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

Conti A. (215034), Tonini F. (211961)

November 30, 2020

Abstract

In the last decade, decentralized technologies have become increasingly popular. From privacy oriented to highly scalable solutions, decentralization is key to the scalability of modern technologies.

This paper provides an implementation of a broadcast-only communication model based on replicated append-only logs. Such solution lies at the foundation of many decentralized applications where networking with arbitrary data packets is replaced by coherent data structures. Moreover, an extensive analysis of the protocol is carried out to evaluate its overall performance in various settings. The primary evaluations are about the variation in bandwidth, in topology structures and in many aspects of the perturbation instances.

Our contributions are the following: (i) we start by describing the properties of the abstraction as described in [1] and implemented in our solution. Then, (ii) we present our simulation results with a study of the collected metrics.

1 Introduction

A broadcast-only communication model shows many similarities with the real world. At the basis of this approach, propagation of perturbations carries information through space. Moreover, a broadcast-only solution does not provide a definition of destination like the one traditionally available in unidirectional approaches. Instead, perturbations propagate infinitely far if not blocked and cover every observers listening. The broadcast model has three main properties:

1. Each perturbation source has a globally unique identifier that is carried with each perturbation.
2. A perturbation and its value eventually reaches all anonymous observers.
3. All observers sense subsequent perturbations coming from a specific source in order.

The implementation of the model presented is built around the Repast Simphony framework and written in Java. Such environment allows to design and evaluate the performance of the application through the implementation of agents that participate in the protocol. A wide set of variables are available to the user so to allow easy testing of a particular simulation in terms of network bandwidth, number of participants, topology, and input load. Repast let the user collect statistics about the simulations which are then processed through a set of Python scripts built specifically for this purpose.

The rest of the paper is structured as follow: in chapter 2 an introduction on previous works on the subject are presented, while chapter 3 focuses on the Repast implementation with details on the structure and options of the simulation environment. Then, chapter 4 describes tests and results of the different scenarios that have been tested. To conclude, in chapter 5 some future works and conclusions are drawn.

2 Related works

The work presented by [1] talks about the usage of solitons as a medium of information propagation that does not leave any disturbance behind. Solitons are known in many fields of science, i.e. as biology and cosmology, and can be applied to communications in the context of fiber optics.

Each solitary wave represents a local broadcast where the message is propagated and sensed by nearby observers in all directions. When a soliton is received by a relay, it is propagated once more, so that the set of small perturbations creates a large one that reaches all of the agents in the network. In different terms, it can be stated that a small perturbation is equal to a local broadcast, and a sum of local broadcasts compose a global one.

In a context where no message is lost and the set of observers is static, the forwarding policy can be trivially implemented by making sure that duplicated messages are not forwarded. This approach makes

so to avoid packets travelling forever. Still, if a message is sensed for the first time, it is clearly forwarded so that to eventually complete the global broadcast.

On the other hand, if the network is dynamic, the solution proposed in [1] consists in adding a data structure that stores the received out-of-order messages. This bag get filled with packets that the relay is not expecting from a source, and get drained whenever it receives the missing perturbations. In regards of lost messages, the option is to implement an Automatic Retransmission reQuest mechanism. With this protocol, each observer periodically announce their next expected packets. In the case in which a nearby relay owns a message contained in an ARQ request, it propagates it.

3 Implementation

To implement the broadcast protocol, the decision fell to the Repast Simphony framework and the Java language. Repast provides many facilities to which build a dissemination protocol, while Java is a programming language encountered in previous works by the authors. The combination of these two technologies provides reliable and easy to use solution to build agent-based environments. Repast offers both simulation and evaluation tools such as graphical representation of agents and common statistical measures and graphs, so to easily evaluate each simulation and test different iterations of the implemented protocol.

3.1 Environment definition

The code is implemented inside a 2D space so that it would make the simulation lighter without sacrificing the outcome of the tests. The 2D space is divided into both a grid and a continuous space. While the continuous space makes it possible to create many observers so to test environments of high density, the grid allows to easily calculate relative and absolute positions of each agent without sacrificing performance.

The framework introduces the definition of "tick" as the atomic unit of time, and a simulation increases the tick so to represent time passing. The presented solution makes use of the tick to schedule each relay's intention to send. Each observer decides whether to send a message every multiple of 200 ticks, while the ARQ mechanism is triggered at every multiple of 300 ticks. When a relay wants to send a message, it generates a payload that contains it alongside other fields to identify each payload during the whole lifecycle of the simulation. Each payload contains:

- Source ID: identifies the creator of the payload.
- Reference ID: per observer unique identifier of the payload.
- Value: value to propagate.
- ARQ flag: identifies ARQ requests from other types of messages.

