Large-Scale Distributed Systems

Project 1. Gossip-Based Dissemination, Peer Sampling Service

Laura Rettig[*]

March 27, 2014

Contents

[*]laura.rettig@unifr.ch

1 Gossip-Based Dissemination

1.1 Anti-Entropy

We would expect a message dissemination that starts slowly when few nodes are infected, since only few of the periodic exchanges are between nodes that have the message, whereas the rest of exchanges is useless. The dissemination speed would then increase over time as more nodes are up to date, allowing for more exchanges between infected and non-infected nodes.

To test this hypothesis, the anti-entropy protocol has been implemented and tested using 40 Splayds (nodes) on the local cluster[1] with a gossiping period of 5 seconds.

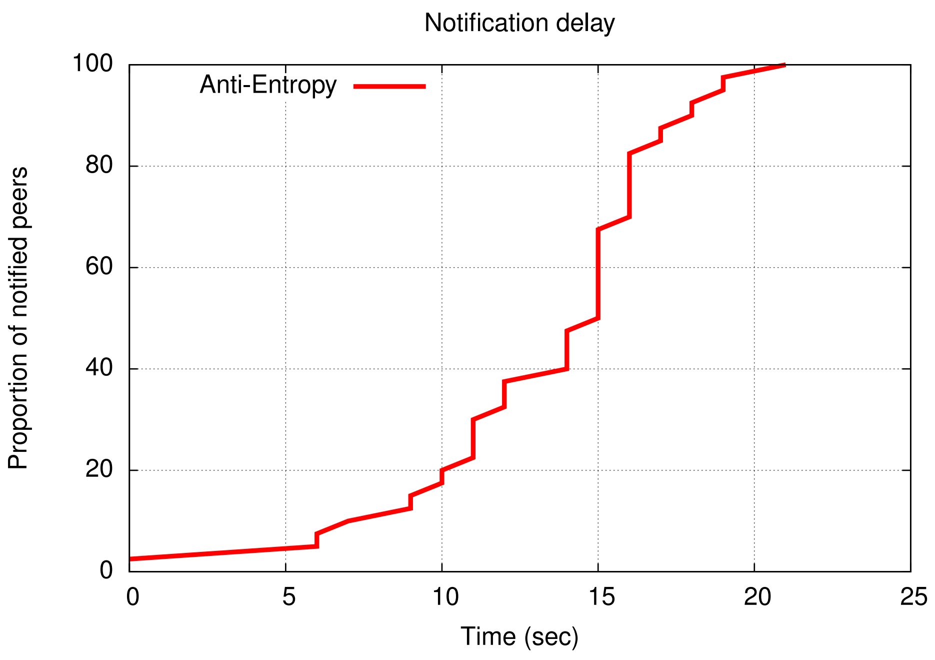

Figure 1: Proportion of infected nodes over time using anti-entropy dissemination.

Indeed, Figure 1 shows that the message dissemination starts rather slowly when few nodes are notified, but after reaching ca. 20% of nodes (after around 10 seconds), the speed of the dissemination (number of nodes per second) increases more and more. Eventually after around 20 seconds, all nodes are notified. However, the exchanges continue until the program terminates, so it continues to consume resources. In the defined timespan of 120 seconds, depending on the initial offset each node initiates an exchange 23-24 times, leading to around 940 exchanges with the aim of notifying 40 peers, meaning there are many duplicates sent after 22 seconds, once all are infected (and increasingly many before). So by the nature of the protocol, only a small percentage of messages are actually useful in the given case (with only one message to be passed, frequent exchanges, and no incoming additional messages).
On the other hand, the randomness and the periodic exchanges guarantee that eventually all nodes will be infected.

[1]http://splay3.unineuchatel.ch:8080

1.2 Rumor Mongering

Hypothetically, the upper limit for the number of peers that can be reached with a certain combination of values for f and HTL can be easily computed:

```
For f=2, HTL=3:
neighbors at each cycle:  (source)1 + 2 + (2 + 2) + (2 + 2 + 2 + 2) = 15
remaining hops:                   3   2    1          0
```

As a general formula,

$$max\_nodes = \sum_{i=0}^{HTL} f^i$$

However, this would assume that no duplicates are sent. Therefore, the true values are notably lower (cf. Table 1).
Additionally, even the sum of infections and duplicates is lower than given by the formula, since nodes might receive a message while still having one in buffer, leading them to discarding the one with fewer remaining hops, such that some messages never fully consume all their hops.

