Report on Final Assignment for Distributed Systems II

Daniele Parmeggiani

Parameter	Values
Protocol	Newscast, Cyclon
Graph	geo, random, lattice, star
Nodes	1000
View size	20, 50, 100
View to send size	6, 10, 15
ΔT	1, 4, 10
Disaster intensity	50%, 75%, 95%

Table 1. Parameter values used.

Nodes		Latency
global	global	10
geographical	geographical	4
local	local	1
global	geographical	4
geographical	local	1

Table 2. Latencies of edges of a Geo network.

The topic chosen for this project is among those suggested: comparing the Newscast and Cyclon protocols.

In this report, one particularly emphasized aspect is the underlying communication network's latency effect on the protocol. Such was considered after reading this passage in [Voulgaris et al., 2005, §3.2]:

> Nevertheless, ΔT should not be comparatively short to twice the typical latencies in the underlying network, as network delays would unpredictably affect the order in which events are taking place.

It occurred to try and model network latencies in the protocol, which was attempted in the presented implementation.

1. ON THE MODELING OF THE EXPERIMENTS

The experiments are modeled using both agent-based, and discrete event simulation techniques. The reason for using agent-based modeling is self-evident in the nature of the experiment, i.e. instances of some protocol that communicate with each other is a prime situation in which to use agent-based models. The reasoning for choosing to use discrete-events lies in the necessity to model message latency as well, and is presented in detail in the following §1.2.

In this model, we need to consider two kinds of networks, let us call them communication network and overlay network, even though these two terms are used interchangeably for instance in Jelasity et al. [2004]. Respectively, they are intended to model the ISO/OSI Layer 3 network, and the application layer communication; that is, the graph that can be constructed by considering the views of the nodes in the experiment.

The experiments are carried out over both protocols, using the parameters presented in Table 1.

Let us briefly discuss the different graphs mentioned in the table:

- *geo* is explained in detail in the following §1.1,
- *random* is an Erdős–Rényi graph as in $G(n = 1000, p = 10\%)$,
- *lattice* is a ring network of 1000 nodes, and
- *star* is a star network, with a central node and the rest being the $1000 - 1$ nodes connected to it.

Although the suggested parameter value of 100 000 was considered, it was ultimately not used because of the higher memory requirement. Using 1000 nodes allowed the experiment runs to be completed on the system available.

Let us distinguish two terms to be used as time units. We may distinguish the notions of steps, as in steps of MESA's schedule, and of cycles, that is, let a cycle be the amount of steps equal to ΔT. This is useful so as to consider the protocols behavior regardless of the value we may assign to ΔT. It is reasonable to assume that in the steps between cycles, differences in the metrics evaluated (see §1.5) are not great.

1.1 The Geo network

One structured network over which latencies are also modeled is referenced here and in the code as a *Geo* network.

Ideally, this graph is intended to model a network that spans greater geographical regions, having one core "global" network of 6 nodes, 6 geographical networks each composed of 6 nodes, and $\lfloor \frac{1000}{36} \rfloor = 27$ nodes for each local network (that is, a network attached to any of the 36 geographical nodes).

The latency of graph edges is defined based on the class of involved nodes, and is summarized in Table 2.

The total latency between nodes is then computed by using Dijkstra's algorithm.

The interest in this particular network lies in finding a scenario in which partitioning of the network seemed very likely. That is, we're interested in finding the effect of latency over the two protocols, namely whether or not they're able to avoid partitioning in this network, and under what circumstances.

1.2 Messaging and Latency

In the Cyclon and Newscast protocols, nodes exchange messages with each other. In the presented implementation, this is modeled by means of a discrete event simulation.

Essentially, each message is pushed into the recipient node's incoming messages queue (which is a heap queue), sorted by the time the message is to be read; that is, the time at which the message was push into the recipient queue, plus the latency between the sender and the recipient.

The reading time is computed by the sender node which uses a pre-computed routing table, which is then used to compute the total latency between the two nodes. In all networks, except the Geo network, all edges have latency $= 1$.

A peculiar aspect of the modeled messaging system is that nodes have an infinite capacity to process messages; that is, no matter how many messages a node has to process at any given step t, at step $t+1$ all those messages will have been processed. While not being an accurate representation of really existing computing networks, the underlying idea is that the computational complexity of processing messages pertaining to these protocols should be negligible.

Throughout an experiment, an agent i (being the unique numeric identifier of each agent) may choose to send a message at step t, if the following holds:

$$(t+i) \bmod \Delta T = 0$$

This yields notable properties:

- the messages respect the cadence of being sent once per cycle, and
- the messages are not sent synchronously.

The latter being an important property mentioned in [Voulgaris et al., 2005, §2.2]. We can notice that this in effect will entail that at each step, the number of messages sent is the same, since there's a uniform probability that an agent will send a message at any of the steps in a cycle. This may not model exactly the behavior of a really running protocol, but no evidence pointed to consider this as a possible refutation of the results presented.

1.3 Disaster and Resurrection

At cycle 100, a disaster is set to happen, meaning that an amount of nodes equals to 1000 $\times$ disaster intensity is set to crash: losing its memory and halting communications. Some nodes are exempt from crashing: in the Geo network only local nodes can crash, and in the star network, the central node cannot crash. This is intended to model a supposed higher availability of certain nodes that are more important in the communication network.

Note that regardless of a node's state, routing of messages is not affected. This actually has no implications for the Geo and star networks, since the routing nodes are exempted from crashing, but in the lattice and random networks this behavior may sound a little unreasonable. Nevertheless, we're mostly interested in creating an evaluation baseline in these graphs: this improper feature should not render the results meaningless.

