Scaling up publish/subscribe overlays using interest correlation for link sharing (supplemental document)

Miguel Matos, *Member, IEEE,* Pascal Felber, *Member, IEEE,* Rui Oliveira, *Member, IEEE,* José Pereira, *Member, IEEE,* and Etienne Rivière, *Member, IEEE*

Abstract—This document provides further information on StaN, a decentralized protocol that optimizes the properties of gossip-based overlay networks for topic-based publish/subscribe by sharing a large number of physical connections without disrupting its logical properties [1]. The design and evaluation of the dissemination protocol leveraging StaN is described in Appendix A. In Appendix B we compare StaN with a greedy-omniscient approach and finally in Appendix C we further evaluate StaN in dynamic scenarios with message loss and churn.

Appendix A Dissemination

StaN maps several logical links to a single physical link to improve resource usage. This is possible due to the non-negligible probability of having overlap among node interests, as observed in many real scenarios [2]–[4]. Moreover, further studies point out that not only node interests overlap but also the occurrence of messages posted to multiple topics, a phenomena known as crossposting, is non-negligible [5], [6]. For instance, on Usenet at least 30% of the messages are crossposted and the average crossposted message targets 3 topics [5], [6]. Consequently, a crosspost-aware dissemination protocol may be able to reduce bandwidth usage by combining crossposted messages with StaN's link sharing.

In the remaining of this section we analyze the design of such a protocol, which we call CrosspostFlood. For simplicity it is an infect and die flooding protocol, i.e., the first time it receives a message it relays it to all neighbors on the given topic(s). The only assumption is access to the list of topics a message is posted to, which can be easily included as metadata.

The basic idea is very simple and depends only on local knowledge: when the topics of a crossposted message (or a subset of it) matches a mapping of logical links to a physical one, only a single message copy is sent through the physical link. Upon reception, it suffices to deliver that message to the relevant topics.

As an example, suppose node n_0 on the bottom of Figure 2 in [1] receives (or creates) a message m tagged with topics A and B. Node n_0 needs to relay the message to neighbors n_2 and n_3 to topic A and do the same to topic B. Instead of sending two copies of m through each logical link to each neighbor, n_0 sends a single copy to either logical link. Upon reception of m, n_2 and n_3 detect that it has been posted to topics A and B (by observing m's metadata) and locally deliver m to topics A and B, effectively reducing the number of messages in transit from four to two. It is important to note that, although independent from the link alignment promoted by StaN, the dissemination is most effective when combined with crosspost detection. For instance, the exact same run on a non optimized version of the overlays (top of Figure 2 in [1]) would not bring any bandwidth savings.

The pseudo-code for CrosspostFlood is shown in Algorithm 3. As previously, we assume the existence of a network-level primitive $send$ and that each node p maintains one view per overlay t, denoted by $p.views[t]$. The node's topics are accessed by $p.topics$, and $p.receive(m)$ corresponds to the delivery of a message m to topic t (see architecture in Figure 1 in [1]).

To avoid delivery of duplicates each node maintains a set of previously known messages, $receivedMessages$, initially empty (lines 1–2).

A message m is generated with a unique identifier, $msgId$, and the set of topics it belongs to, $msgTopics$.

Upon reception of a message (MSG), a node first checks if the message is new by observing the set of known message identifiers, and discarding it otherwise (lines 4–5). The message is then delivered to the node's topics that match the message topics, $msgTopics$ (lines 7–8). The delivery is done by invoking the receive method for each matching topic t (Figure 1 in [1]). Additionally, for each matching topic t the node collects the identifiers of its neighbors in each topic in a set called $relayNodes$ (lines 9–10).

• *M. Matos, R. Oliveira and J. O. Pereira are with INESC TEC and the University of Minho, Braga, Portugal.*
P. Felber and E. Rivière are with the Computer Science Department, University of Neuchâtel, Switzerland.
Contact: `miguelmatos@di.uminho.pt`

Algorithm 3: CrosspostFlood protocol (node p)

```
1  initially
       // Contains received message identifiers, to avoid duplicates
2      $receivedMessages \leftarrow \emptyset$

   // Message reception
3  upon receive MSG($msgId, msgTopics, msgData$)
4      if $msgId \notin receivedMessages$ then
5          $receivedMessages \leftarrow receivedMessages \cup \{msgId\}$
           // Set that will contain the nodes to forward the message to
6          $relayNodes \leftarrow \emptyset$
7          foreach topic $t \in p.topics \cap msgTopics$ do
               // Deliver message to topic t
8              t.receive($msgId, msgData$)
               // Collect the nodes subscribed to topic t
9              foreach node $n \in p.views[t]$ do
10                 $relayNodes \leftarrow relayNodes \cup \{n\}$
           // Relay the message
11         foreach node $n \in relayNodes$ do
12             send MSG($msgId, msgTopics, msgData$) to $n$
```

Finally, the message is relayed to this set of neighbors as usual. By first collecting the neighbors that a message needs to be relayed to in a set, and only then sending it effectively, we eliminate possible duplicates (i.e. a node that is a neighbor in two topics) thus avoiding redundant transmissions.