As an experiment to determine the optimum combination of f and HTL values for the system, the implementation was run with different constant values for `HTL` and `f`, i.e. to have different "message lifetimes" and to select different numbers of peers when there is a new message to disseminate. Table 1 displays the results, comparing the number of infected nodes (out of the 40 that were in the system) to the number of duplicate messages that were received.

Table 1: Infection and duplicates for different HTL and f values.

		HTL: Hops To Live			
		3	4	5	6
f: number of peers	2	infected: 11 duplicates: 1	infected: 21 duplicates: 7	infected: 29 duplicates: 15	infected: 33.5 duplicates: 32.5
	3	infected: 25 duplicates: 9	infected: 32 duplicates: 27.5	infected: 37.5 duplicates: 61	infected: 38.5 duplicates: 121.5
	4	infected: 26.5 duplicates: 20.5	infected: 38.5 duplicates: 94.5	infected: 39 duplicates: 106	infected: 40 duplicates: 501
	5	infected: 31.5 duplicates: 24.5	infected: 38 duplicates: 74.5	infected: 39 duplicates: 217.5	infected: 40 duplicates: 691
	6	infected: 37.5 duplicates: 47.5	infected: 40 duplicates: 486	infected: 40 duplicates: 687	infected: 40 duplicates: 843

Since these values are averages, a value slightly below 40 means that sometimes all nodes are reached, sometimes just almost. Setting the values such that all nodes are guaranteed to be notified rapidly increases duplicates (compare for example the messages for f=4, HTL=5 with those for f=4, HTL=6).

The choice of f/HTL values is hence a trade-off between the acceptable amount of traffic and the importance of information consistency (i.e. are all nodes up to date on the most recent message, or did we miss a few?).

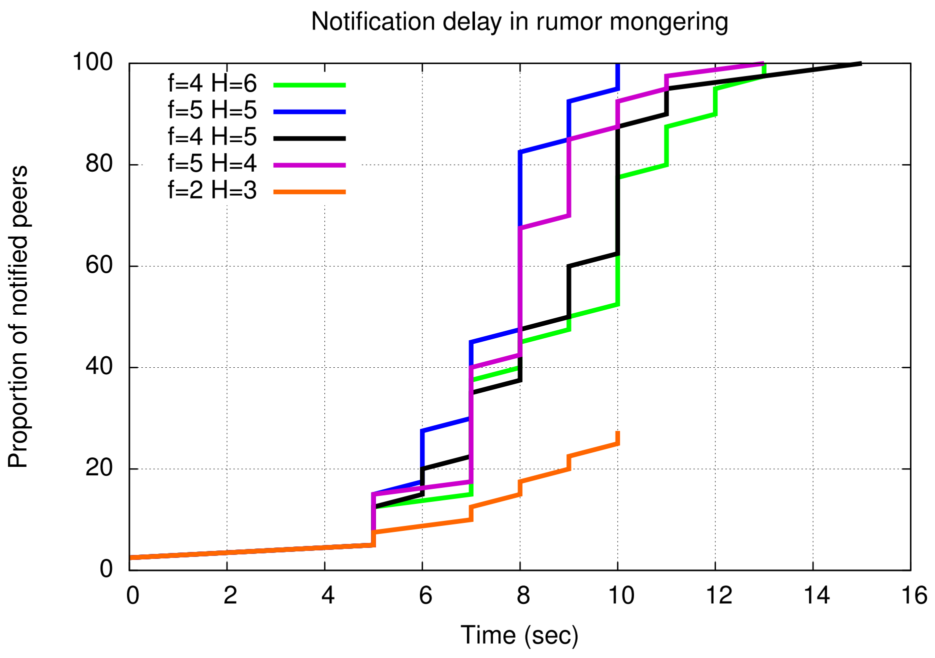

Figure 2: Proportion of infected nodes over time using rumor mongering and different f/HTL values.

For all parameter settings, the graph in Figure 2 starts flat: during the first 5 seconds, all have only the first node infected, which then initiates the first spread of the message. It is then clearly recognizable how the number of infected nodes increases by f for all at the same time: the green and the black line by 4, the blue and the purple by 5.