At cycle 200, all the nodes that crashed are recovered, and at cycle 300 the experiment finishes. Recovered nodes undergo bootstrapping once again.

1.4 Bootstrapping

By bootstrapping, in the context of the presented implementation, we mean the mechanism for which the view of a node is populated, either at the beginning of the experiment, or when the node is recovered.

The bootstrapping mechanism differs for each network type. Let us examine them separately.

In the lattice network, every node is bootstrapped with the two other nodes that are adjacent to it.

In the star network, the central node is bootstrapped with 5 random other nodes, the rest are bootstrapped with the central node.

In the random network, each node is bootstrapped with 5 other adjacent nodes, randomly selected.

In the Geo network, the bootstrapping is intended to model some particular features that are considered to be reasonable for such a topology: the local nodes are bootstrapped dynamically, and the 42 global and geographical nodes are bootstrapped statically to know each other; that is, every global node knows about every other global node, and the geographical nodes that are adjacent to it, while the geographical nodes know about the other nodes in their geographical network as well as the global node they are related to. Additionally, the geographical nodes are also dynamically bootstrapped with 5 random nodes in the local network for which they are the "gateway." Finally, the local nodes are bootstrapped with 5 random nodes in their network, as well as their geographical "gateway."

The dynamical behavior of the local nodes is intended to model a bootstrapping mechanism involving a local network broadcast request. Therefore, we can choose some random nodes in the same "LAN" to be responding to this request. In particular, we choose 5 as the number of nodes that will respond, this is because in an actual implementation of such a broadcast we wouldn't want to have the broadcasts to be actually executed by every node, so as to not flood the network with such requests, especially when the protocol is initialized over the network.

So, a somewhat obvious optimization is to have a cooldown period at the beginning of a node's bootstrapping in which the node listens to the possible broadcasts requests occurring, and only after that actually initiate the broadcast. If the initialization is not simultaneous, we can imagine that the first broadcasts would serve a significant portion of the nodes in the local network.

This is not modeled exactly in the implementation presented, but the 5 randomly selected nodes have been considered to be a sufficient depiction of this situation. A more sophisticated implementation would try to increase the probability of a node being randomly selected if it was previously selected by another node.

Metric	Description
Clustering Coefficient	As in [Montresor, 2018, p. 11].
Average Path Length	As in [Montresor, 2018, p. 13].
Degree	As the mean of [Montresor, 2018, p. 15].
Unprocessed Messages	Number of messages that are in the incoming messages queues.
Average Message Latency	Average of amount of time left before a message is read, over the messages in the incoming messages queues.
Partitions	Number of connected components of the overlay network.
Pollution	Percentage of dead nodes in alive nodes views.
Alive Agents	The total number of protocol agents that are running.

Table 3. Metrics collected.

1.5 Collected Metrics

Table 3 shows the metrics that have been collected for each step of the simulation. Each metric is computed over the alive nodes at each step of the simulation. Metrics referring to all nodes, indiscriminate of their status, have also been collected, for each of those listed in Table 3, except for *pollution* and *alive agents*, though they are less interesting.

1.6 Implementation in Python

The model described so far was implemented in Python 3.10. This implementation can be found on Parmeggiani [2022].

Several libraries were used in the making of the implementation; some are listed here in no particular order:

- `igraph`,
- `matplotlib`,
- `numpy`,
- `pandas`,
- `typer`.

In Figure 1 we can see an outline of the structure of the files involved. Most notably, the `main.py` file can be executed directly and provides both a way to execute one single experiment in an interactive and visual way (calling the `viz.py` file internally), and a way to execute all the experiments parameterized as in Table 1 using the `multiprocessing` library for added performance. The `experiment.py` file holds the core instructions that implement the model presented, and the `graph.py` file holds utilities for building the four classes of graph described. Finally, the `analyses.ipynb` file contains a notebook that produced the data found in the following §2.

The core of the `experiment.py` file is around the exchange of messages between agents of the protocol, see lines 342–397.

2. EXPERIMENTAL RESULTS

Let us provide a way to reference experiment runs,[1] based on the parameters values employed,[2] to be used

```
ds-2-assignment
  analyses.ipynb
  experiment.py
  main.py
  viz.py
  requirements.txt
  report/
      ...
  runs/
      ... .csv
```

Fig. 1. Structure of the implementation files.

throughout this section. Taking P to mean the protocol, G the graph, n the number of nodes in G, V the size of a node's view, S the shuffle length,[3] ΔT the number of steps in a cycle, and d the disaster intensity, an experiment run may be unequivocally referenced as a tuple:

$$(P, G, n, V, S, \Delta T, d)$$

2.1 Convergence Metrics

With regards to the convergence metrics (i.e. clustering coefficient, average degree, average path length), Figures 5–16, it is hard to provide valuable insights from the plots.

The values are not as they were expected, and in some instances, even contradictory. For instance see Figure 6, and recognize that Cyclon 20 averaged a higher clustering coefficient than Cyclon 100, which should not take place since the second should inherently have a higher degree, given the larger view size. Consider even, that it is possible, in Cyclon 100, for a node to hold information about 100 neighbors, which amounts to ~10%[4] of a 1000 nodes network.

This is probably due to a very sharp difference between the model presented here (the added latency), and the models that were used in Jelasity et al. [2004], and Voulgaris et al. [2005]. For instance, consider that between steps 2 and 5 of [Voulgaris et al., 2005, §2.2], a node's view may have changed.

For these reasons, it would possible to devise a new model that considers these issues and tries, for instance, to hold a session memory when interchanging messages between nodes.

