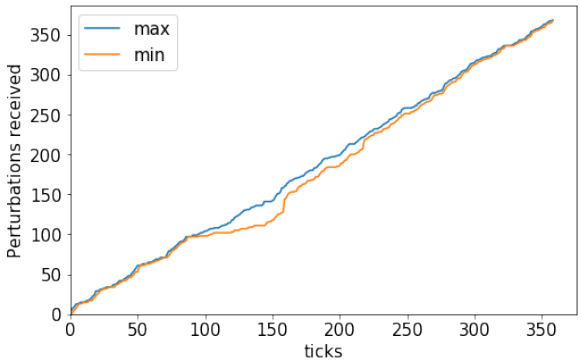

Figure 2: *Minumum and maximum number of perturbations received by different nodes in the case of Relay III and network losses*

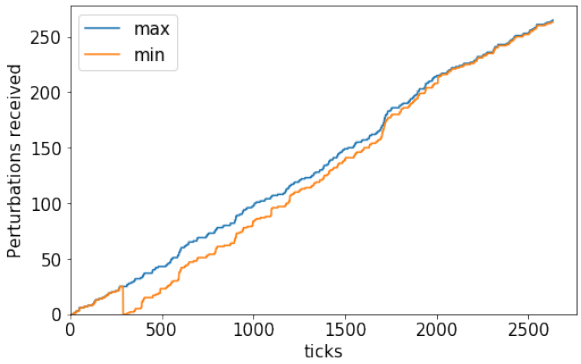

Figure 3: *Minumum and maximum number of perturbations received by different nodes in the case of Relay III and new node joining the netwrok*

We will firstly show how much the retransmission mechanism impacts the network load, and secondly how the broadcast nature is quite heavy in terms of sent messages.

4.2. Impact of ARQ mechanism

To show the first point, we run the same simulation two times with the same parameters. The only difference is the presence of the retransmission mechanism in the second run.

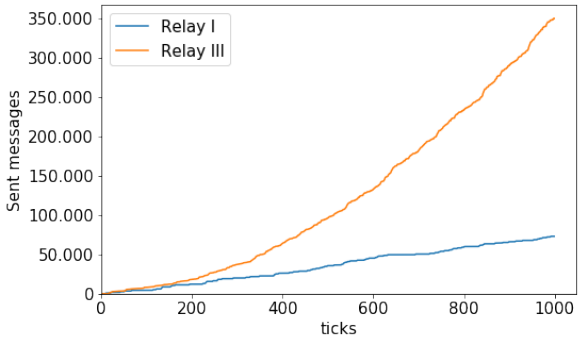

Figure 4: *Number of messages sent with and without the ARQ mechanism*

The comparison, shown in figure 4, clearly show an increment in the network traffic of 4 times. This includes traffic from ARQ requests (which are not propagated through the network) and ARQ replies that are broadcasted over the network. In this example, we find another point of the protocol which deserve to be discussed: even during normal operations (absence of losses) many "useless" messages are generated. The first problem is related to ARQ requests: nodes generate them periodically, even if they have not discovered any lost perturbation. This could be useful in a setting where low latency is needed and messages from the same source are not frequent but, in a more relaxed scenario, this could lead to a high increase in the network load. Second, also ARQ replies are generated, even if the network is working well. This is because of delays in the transmissions of messages, which lead to redundant retransmission (when the ARQ reply arrives, the original perturbation already reached the sender of the ARQ request).

4.3. Broadcast and network load

Let's now focus on the broadcast nature of the protocol and try to understand which consequences it brings in terms of network load. As before, we measure the load as the number of messages exchanged in the network in a fixed amount of time. We start by identifying factors which influence the number of messages in the network. First, is the number of perturbations sent across the network, second, the broadcast domain, third, the number of nodes and finally, as already described in the previous paragraph, additional layers of the protocol, such as the AQR mechanism. To understand how each of these aspects affects network load, we run multiple simulations, addressing one aspect at a time.