The blue line then shows the greatest increase in nodes at each second, followed by the purple line. This makes sense since they both spread to the greater number of peers at each exchange. Differences, such as the purple line not increasing at 6 seconds, can be explained by the initial offsets and nodes receiving duplicate.

The number of notified nodes per second peaks after around 8-10 seconds. This is when most messages are “on the run”, still have sufficient hops left, and random peer selection returns many not yet infected peers. After this peak, the number of notified nodes per second decreases again, due to the fact that then many nodes are already infected and duplicates are sent.

The blue line is the first to finally infect all nodes after 10 seconds. It is followed at 13 seconds by both the purple line - which infects more neighbors at the same time, but has messages reaching the HTL value and not being spread further; and the green line, which infects slower due to its smaller f value, but does not yet run out of hops. Finally, the black line is the last to have infected all nodes, flattening towards the end since many messages that are sent are duplicates. The orange line clearly demonstrates how messages with too low f and HTL run out of hops before reaching all nodes, and dissemination stops early on.

Higher f values, as would have been assumed, lead to a faster spread of information and are therefore useful for distributing more critical updates. On the other hand, in a dynamic network, a slower spread of information may be desired in order to take account of possibly joining nodes, so that messages remain active for a longer time by setting lower f and higher HTL values.

1.3 Comparison and Combination of Mechanisms

Due to the combination with anti-entropy, all nodes will eventually be infected with the combined protocol, such that lower values for f and HTL, such as f=2 and HTL=3, are theoretically sufficient (although for rumor mongering alone, they would not be able to infect all nodes). However, in order to be able to compare the runs, the settings for rumor mongering (alone) and the combined protocol are at f=5 HTL=4 in Figure 3.

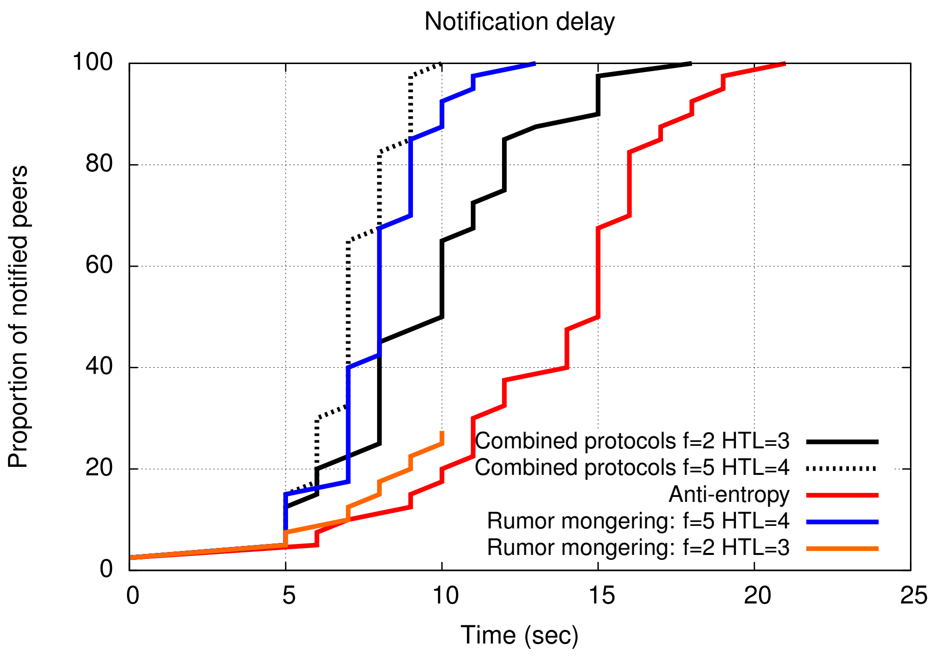

Figure 3: Infections over time using a combination of anti-entropy and rumor mongering, comparison of mechanisms.

1.3.1 Speed

Anti-entropy is slower, especially at the beginning, because it only infects one at a time (instead of f), but guarantees that eventually, all peers will be notified. For rumor mongering, the dissemination speed depends largely on the setting for the f parameter. The same goes for the combined protocol: With the same parameters as rumor mongering, the dissemination is just slightly faster than rumor mongering alone, thanks to the random infection at each period from anti-entropy. With lower values, we obtain a faster initial dissemination than anti-entropy, but then get closer to pure anti-entropy once the hops of the messages are consumed (after ca. 10 seconds, see orange line) and infections only spread randomly (consider red and black between seconds 5 and 15: diverging first, then approaching again at around 80% infection).