2.2 Partitioned Runs

Let us call an experiment run *partitioned*, if the number of partitions (i.e. connected components) was greater than 1 within the first 100 cycles of the simulation; thus, before disaster. The partitioned runs are listed in Table 4.

Unsurprisingly, all of them took place in the Geo network, the only one actually exhibiting substantial delays in message exchanges.

It must be noted though, that almost all the partitioned runs were recovered; that is, the number of partitions was greater than 1 for a limited amount of steps. In fact, a common behavior of partitioned runs involves an initial

[1] That is, an instance of the model being executed from the first to last step, having its results recorded.
[2] Recall Table 1.

[3] Not applicable to runs pertaining to Newscast.
[4] It is *about* 10%. For instance, a Geo network of 1000, actually has 1014 nodes. Refer to [Parmeggiani, 2022, `graph.py`] for details.

Protocol	Graph	Nodes	View size	Shuffle length	ΔT	Disaster intensity
Cyclon	geo	1000	20	6	1	50%
Cyclon	geo	1000	20	6	1	75%
Cyclon	geo	1000	20	6	1	95%
Cyclon	geo	1000	20	10	1	50%
Cyclon	geo	1000	20	10	1	75%
Cyclon	geo	1000	20	15	1	50%
Cyclon	geo	1000	20	15	1	75%
Cyclon	geo	1000	20	15	1	95%
Cyclon	geo	1000	50	6	1	50%
Cyclon	geo	1000	50	6	1	75%
Cyclon	geo	1000	50	6	1	95%
Cyclon	geo	1000	100	10	1	50%
Cyclon	geo	1000	100	10	1	75%
Cyclon	geo	1000	100	10	1	95%
Newscast	geo	1000	20	N/A	1	50%
Newscast	geo	1000	20	N/A	1	75%
Newscast	geo	1000	20	N/A	1	95%
Newscast	geo	1000	20	N/A	4	50%
Newscast	geo	1000	20	N/A	4	75%
Newscast	geo	1000	20	N/A	4	95%

Table 4. Partitioned runs.

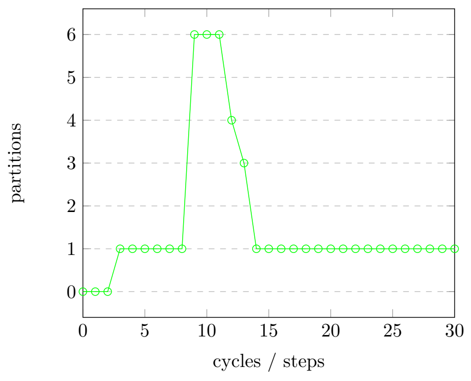

Fig. 2. Number of partitions in experiment (Newscast, geo, 1000, 20, N/A, 1, 50%).

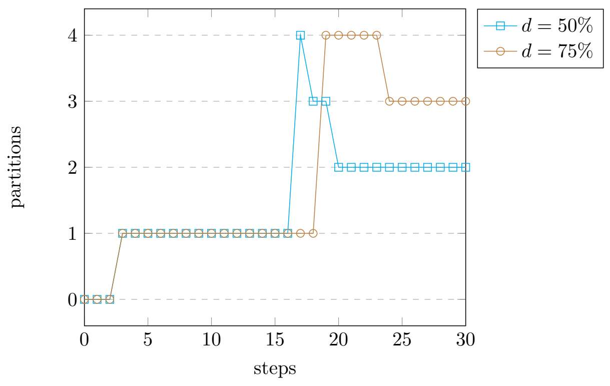

Fig. 3. Number of partitions in experiments (Newscast, geo, 1000, 20, N/A, 1, d).

phase of a few steps in which the number of partitions is equal to 1, followed by an unstable stage of fluctuations in the number of partitions, later returning to a stable single partition. This pattern is exemplified in the run shown in Figure 2.

Note the behavior of bootstrapping in relation to this Figure. That is, in the initial steps of a simulation, the nodes have their views set by the bootstrapping mechanism, which entails that the overlay network is composed of a single connected graph. Therefore, it is expected to find a single partition in the first steps. Furthermore, at step 10, the *global* nodes have sent messages to each other, according to their bootstrapped view, but are yet to receive them. This can explain the fluctuations in number of partitions; that is, in the experiment in Figure, after 10 steps, 10 cycles have also elapsed, thus 10 times have the views of the global nodes been updated. But such updates could not include the other global nodes because of their connection's latency, therefore partitioning is likely to happen.

The reasoning above applies to graphs other than Geo as well, since the same preconditions regarding bootstrapping also apply.

This explanation also encourages to think that the extent of initial partitioning is inversely proportional to ΔT, which can also explain why most of the partitioned runs pertain to experiments having $\Delta T = 1$.

There were only two runs which didn't not undergo this recovery behavior. Namely:

- (Newscast, geo, 1000, 20, N/A, 4, 50%), and
- (Newscast, geo, 1000, 20, N/A, 4, 75%).

Their initial behavior with regards to partitioning is shown in Figure 3.

Peculiarly, Cyclon runs experienced partitioning exactly at cycle 8,[5] not before, nor after.

Although the number of partitioned runs is skewed towards a greater number of Cyclon runs, we should not be mislead to think this is due to Cyclon's higher susceptibility to partitioning. In fact, Cyclon runs compose ~75% of the total runs performed, and 70% of the partitioned runs, thus it should seem less prone to partitioning. Furthermore, all partitioned Cyclon runs were recovered, while 2 out of the 6 partitioned Newscast runs were not. Also, it should be noted that Cyclon exhibited no partitioned runs

[5] Or step 8, since $\Delta T = 1$ in all of Cyclon's partitioned runs.