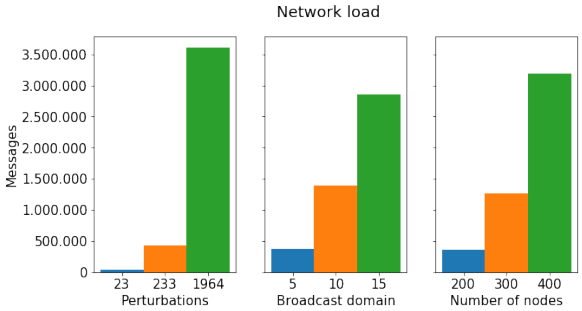

Figure 5: *Network load for different parameter values*

Figure 5 shows how different values of the first 3 factors relates to network load. From these values, we could observe that network load is: linear to the number of perturbations, quadratic to the broadcast domain and exponential to the number of nodes.

4.4. Delivery rate and latency

We will now focus on the two main aspects which measure the actual performance of the protocol: delivery rate and latency. As done for the network load in the previous section, we will analyse these metrics for different executions of the protocol in which we change some parameters to show their influence on them.

The parameters we are interested in, for this evaluation, are: the broadcast domain, the available bandwidth, network mobility and loss probability. Other factors, will for sure affects delivery rate and latency, such as the number of messages and number of nodes, but only because they correlate with the network load and therefore with available bandwidth. The same does not apply to broadcast domain, because it directly affects the number of retransmissions

required for a perturbation to propagate and the number of nodes in the network which are reachable or isolated.

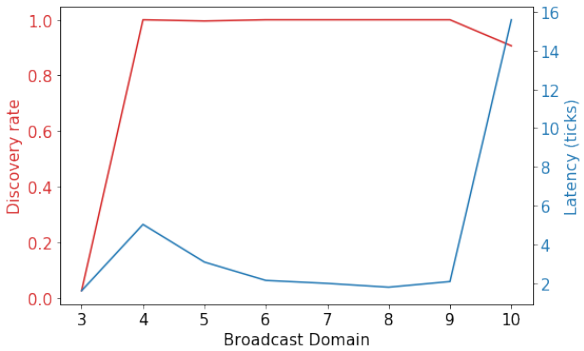

Figure 6: *Performances of the protocol depending on broadcast domain*

Starting from the broadcast domain, in a network of 200 nodes, we can observe (figure 6) that, with a value of 3, the delivery rate is near to 0: this is because nodes cannot receive messages from their neighbours and the network is disconnected. As the broadcast domain starts to grow, the network quickly becomes connected and delivery rate reaches 100%. If we continue to increase the broadcast domain we have a significant reduction in latency, since a smaller amount of retransmissions is needed to reach all the nodes. At some point, however, this reduction stops because another factor comes into play: as the transmission domain increases, the number of messages needed by the transmission protocol does the same, and the bandwidth is quickly saturated. This brings again to higher latency until the bandwidth is not enough to proceed with the protocol.

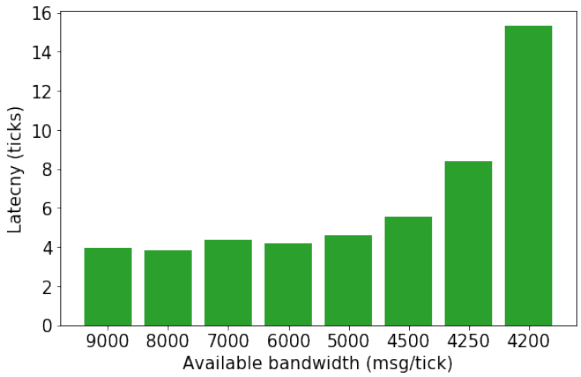

Figure 7: *Performances of the protocol depending on available bandwidth*

This observation leads us to the next chart (figure 7) where we can analyse in details the behaviour of the protocol with respect to the available bandwidth. For high values of the bandwidth, the protocol generates a constant latency due to the number of retransmissions needed to spread the information across all the network. As the bandwidth gets reduced, nodes have to wait before transmitting messages, putting them in their out queue. This aspect leads to an increment in the bandwidth which is sustainable until the average growth factor of the queue is less than its average decreasing factor. In the analysis, as reported in the graph, we reached this point for a bandwidth of $\sim$ 4200 msg/tick in a network with 200 nodes, a broadcast domain of 5 and a send probability of 0,001.