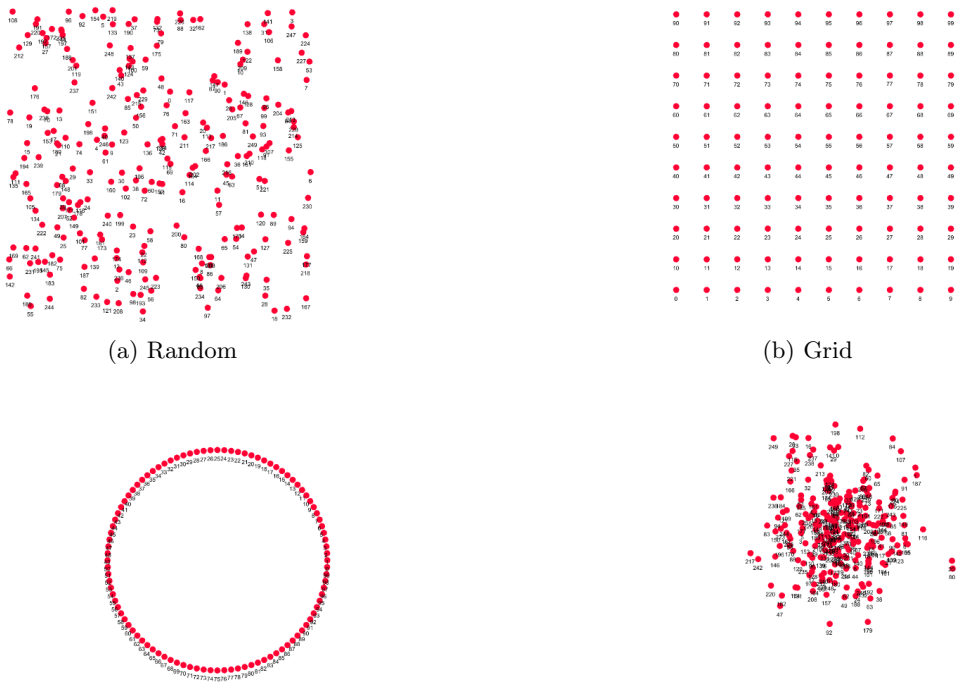
The payload gets embedded into a perturbation object that travels through the 2D space. The perturbation initial size is equivalent to the size of its creator and increases its radius by a constant value until its time-to-live (TTL) drops to 0. At this point the perturbation is removed from the simulation. The TTL is updated at every tick and it is calculated as the product between twice the perturbation radius and the size of the observer divided by the speed of the perturbation. After each tick, each ongoing perturbation queries the nearby cells of the space to find observers. If any relay is found, the perturbation delivers the payload, as to simulate the "sensing" of the perturbation.

When an observer receives a valid payload for the first time, it appends it on its private log and then forward it embedded in a new perturbation. If an incoming payload has already been processed, the observer drops it without forwarding it.

3.2 Agents

The proposed solution considers 3 agents that interact and share information during a simulation. Two different perturbations objects are defined, one representing a normal broadcast, another used for the ARQ protocol. The primary reason for a separation of this kind is to visualize the perturbations in two different views inside the GUI of Repast. A normal perturbation is responsible for the dissemination of the payload throughout the space. To to reduce wireless overhead a mechanism was implemented to reduce the amount of retransmission request without any performance degradation.

An observer actively participate in the broadcast by generating payloads that are then embedded in a perturbation that propagates in the space. Also, a relay can receive payloads from neighbors and react accordingly. Each observer can drop a payload if it has been already processed; if it receives an ARQ request, it answers with a new forward. Moreover, any relay periodically emits ARQ requests so that to make sure that no payload is lost. Finally, each observer collects a set of statistics such as perturbations' delay, number of payloads created, forwarded and number of ARQ requests that have been fulfilled.



(a) Random (b) Grid

(c) Circle (d) Normal

Figure 1: A visual representation of the implemented topologies.