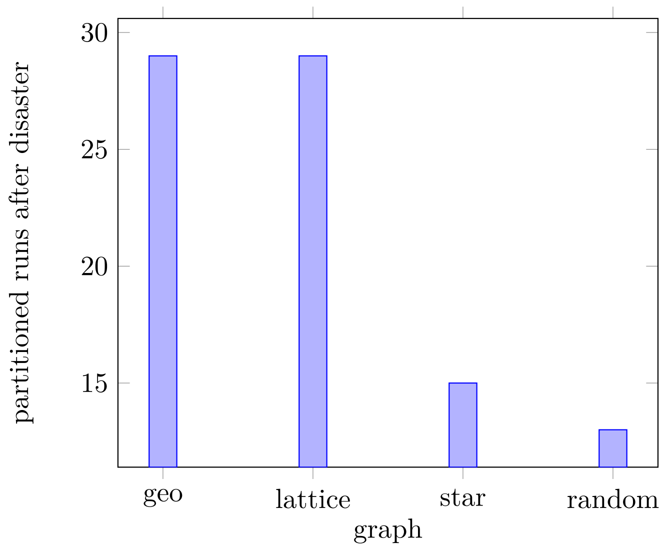

Fig. 4. Distribution of runs partitioned, by communication graph.

with $\Delta T > 1$, while Newscast did. Considering this, the evidence clearly points to believe that Newscast is more inclined to partitioning than Cyclon.

2.3 Robustness to catastrophic failures

Let us conduct a similar reasoning as per the above §2.2: we will consider the runs that partitioned as a result of the crashing of nodes.

Let us consider a run to be partitioned after disaster, like in the above notion, in case the number of connected components in the overlay graph is greater than 1, between cycles 100 and 200. Let us also disregard the set of runs which became partitioned prior to disaster, as we would trivially expect them to be partitioned after disaster as well.

The partitioned-after-disaster runs are listed in Table 5. Some of these runs did get back in a single partition before all crashed nodes were recovered, and are marked as such.

Looking at the table, we can notice how the underlying communication graph can impact the tendency to partition. In Figure 4 the distribution of partitioned runs (including those runs that partitioned before disaster) is depicted. As mentioned before, 20 out of the 29 partitioned runs over the Geo network, partitioned prior to disaster.

Let us now consider the effect of disaster intensity. We can see that 63 out of the 144 (~44%) runs with 95% disaster intensity partitioned after disaster, while only 2 out of 144 (~1%) runs with $d = 75\%$, and 0 runs with $d = 50\%$ were affected by the disaster. Although it is evident that disaster intensity is a great predictor of a run partitioning, more evidence needs to be collected [6] in order to gather a more precise function relating disaster intensity to unrecoverable partitioning.

[6] Possibly, over more numerous graphs.

2.4 Self-Cleaning

The metric related to the self-cleaning behavior of the two protocols is pollution (cf. Table 3). Foremost, we can see from Figures 17–20 that under no circumstances every crashed node was removed from every correct node's view, within 100 cycles. This was not expected. [7]

A reasonable explanation related to the distinct feature of latency introduced in the presented model, is that while one node may have crashed, the messages it sent out while still being correct are still "traversing" the communication network, even after having crashed.

3. CONCLUSIONS

We have seen how a model may be devised so that it was different from those proposed in Jelasity et al. [2004] and in Voulgaris et al. [2005], and tested the Newscast and Cyclon protocols against it.

We have seen that some of the results presented in this report were not as expected, and have provided some insight into why that would be. Nevertheless, we have drawn some conclusions from those results that could be explained on the grounds of the model itself.

It is possible to outline a new model that can overcome the issues encountered, and see if it would still reflect the findings of this one.

REFERENCES

Mark Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Marteen van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. *Fifth ACM/IFIP/USENIX International Middleware Conference, Toronto, Canada*, 13(2), October 2004.

Alberto Montresor. *Distributed Algorithms. Epidemic Protocols: Beyond dissemination.* October 2018.

Daniele Parmeggiani. Final assignment for distributed systems ii. January 2022. URL `https://github.com/dpdani/ds-2-assignment`.

Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2), June 2005.

[7] See [Montresor, 2018, p. 17].