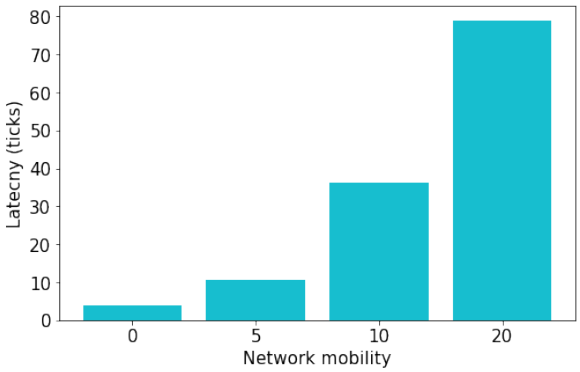

Figure 8: *Performances of the protocol depending on network mobility*

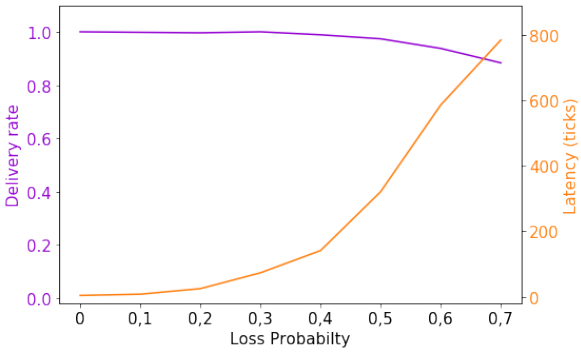

Figure 9: *Performances of the protocol depending on network losses*

The last two graphs (figure 8 and figure9) are in some way related between them. Indeed, in both cases, the increase in the latency is linked to a growth in the number of lost perturbations. Starting from the network mobility, as we increase the space by which nodes can move, we increment the probability of losing some perturbations. The same applies when we increment the probability of a loss of a message during its transmission. The growth of the latency is therefore caused by the ARQ mechanism, that permits to reach anyway a delivery rate of 100% in most of the cases. Since ARQ requests are broadcasted only at fixed intervals and they are not always fulfilled, because neighbouring nodes might not have the required information too, this process takes time and brings to an increase in the maximum latency of lost perturbations.

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 analysis was performed. 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 and the cons of the solution proposed in [1]. The main strength of this protocol is certainly the fast replication of information in the network. As measured, the latency to cover all the network in the

average case is very low and the presence of the information in all the nodes allows for a quick recovery in the case of losses. Moreover, the protocol needs not to maintain any global or local information, such as routing tables or neighbour lists. This anyway has a negative impact on the performance of the network in terms of bandwidth consumptions.

These considerations done, we have some elements to envision when choosing to deploy this protocol. On one hand, we must carefully choose the parameters based on environmental conditions, such as network topology, number of nodes, required latency and expected loss probability. On the other hand, we have to consider the huge network load generated by the retransmissions: maybe this protocol is not appropriate for a low-power wireless setting such as the one of an IoT sensor network but can be more suitable for a wired network which needs to have a high grade of replication with a simple and light implementation and fast operations.

6 Installation guide

In order to run the simulator, ensure at least version 1.8 of Java RE is installed on your system, then follow this steps:

1. Download the installer `bcastonly.jar` from Google Drive [1]
2. Run the installer by clicking on it or typing `java -jar setup pbcast.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] Christian F. Tschudin (2019). A Broadcast-Only Communication Model Based on Replicated Append-Only Logs. ACM SIGCOMM Computer Communication Review. 49. 37-43. 10.1145/3336937.3336943.

[1] `https://drive.google.com/file/d/117PpMyic36N0hocu2E7-09TsL0-yZvHj/view?usp=sharing`