1.3.2 Completeness

The randomness of anti-entropy guarantees eventual completeness if the system is running for long enough (here, reaches completeness after ca. 22 seconds). For rumor mongering, it depends on the f/HTL settings. Due to the presence of anti-entropy in the combined mechanism, eventual completeness is also guaranteed and the difference is merely in the speed of reaching completeness (with higher values, dashed line, after 10 seconds; with lower, after around 18).

1.3.3 Duplicates

Anti-entropy has a large number of duplicates (exchanges between infected peers) and useless messages (exchanges between peers that are both not infected). For rumor-mongering, if guaranteeing completeness, the number of duplicates will also be high (nevertheless lower than anti-entropy, see Table 1).

The different f/HTL parameters in the combined protocol also yield a different number of duplicates (counting both the duplicates from anti-entropy and rumor mongering): over 120 seconds, 1749 total duplicates with f=2 HTL=3, and 2138 duplicates with f=5 HTL=4. Based on the theoretical formula, there is a difference of 766 messages between those two configurations. This difference is in reality lower, which can again be attributed to “lost” messages (arriving at one node at the same time).

A strategy in combining anti-entropy and rumor mongering would thus be to choose lower parameter values in order to reduce traffic, but have a higher number of sources of infection to start with, and then engage in anti-entropy behavior, possibly with less frequent exchange cycles. But again, it depends on the type of system and the urgency of the message.

2 Peer Sampling Service

Parameters for the *Blind* protocol are: `H=0, S=0, SEL=rand`. It removes at random nodes from the merged view until the view size is equal to c. Therefore, the generated overlay graph is completely random without any additional special properties. For the *Healer*, the parameters are `H=(viewsize/2)=4, S=0, SEL=rand`. The Healer discards old links in favor of newer ones. Finally the *Swapper* with settings `H=0, S=(viewsize/2)=4, SEL=tail` guarantees a great number of exchanged nodes (for even load distribution).

The current implementation of the PSS does not take into account the possibility of two peers selecting each other for exchanging their views at about the same time and the arising concurrency issues. A possible improvement to the code could be to lock the view while performing operations on it. Although these situations may arise, it is for now assumed that possibly resulting errors will be fixed by the protocol over time.

2.1 Partitions

Over a number of runs, the networks were connected. However, in general, since the nodes have no global knowledge of the system, they have no way of assuring that connected graphs occur. Randomness leads to the possibility of disconnection, but may be fixed at the next period since the disconnected node still has the other nodes in its view, so it is going to reinsert itself. The disconnected graph is a momentary snapshot of the graph. The different peer selection protocols lead to different risks for partitions: Since the Blind protocol has the least strategy, it has happened that a node had an in-degree of zero, though this is not the norm.

Initially, I had a bug in the code leading to a disconnected system. The view was being sorted by age in `selectToSend()`. Therefore, the youngest were always at the head and being removed, meaning the node kept only old links. solved by duplicating the view for sorting.

2.2 In-degree Distribution

Figure 4 shows for the various existing in-degrees the number of nodes having each value for the different protocols. For an even distribution of the load, a Gaussian distribution is desirable for the degrees.