Protocol	Graph	Nodes	View size	Shuffle length	ΔT	Disaster intensity	Recovered
Cyclon	geo	1000	20	6	4	95%	no
Cyclon	geo	1000	20	6	10	95%	no
Cyclon	geo	1000	20	10	1	95%	no
Cyclon	geo	1000	20	10	4	95%	no
Cyclon	geo	1000	20	10	10	95%	no
Cyclon	geo	1000	20	15	4	95%	yes
Cyclon	geo	1000	20	15	10	95%	no
Cyclon	geo	1000	50	6	10	95%	yes
Cyclon	geo	1000	50	15	10	95%	yes
Cyclon	lattice	1000	20	6	1	75%	yes
Cyclon	lattice	1000	20	6	1	95%	no
Cyclon	lattice	1000	20	6	10	95%	no
Cyclon	lattice	1000	20	6	4	95%	no
Cyclon	lattice	1000	20	10	1	95%	no
Cyclon	lattice	1000	20	10	4	75%	yes
Cyclon	lattice	1000	20	10	4	95%	no
Cyclon	lattice	1000	20	10	10	95%	no
Cyclon	lattice	1000	20	15	1	95%	no
Cyclon	lattice	1000	20	15	4	95%	no
Cyclon	lattice	1000	20	15	10	95%	no
Cyclon	lattice	1000	50	6	1	95%	no
Cyclon	lattice	1000	50	6	4	95%	yes
Cyclon	lattice	1000	50	6	10	95%	yes
Cyclon	lattice	1000	50	10	1	95%	yes
Cyclon	lattice	1000	50	10	4	95%	yes
Cyclon	lattice	1000	50	10	10	95%	yes
Cyclon	lattice	1000	50	15	1	95%	yes
Cyclon	lattice	1000	50	15	4	95%	yes
Cyclon	lattice	1000	50	15	10	95%	yes
Cyclon	lattice	1000	100	6	1	95%	no
Cyclon	lattice	1000	100	6	4	95%	no
Cyclon	lattice	1000	100	6	10	95%	no
Cyclon	lattice	1000	100	10	1	95%	no
Cyclon	random	1000	20	6	1	95%	no
Cyclon	random	1000	20	6	4	95%	no
Cyclon	random	1000	20	6	10	95%	no
Cyclon	random	1000	20	10	1	95%	no
Cyclon	random	1000	20	10	4	95%	no
Cyclon	random	1000	20	10	10	95%	no
Cyclon	random	1000	20	15	1	95%	no
Cyclon	random	1000	20	15	4	95%	no
Cyclon	random	1000	20	15	10	95%	no
Cyclon	random	1000	50	10	1	95%	no
Cyclon	random	1000	50	15	4	95%	no
Cyclon	star	1000	20	6	1	95%	no
Cyclon	star	1000	20	6	4	95%	no
Cyclon	star	1000	20	6	10	95%	no
Cyclon	star	1000	20	10	1	95%	no
Cyclon	star	1000	20	10	4	95%	no
Cyclon	star	1000	20	10	10	95%	no
Cyclon	star	1000	20	15	1	95%	no
Cyclon	star	1000	20	15	4	95%	no
Cyclon	star	1000	20	15	10	95%	no
Cyclon	star	1000	50	6	4	95%	no
Cyclon	star	1000	50	6	10	95%	no
Cyclon	star	1000	50	10	1	95%	no
Cyclon	star	1000	50	10	4	95%	no
Cyclon	star	1000	50	15	1	95%	yes
Cyclon	star	1000	50	15	4	95%	no
Newscast	lattice	1000	20	N/A	1	75%	yes
Newscast	lattice	1000	20	N/A	1	95%	no
Newscast	lattice	1000	20	N/A	4	95%	no
Newscast	lattice	1000	50	N/A	1	95%	no
Newscast	lattice	1000	50	N/A	4	95%	no
Newscast	random	1000	20	N/A	10	95%	no
Newscast	random	1000	20	N/A	4	95%	no

Table 5. Runs partitioned after disaster.

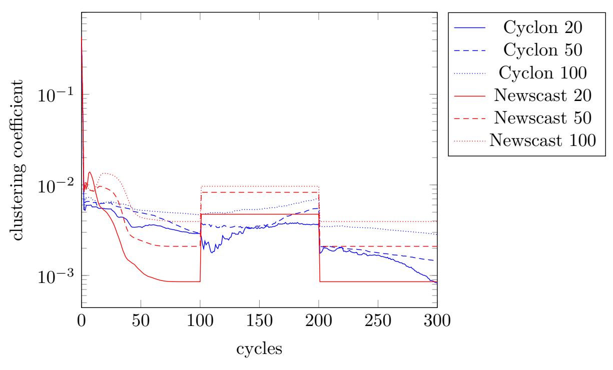

Fig. 5. Clustering coefficient in Geo graphs.

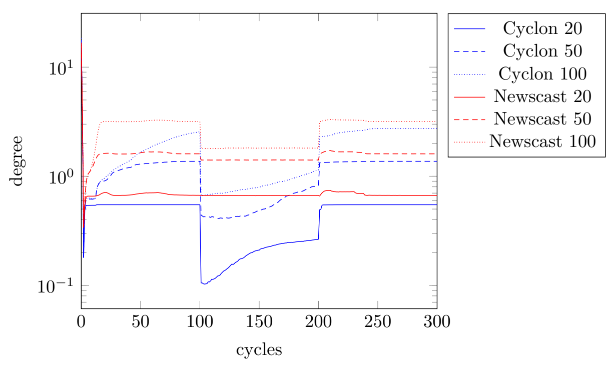

Fig. 9. Degree in Geo graphs.

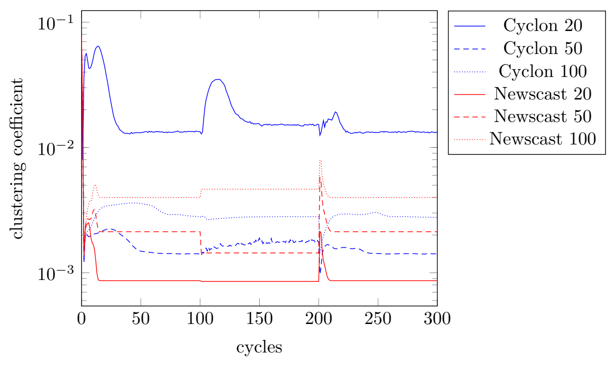

Fig. 6. Clustering coefficient in random graphs.