To simulate a more realistic scenario, each observer decides whether to send a message based on a predefined probability. This also applies to the sense and forward of a perturbation: an observer has the ability to drop a forward and ask for a retransmission later in the simulation thanks to the ARQ mechanism available. This drop chance is introduced to simulate faulty channels. In each simulation, it is assumed that each relay is reachable by perturbation and the network is completely connected.

3.3 Topology

To simulate different scenarios and better understand the performance of the protocol, different topologies were implemented to distribute relays in specific coordinates of the space. The presented topologies are listed below:

- Grid: relays are equally distributed in a grid fashion.
- Circle: observers are equally positioned in a circle.
- Normal: relays are distributed from the center of the space following a normal distribution.
- Random: observers are allocated in the space through a uniform distribution.

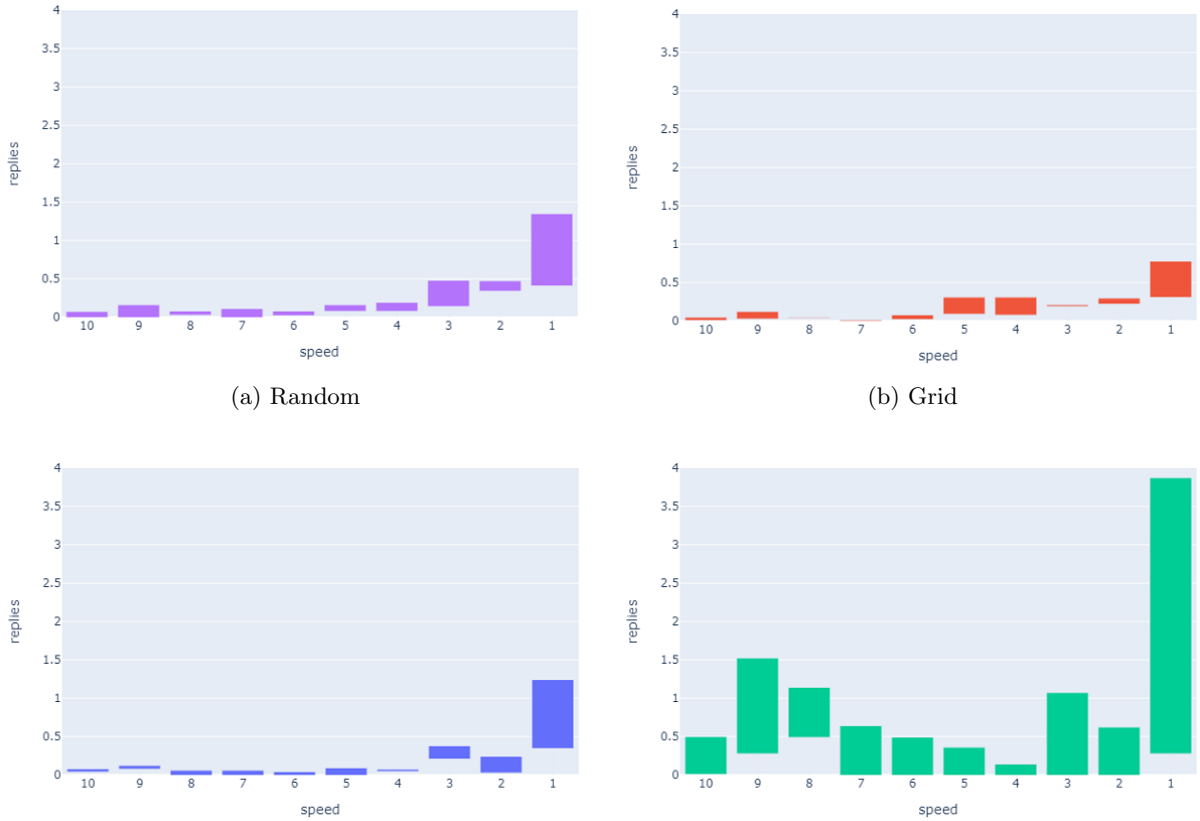
3.4 Application-level protocols

While the protocol implements broadcast by design, application-level implementation like point-to-point, secure point-to-point and pub/sub are feasible. The presented code implements point-to-point as an alternative communication solution to broadcast. While a broadcast payload does not have a destination ID (i.e. the identifier of the relay who should receive the message), a point-to-point payload includes such field. When an observer sends a point-to-point payload, each relay whose receives the message will both append it to the log and check if the destination identifier matches its internal ID. If that is the case, the observer delivers the message to the higher-level protocol who is using point-to-point. The same principle can be applied to pub/sub and secure communication models.