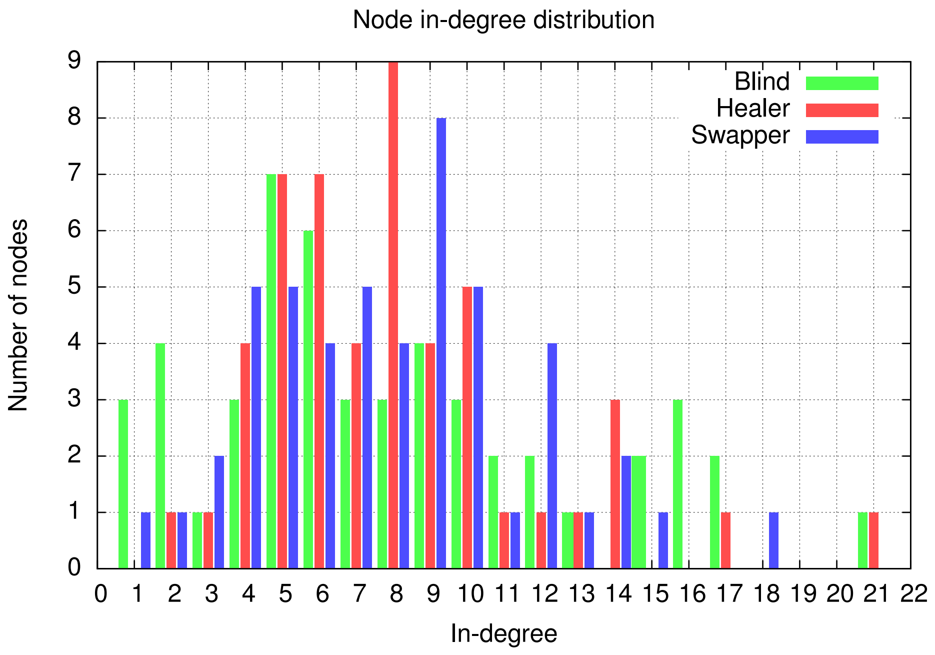

Figure 4: In-degree distribution.

We can see that Blind has the worst (= most random) distribution, with a greater number of peers having low in-degrees (1,2), comparatively few in the middle areas, and again some with many incoming links. Both Healer and Swapper get closer to the Gaussian distribution, although the Healer demonstrates some highly connected peers. The Swapper, from which the best distribution would have been expected, is only slightly closer to Gaussian. Its good properties would probably become more evident in a larger system after a longer timespan, but nevertheless it displays a very even distribution in the middle area (between 4 and 10).

2.3 Clustering

Clustering is not desired, since highly connected clusters are weakly connected to the rest of the system, leading to duplicate messages within the cluster and few leading outside. For best randomness, the clustering should thus be low and as equal as possible for all nodes. Ideally, the graph would be a flat line over all the nodes.

Although the random selection of nodes to keep in the view should generate a system of which the structure resembles a random graph, the *Blind* protocol, despite its randomness, lacks a strategy and therefore will not achieve random graph properties for clustering. The *Healer* protocol produces high clustering because it leads to frequent exchanges with the same peers by always discarding those with which there weren't any exchanges for a longer time. On the other hand, the *Swapper* protocol creates low clustering by maintaining the randomness of Blind, but exchanging with the oldest node, i.e. the one that has not been contacted by this peer or another

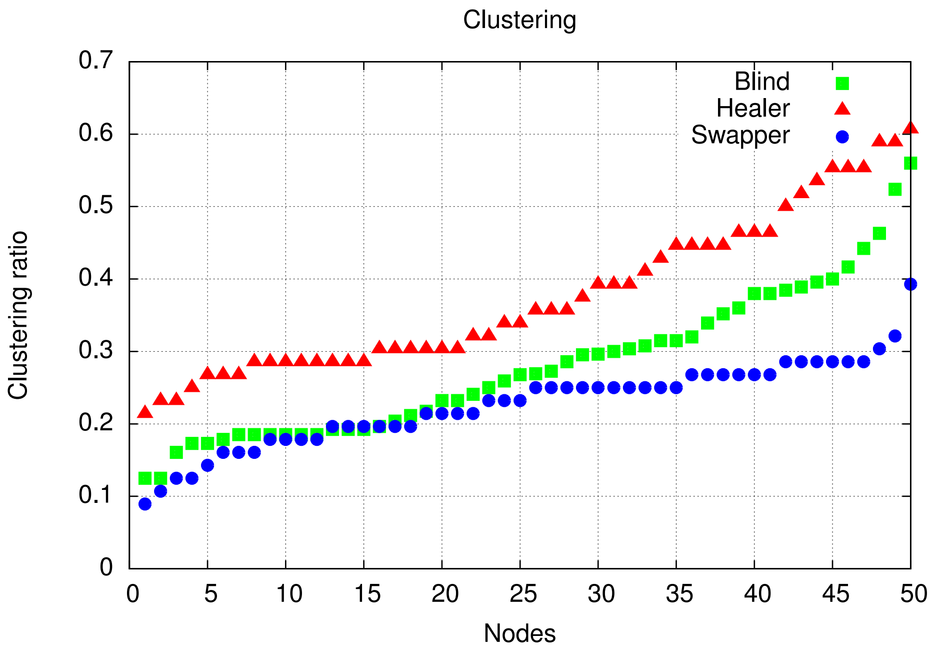

Figure 5: Clustering ratios over the peers.

peer from a previous exchanges in a long time. It is expected that the Swapper produces an even better randomness than the Blind.