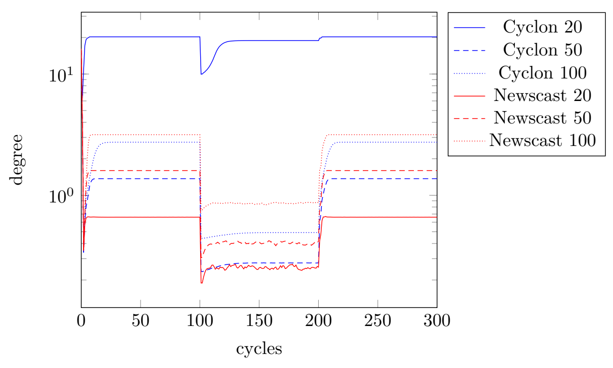

Fig. 10. Degree in random graphs.

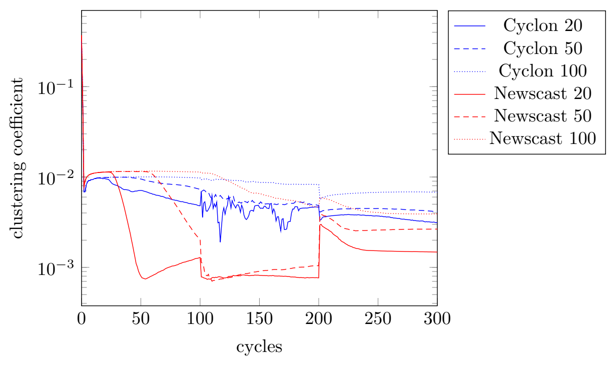

Fig. 7. Clustering coefficient in lattice graphs.

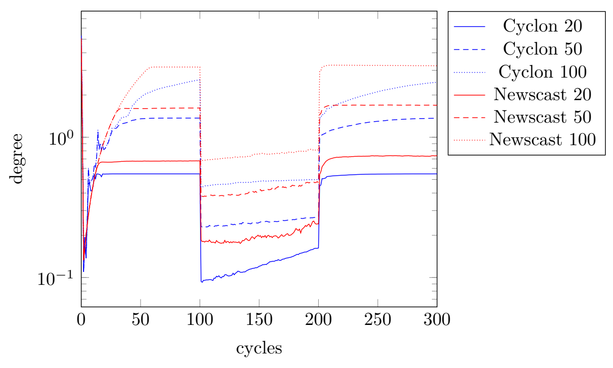

Fig. 11. Degree in lattice graphs.

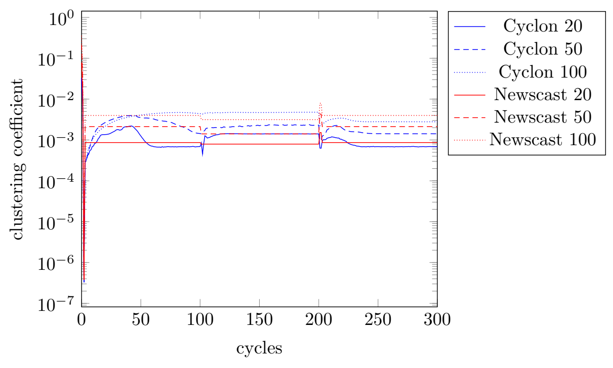

Fig. 8. Clustering coefficient in star graphs.

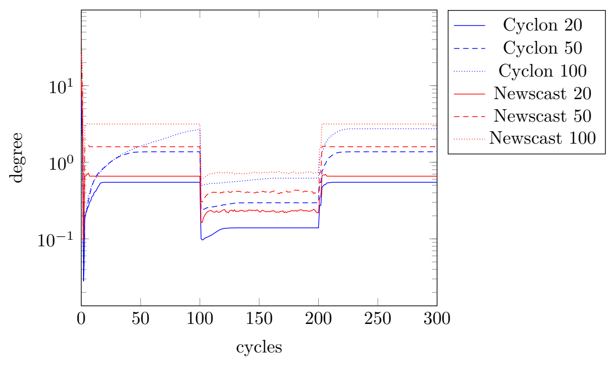

Fig. 12. Degree in star graphs.

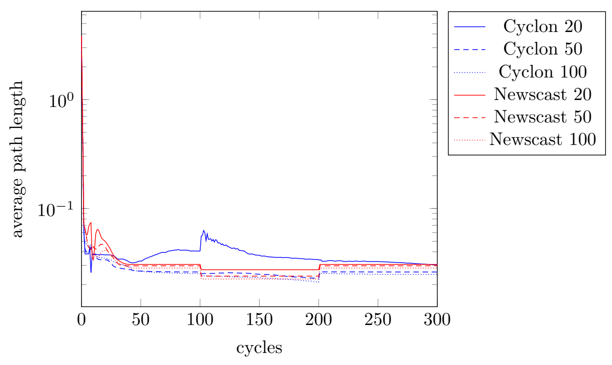

Fig. 13. Average path length in Geo graphs.

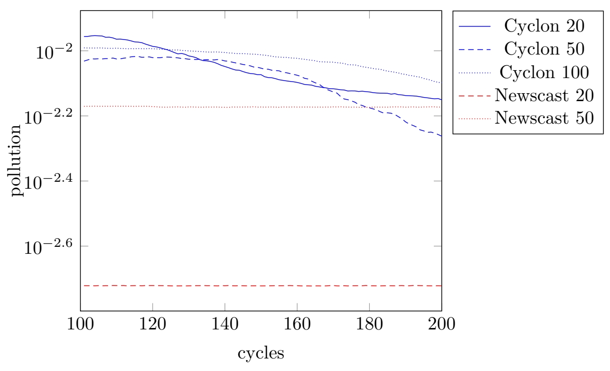

Fig. 17. Pollution in Geo graphs.