3.5 Simulation parameters

Many of the variables that defines each simulation are user-configurable without modifying and recompiling the model. Such parameters change key aspects to the simulation such as:

- Grid size: side of the space; the area is evaluated as (grid size)x(grid size). The space is represented by a 2D square and its size never changes during simulation.



(a) Random

(b) Grid

(c) Circle

(d) Normal

Figure 2: Average number of ARQ replies with different propagation speeds. The columns represents the ranges of values retrieved during the multiple simulations. A narrower column indicates a lower deviation in the results.

- Send probability: the likelihood that an observer sends a payload when triggered.
- Drop probability: the chance that a payload gets discarded by a relay.
- Observers count: the number of observers that populates the space.
- Topology: defines how to position each observer in the space.
- Communication model: defines the type of communication of the simulation.
- Perturbation radius and speed: defines the speed and radius of each perturbation (those two values will be part of the TTL calculation of the perturbation).

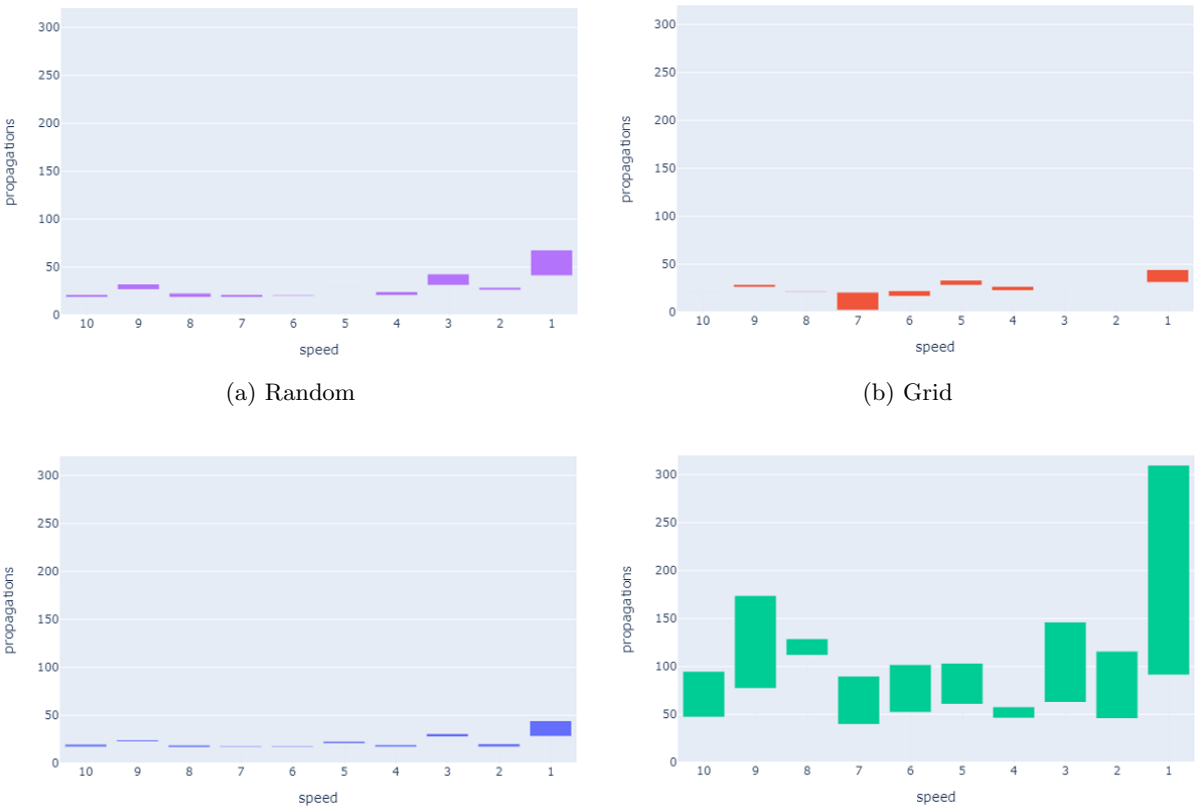
4 Experiments

Various experiments were tested on the broadcast: more than 200 runs were executed and all of the data were collected to provide a better analyse of the performance of the protocol under various settings. Multiple configurations were evaluated and the generated data were transformed both with Repast framework and other technologies. Indeed, some of the metrics were plot inside the Repast GUI using aggregations and built-in functions, while others were evaluated using Python and some scientific libraries.