The graph in Figure 5 clearly proves these assumptions. We can see that the Healer has by far the highest clustering over all nodes, and also a fairly uneven distribution with a difference of 0.39 between the lowest and the highest clustering ratio, an inclining function. The Blind protocol has overall lower clustering, but an even greater difference between the lowest and highest value at 0.43. Towards the right end, it has some peers that are almost as highly clustered as those in the Healer protocol.

As expected, we get the best clustering ratios for the Swapper. Aside from the extremes (with a difference of 0.30, so still lower than for the other protocols), we obtain an almost flat line, proving that its behavior of controlled exchanges of nodes with as many others as possible leads to good clustering. This allows for a fast dissemination of messages over the system with less duplicates.

2.4 Robustness to Failure and Churn

The robustness of the peer sampling service for both the Healer and Swapper protocols under the presence of churn was checked using the file `massive_fail.churn_trace`.

Unsuccessful calls are first put into "wait" mode and then periodically aborted, which is why the graphs only start to peak ca. 40 seconds after half of the peers quit (at 180 seconds); and why the graph displays periodic peaks.

To demonstrate how the views are repaired, the ruby script `pss_check_partition.rb` was modified (`pss_check_partition_fail.rb`) to include checking whether the peers in the views are still available. If not, it prints a message to the output.

Running this script verifies the graph in Figure 6. While the Swapper does not repair itself and keeps making unsuccessful calls to its peers, the Healer goes back to normal quickly. Finally on Healer, according to the script, there is no broken link to a failed node in any of the remaining

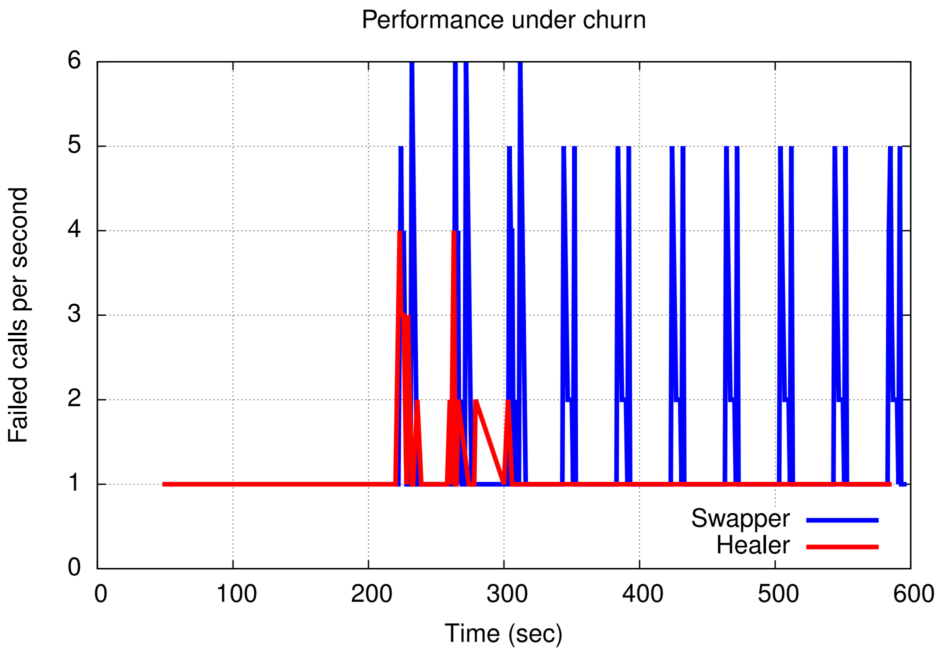

Figure 6: Unsuccessful calls per second with churn.

30 views. On the other hand, the Swapper maintains many failed nodes in the views: the 30 remaining peers (out of 60 initially) contain a total of 119 broken links. Considering they have a view size of 8, this means that 50% of all peers in the views have failed (= exactly the percentage of peers that have failed in total); the network has not repaired itself. Also, the 4 peers are not connected to the system (though they might reinsert themselves provided they still have valid links in their views.)