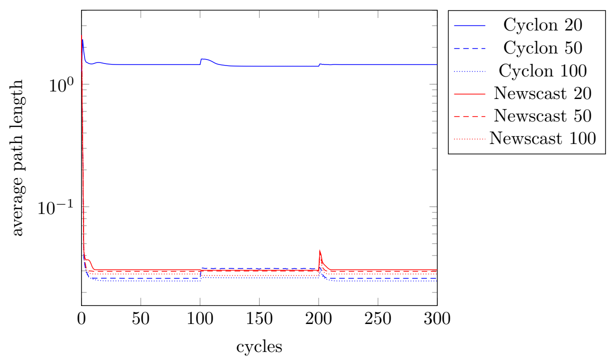

Fig. 14. Average path length in random graphs.

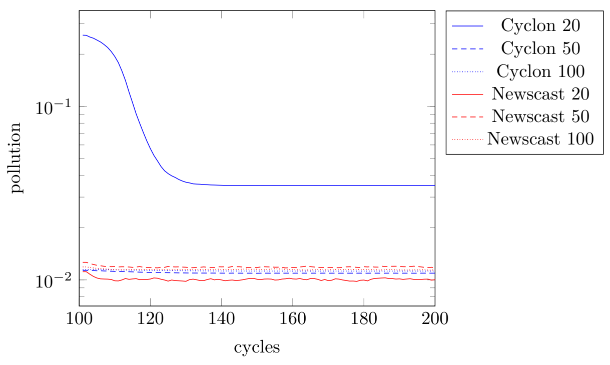

Fig. 18. Pollution in random graphs.

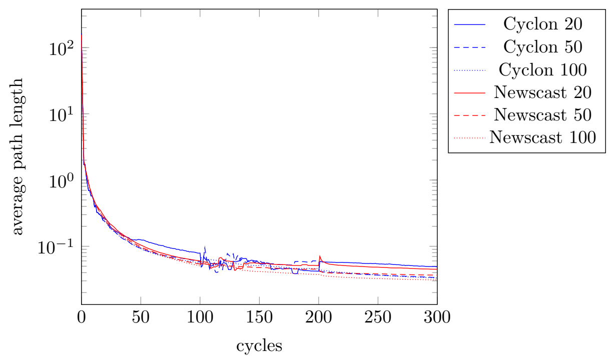

Fig. 15. Average path length in lattice graphs.

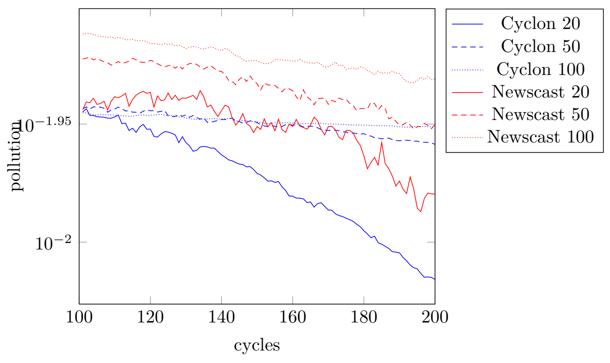

Fig. 19. Pollution in lattice graphs.

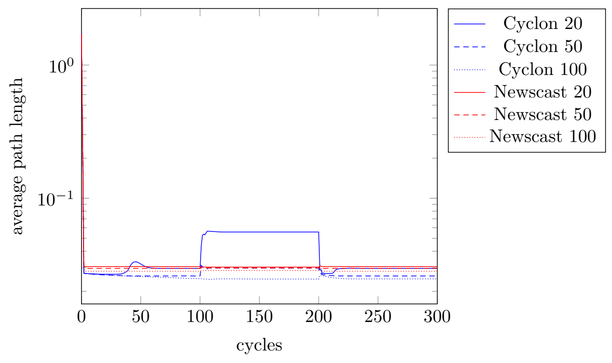

Fig. 16. Average path length in star graphs.

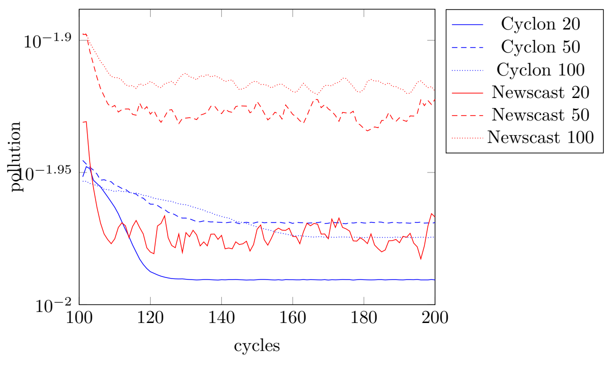

Fig. 20. Pollution in star graphs.

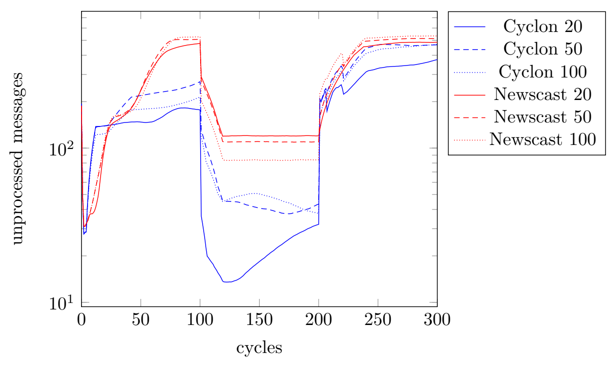

Fig. 21. Unprocessed messages in Geo graphs.