In the following, different analysis that were carried out are presented, letting the reader know that all of the retrieved statistics were averaged over multiple runs to get more realistic results. That is, all of the tested configurations were evaluated three times and their average was used in the data visualization phase.

The different configurations varies by multiple factors, such as the topology used, the speed and the range of any perturbation, and the presence or absence of faulty channels. By considering the bandwidth as the number of perturbations received in a unit of time, its variation is simulated by changing the speed of perturbation. To emulate faulty channels a non-zero probability was introduced for any relay to drop the received perturbation without registering and forwarding it.

To evaluate the performance of the different configurations, the authors consider the average number of received perturbations and the average number of ARQ replies. These averages are evaluated on the



(a) Random

(b) Grid

(c) Circle

(d) Normal

Figure 3: Average number of received propagations with different propagation speeds. The columns represents the ranges of values retrieved during the multiple simulations. A narrower column indicates a lower deviation in the results.

total number of unique packages sent into the network and on the total number of observer. To put it simply, given P total perturbations in a network with N relays and M unique packets, the average number of received perturbations is evaluated as $avg = P/(N * M)$. For the evaluation of average ARQ replies, the formula is equivalent, but the numerator changes to take into consideration the total number of ARQ replies in the execution.

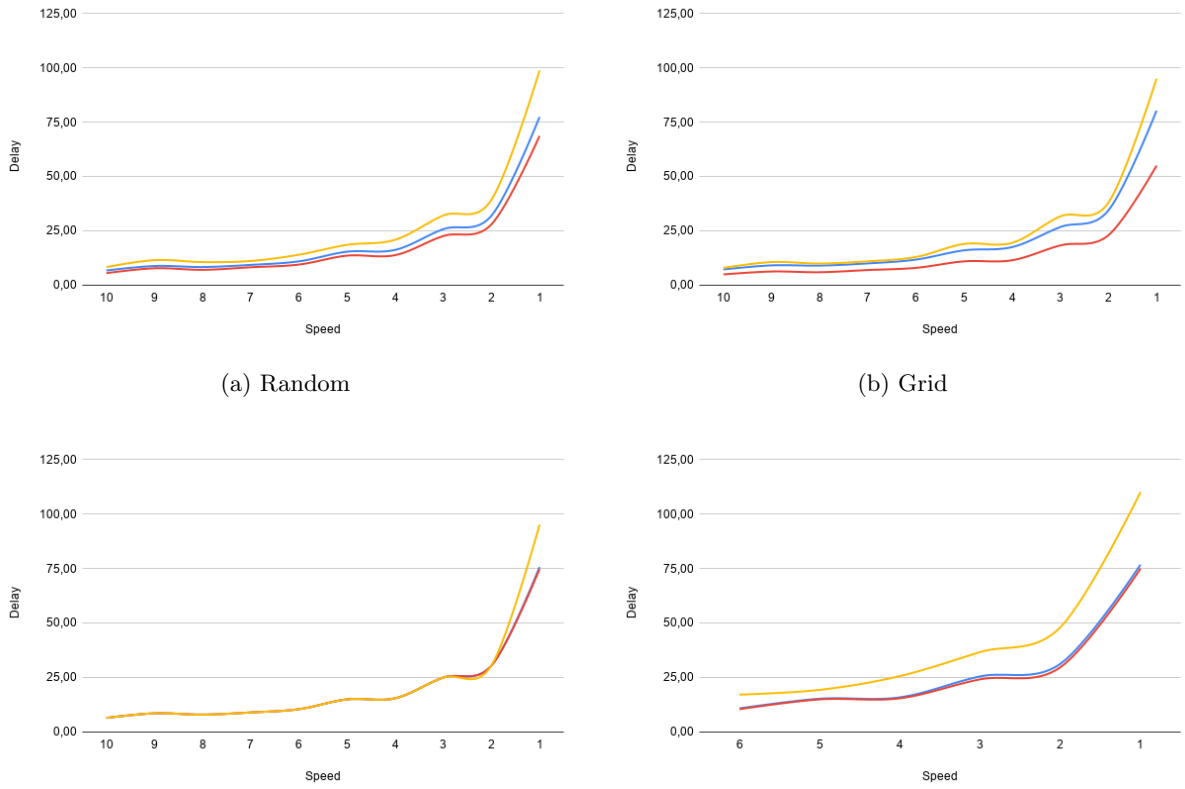
Small averages are synonyms of more optimized networks, where resources are not allocated to propagate duplicated messages. On the other hand, high averages implies an higher redundancy in the network. Therefore, the proposed metrics represents a tradeoff between performance and redundancy, where e.g. an average of 5 may be better than both a value of 1 and a value of 200.