3 Combination of Gossip-Based Dissemination and Peer Sampling Service

Finally, the properties and advantages of the PSS were tested with the gossip dissemination protocols from section 1. For this, an additional configuration was added for the PSS which is none of the extremes: combining Swapper and Healer (with parameters H=2 and S=2) for a system that has better self-healing properties, while at the same time constructing properties closer to a random graph. Figure 7 compares the notification delays of the various protocols.

The first dissemination period is the same as the time of infection of the first node; this is why, unlike the previous graphs which started after 5 seconds, dissemination starts at 0 in this graph.

What can be seen is that the Healer protocol, and the combined Healer/Swapper, are the last to infect all peers. This can be attributed to the clustering of the healer protocol, which leads to slower dissemination. Especially towards reaching all peers, at over 80%, both flatten strongly because the few remaining peers are weakly connected and harder to reach (but still, eventually are reached, so we have no partitioned overlay networks for any of the protocols). The Swapper starts out about as slowly as the two protocols with Healer components, but then does not flatten towards the end and instead maintains its speed of dissemination.

While the Blind protocol, due to its randomness, is initially the fastest in infecting a large

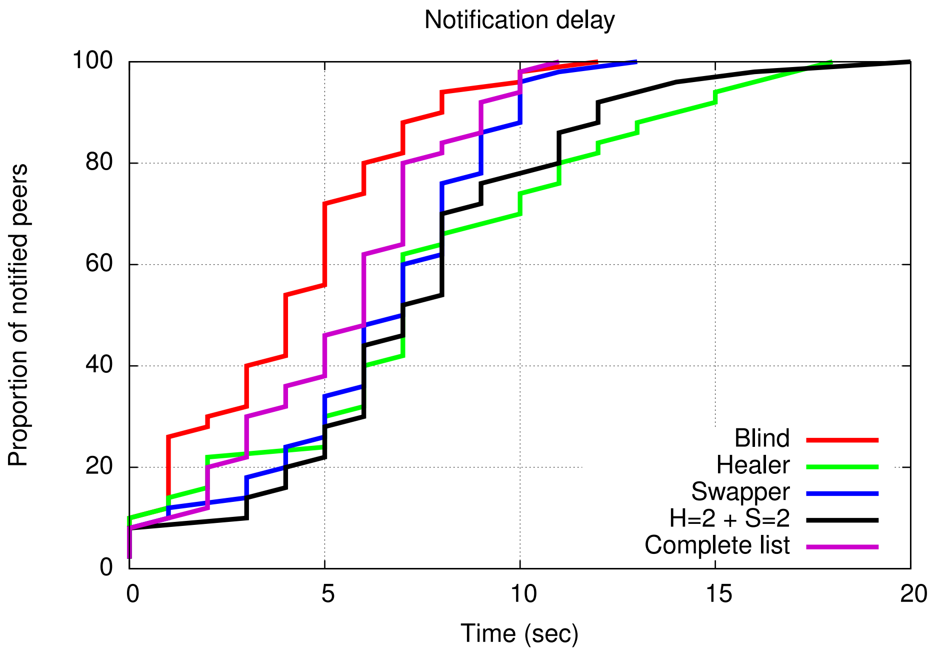

Figure 7: Binary message dissemination using the PSS with different configurations, and the complete list.

number of peers (followed by the complete list, which has similar properties), finally it completes at about the same time as the Swapper and the complete list. The complete list is by little the first to infect all.

Duplicates

Duplicates are mostly interesting from the rumor mongering, since anti-entropy will always yield a high number of duplicates (for example, even when an infected node contacts a non-infected node, it receives the infection again itself). Therefore, the duplicates are considered separately.

- Blind: 1548 duplicates from anti-entropy; 5 duplicates from rumor mongering.
- Healer: 1519 duplicates from anti-entropy; 6 duplicates from rumor mongering.
- Swapper: 1500 duplicates from anti-entropy; 7 duplicates from rumor mongering.
- H+S: 1498 duplicates from anti-entropy; 5 duplicates from rumor mongering.
- Complete list: 1535 duplicates from anti-entropy; 6 duplicates from rumor mongering.

As we can see, the number of duplicates do not vary greatly between the different protocols. In terms of useful/useless traffic consumption, they are about equal. However again, a lower exchange period for anti-entropy may be beneficial.