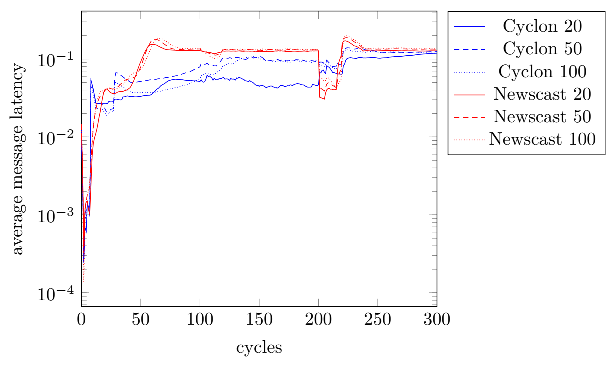

Fig. 25. Average message latency in Geo graphs.

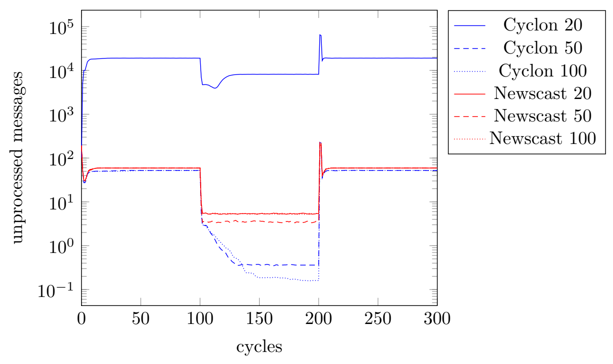

Fig. 22. Unprocessed messages in random graphs.

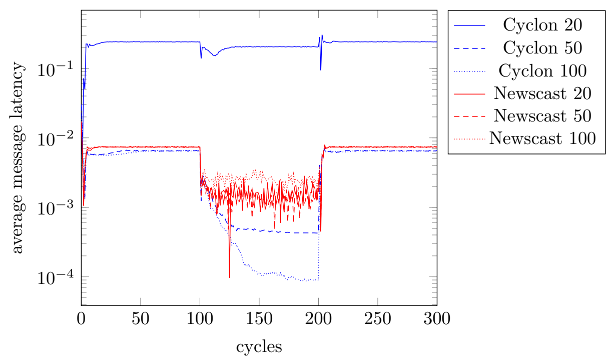

Fig. 26. Average message latency in random graphs.

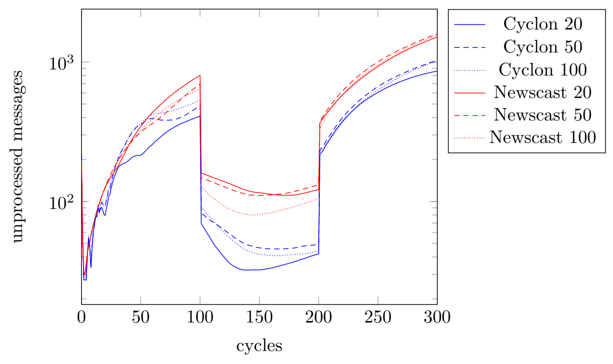

Fig. 23. Unprocessed messages in lattice graphs.

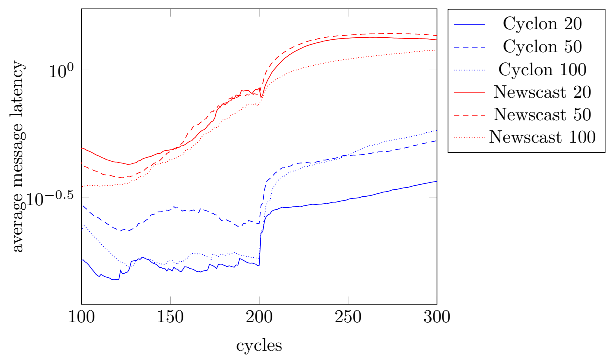

Fig. 27. Average message latency in lattice graphs.

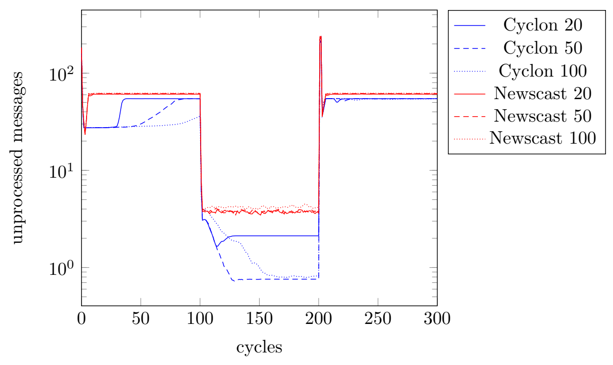

Fig. 24. Unprocessed messages in star graphs.

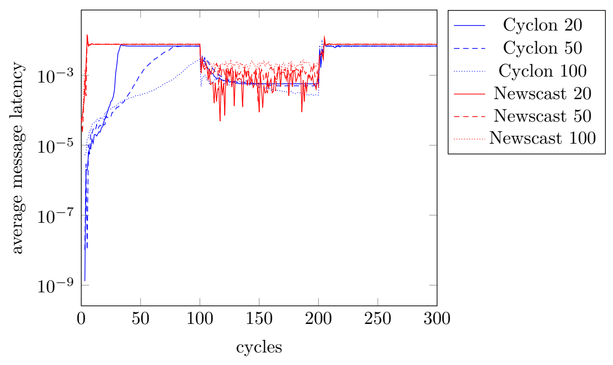

Fig. 28. Average message latency in star graphs.