4.1 Propagation speeds

In this experiment, all of the four topologies were tested with different propagation speeds. This analysis was done for two main reasons: the first one was to evaluate how the broadcast protocol performs in topologies of decreasing structure (i.e. from normal, where there is no fixed structure, to grid where each relay is placed at equal distance from its neighbors). The second cause was to evaluate how the protocol tolerates different bandwidths, here simulated by perturbing the propagation speed. The tested propagations speeds ranges from 10 to 1, where the value refers to the number of pixels a perturbation advance in scale (i.e. a speed of 2 requires half of the time to reach a point when compared with a speed of 1, and a speed of 10 requires ten times less that a speed of 1). The number of observers were set to 250, with a grid of 30x30.

As can be seen in figure 2, the faster a perturbation propagates, the less probable it is for a relay to ask for a retransmission and for another to reply to it. Moreover, in figure 3 the reader can also see that the total number of perturbations get reduced by increasing the speed. This can be explained by understanding that there is a correlation between the two indices, since a greater number of ARQ replies leads to a greater number of propagations.

What's more interesting to notice, is that the deviation in ARQ replies and propagations get smaller and smaller as the level of structure of the topology increases. Indeed, starting from the normal topology, which is the one with the least structure, and moving to the random, the grid and to the circle, the



(a) Random

(b) Grid

(c) Circle

(d) Normal

Figure 4: Average delays with different propagation speeds. Red lines represent average minimum delays, while yellow lines shows average maximum delays. Blue line show the average delay to be expected. An high difference between yellow and red lines shows an high variation in the values.

results get more and more predictable and consistent. More in depth, it is to state that it is not directly the randomness of the network that reduces its performance, but rather the heterogeneity of densities of relays in the space. Indeed, in all of the topologies except the normal one, relays are more or less equally distributed or equally distant between each other in the universe. Even if the random topology is by definition random, the relays are distributed in the space according to a uniform distribution and therefore the density is more or less homogeneous across the whole.

During this experiment, delays have been collected as well. As the perturbation speed increases, the delay decreases. In addition, the more the network is structured, the smaller the standard variation perceived. For more details, see table 1.

Table 1: Average delays per propagation speed

Speed	**Random**	**Normal**	**Circle**	**Grid**
1	76.8 (±0.12)	76.7 (±0.84)	75.6 (±1.18)	80.3 (±0.01)
2	31.7 (±0.05)	31.3 (±0.10)	30.49 (±0.01)	34.43 (±0)
3	25.8 (±0.21)	25.5 (±0.02)	25.0 (±0)	26.7 (±0)
4	16.0 (±0.20)	15.9 (±0.10)	15.5 (±0)	17.57 (±0)
5	15.4 (±0.12)	15.3 (±0.06)	15.0 (±0)	16.0 (±0)
6	10.8 (±0.04)	10.8 (±0.07)	10.5 (±0)	11.8 (±0)
7	9.2 (±0.05)	9.2 (±0.06)	9.0 (±0)	10.05 (±0)
8	8.3 (±0.05)	8.3 (±0.02)	8.0 (±0)	9.0 (±0)
9	8.8 (±0.04)	8.8 (±0.06)	8.6 (±0)	9.1 (±0)
10	6.7 (±0.02)	6.7 (±0.04)	6.5 (±0)	7.3 (±0)

Those results may not be good signs for the broadcast implementation, since the normal topology is the most probable structure in the real world, and since a reduction of bandwidth cannot be avoided in

the long run. However, it seems like the primary reasons for the drop in performance in the normal case is due to the perturbation range and to the ARQ mechanism, which by no means seems to be optimized for an environment with an high number of observers. Indeed, given how the ARQ is implemented (i.e. send a perturbation for each expected next message for each relay in the network at fixed intervals), for every round of ARQ one should expect to get N^2 perturbations in the network, where N is the number of observers. Adding to this, in highly concentrated networks a lot of relays may receive the ARQ request, to which a burst of replies is unavoidable.

4.2 Propagation radius

As an extension to the previous experiment, the authors also tested how the performance varies in the normal and in the random topologies when the propagation range decreases. This step was performed to understand better whether the normal distribution could be effectively improved by fine-tuning the perturbation ranges to the needs of the topology. In this setting, the grid and the circle topologies were not considered, since their structure makes so that relays are equidistant from one to the other and therefore the propagation range has less of an impact. Moreover, as it has been noticed in all of the tests, the performance of those two topologies is always similar to or better than random topologies.

As for the previous experiments, the universe is composed of a grid of 30x30 cells and the perturbation speed is set to 5. The collected information was averaged over three runs and the mean and the standard deviation are presented to give to the reader a summary of the discoveries. The tested values for the propagation radius ranges from 3 to 7, where the factor refers to the number of grid cells it travels through in any direction (i.e. the radius of the circle representing the perturbation at its maximum spread). Therefore, a propagation of radius equal to 3, will start from a dimension of 1 grid cell and will reach a diameter of 6 grid cells at the end of its life. Alongside the different radius tested, there will be presented also the percentage of the area that a perturbation covers at its full extension, to give the readers a better understanding of their dimensions.

Starting from the biggest radius and moving to the smallest ones, there is a reduction in the average number of perturbations perceived by any relay as well as a reduction in standard deviation (presented below in parentheses). Indeed, with a radius of 7 (i.e. a perturbation area that covers 17.1% of the total space of simulation), the average is 199.7 (±100.4) perturbations for the normal topology and an average of 24.8 (±2.2) for the random topology. Moving to a radius of 5, which corresponds to a coverage of 8.7% of the universe, the average lowers to 75.4 (±24.4) for the normal and to 29.9 (±0.43) for the random. Finally, for a value of 3, that covers 3.1% of the total space, the average lowers once more to 27.7 (±1.5) for the Gaussian distributed topology and to 6.2 (±0.85) for the uniformly distributed one.

From these results, summarized in table 2, it is noticeable the improvement in performance in both the networks as the perturbation radius decreases to cover a smaller number of observers while maintaining the network connected. Despite the performance improvements, it seems like redundancy is still too high in the normal setting, with an average of about 28 received perturbation for every message sent in the network. Therefore, it may be advisable to test also other configurations to make it more optimized.

Table 2: Average number of perturbations per propagation radius

Propagation radius	**Random**	**Normal**
3	6.22 (±0.86)	27.7 (±1.54)
5	29.9 (±0.43)	75.1 (±24.5)
7	24.8 (±2.21)	199.7 (±100.4)

4.3 Observer counts

Another experiment that may bring some insights about the performance of the broadcast protocol consists in perturbing the number of observers in the network. Indeed, by doing so it is possible to implicitly reduce the density of relays in the space, and to find a better balance for improving the protocol performances.

As for the previous experiment, the considered topologies are limited to the normal and the random, for the same reasons presented above. The universe is once again a grid of 30x30 cells and the perturbation speed is set to 5. Moreover, the tested values are 250, 125, 62, 31 observers distributed in the space. Going above and below these values seemed to not hold any significance, since increasing the count would have only augmented the density and reduced the performance, while reducing the number below 31 may caused the network to become too sparse and therefore hard to connect without changing the grid size or the perturbation radius.

The collected results shows that indeed the performance increases as the number of observers decreases both for the normal and for the random topology. In fact, starting with the normal topology, a reduction in the average number of perturbations can be observed, from 760.1 (±146.7) for 250 observers to 125.5 (±53.0) by just halving the count. Going forward, once again it can be noticed a decrease in the average to 53.6 (±21.8) with 62 observers and to 15.5 (±1.2) for 31 observers.

For what regards the random, the values are considerably smaller and less variant. Indeed, from 250 to 31 observers the reduction in average is of only 25.24 perturbations, which is almost 30 times smaller than the difference discovered in the normal. The complete list of results is reported in table 3.

Table 3: Average number of perturbations per observer counts

Observers count	**Random**	**Normal**
31	6.7 (±0.77)	15.5 (±1.25)
62	6.9 (±0.40)	53.6 (±21.8)
125	14.5 (±1.44)	125.5 (±53.0)
250	31.9 (±1.59)	760.1 (±146.8)

These results shows that the topologies with an higher concentration of relays have more to gain with the reduction in the total number of observers. What's more interesting to notice is how much the count of observers changes the performance of an heterogeneous and an homogeneous topology. Indeed, one can see an improvement of about 50 times in performance in the normal topology by using 1/8 of the total observers, while for the same configuration, the improvement is of around 5 times for the random topology.

4.4 Faulty channels

The last presented experiment is about faults and lost packets. In this section, it is reported an evaluation on how the average number of propagations varies when every relay have an independent chance of dropping a received packets. One last time, the universe is set to be a 30x30 grid cell, and the propagation speed and radius are set to 5. For this evaluation, all the four topologies are considered and their results are reported below. The drop chance was set to 0.0, 0.01 and 0.1 and represent the probability of an observer to drop an update without registering and forwarding it.

Once again, the results of the random, grid and circle topologies are almost identical, with their averages that lower slightly with an increase of the drop chance. In fact, in these three topologies there is on average a reduction of around 1/3 of the received propagations. In regards of the normal distribution, it can be perceived an opposite effect of the drop chance on the performance. Indeed, by increasing the drop chance from 0% to 10%, the average number of perturbations doubles. Table 4 contains the results retrieved in this experiment.

Table 4: Average number of perturbations per drop chance

Drop chance	**Random**	**Normal**	**Circle**	**Grid**
0%	29.9 (±0.44)	75.1 (±25.5)	22.2 (±1.13)	31.4 (±2.78)
1%	17.3 (±2.57)	133.3 (±47.3)	22.6 (±1.68)	20.9 (±2.46)
10%	13.6 (±1.11)	153.3 (±42.8)	21.0 (±2.07)	19.2 (±1.66)

These results, even if completely different between heterogeneous and homogeneous topologies, tells a lot. Indeed, thanks to the high redundancy of the network, even with an high drop chance of 10%, the protocol is still able to work really well in topologies such as random, grid and circle. Moreover, even in a normal topology, the drop in performance is not as drastic as it was in the previous experiments when the count of observers increased, or the propagation speed decreased. In addition, from the data it seems like the increase in the averages will not diverge to infinite, but rather will converge to a finite value.

The authors' hypothesis is that this reduction in quality in the normal topology is due to the ARQ protocol. Indeed, it seems like (i) it is highly unpredictable when an ARQ reply will be fired by a relay and (ii) in a highly dense network, a single ARQ request will cause a burst of ARQ replies and therefore strongly decrease the overall performance. About the increase in performance in the other topologies, it can be explained by restating the effect that the drop chance has on the network. Indeed, ignoring a perturbation means reducing the total number of packets received by the drop chance (e.g. with a drop

chance of 10%, the network senses 10% less packets). On the other hand, skipping an update may cause a series of ARQ replies, but this situation is still unlikely in structured topologies due to their shape.

5 Conclusions

In this work, an implementation of the append-only broadcast proposed by [1] was implemented and extensively tested. Despite the interest of the authors to perform more experiments, the general feeling is that the evaluated ones are enough to draw some conclusions about the performance of the broadcast protocol.

First of all, it was noticeable the effect that the ARQ protocol has on every tested topology. Indeed, it does not only enfeeble the overall performance of the network when the number of participants is considerably high (i.e. 250 relays), but it does so even when evaluated on a small group of observers. Moreover, according to the collected results, the number of ARQ replies increases as the network bandwidth decreases.

Another finding that can be tracked in every proposed experiment shows that the broadcast protocol is extremely sensitive to the selected parameters of the simulation. Indeed, even a single wrongly initialized argument may leads to a catastrophic reduction in performance. In fact, the tests have shown how much the average number of received perturbation varies by simply perturbing a little the propagation radius.

Overall, the protocol was perceived to be very unstable, and it may requires to draw some more assumptions to work effectively. To state it in other terms, it seems like developers should perform a grid search to find the best parameters to fit their needs. Indeed, when fine-tuned, the network was able to operate well, but the issue is that for most of the configurations that wasn't the case. Moreover, the fine-tuning process may be insufficient to optimize the performance when the protocol is evaluated in a completely dynamic setting (i.e. where nodes may leave or join).

Future works may involve the evaluation of another suite of experiments to deepen the understanding of the performance curve. To the authors it is of particular interest the implementation of asymmetric relays, where they don't share the same configuration, but rather have independent settings. Indeed, starting from this point it may be possible to make so that relays could learn to automatically set their parameters according to the aggregations collected at run-time. In fact, it seems like making the network self-configurable may be the best step to improve the performance.

References

[1] Christian F. Tschudin. A broadcast-only communication model based on replicated append-only logs. Technical Report 2, 2019.