A.1 Evaluation

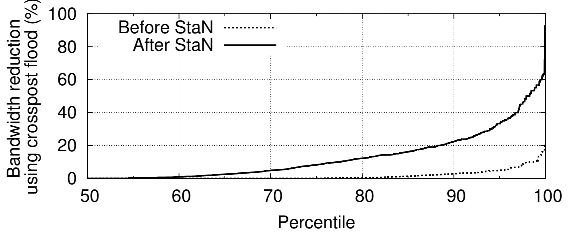
As StaN maintains the desirable properties for gossip-based dissemination (Section 3 in [1]) we now study the behavior of the CrosspostFlood dissemination protocol. This is done by analyzing bandwidth usage, in terms of number of messages exchanged, and latency, in terms of hops.

Based on real observations where, on average, each crossposted message targets 3 topics [5], [6], we devised a simple workload to compare the effectiveness of CrosspostFlood with a baseline infect and die flooding protocol, which we call SimpleFlood.

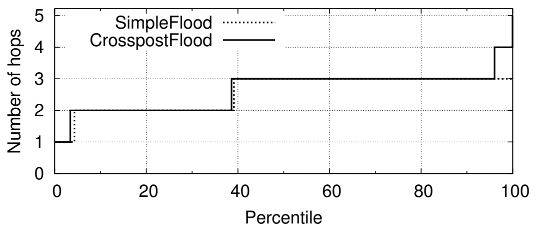
The dissemination is done on all overlays as follows: for each topic T, we select T' and T'' as the most correlated topics with T. Next, we randomly pick 10 nodes subscribed to T, T' and T'' and have each of them inject a new message on the system tagged with the triplet (T, T', T''). This is done for each of the 10 independent runs of StaN analyzed before. Results presented are the average of 100 independent runs (10 disseminations for each of the 10 runs) for both CrosspostFlood and SimpleFlood in the $\mathcal{L}_8$ universe.

Figure 10(a) presents the bandwidth reduction when using CrosspostFlood and SimpleFlood before and after the optimizations performed by StaN. Results are obtained by calculating the ratio between the number of messages sent by each node using CrosspostFlood against SimpleFlood. Thus, a value of X% means that CrosspostFlood sent less X% messages overall than SimpleFlood.

It is important to note that on the worst case (no crossposting or no link sharing) CrosspostFlood degenerates to SimpleFlood. As expected, CrosspostFlood reduces the number of messages sent (the ratio is positive) thus saving bandwidth. This is more evident when disseminating after the optimizations made by StaN as there are more logical links mapped to the same physical link thus enabling further reductions. For instance, without StaN's optimizations the amount of nodes able to save (reduce) more than 10% when using CrosspostFlood is negligible. On the other hand, when using optimized overlays, more than 20% of the nodes are able to achieve reductions greater than 10% when using CrosspostFlood.



(a) Bandwidth reduction distribution.



(b) First delivery hops distribution.

Fig. 10: SimpleFlood vs CrosspostFlood on universe $\mathcal{L}_8$: a) bandwidth reduction before and after optimizing the overlays with StaN b) hops necessary for first delivery.

At a local level, these savings are interesting as the cost is negligible: nodes only need to check if several logical links map to the same physical link. To assess the cost at a global level, we need to measure the latency, in terms of number of hops, needed to infect all nodes. This is because reductions in bandwidth typically tend to negatively affect latency.

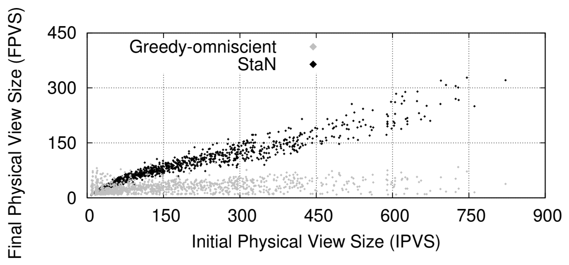
Figure 10(b) shows the hop count distribution for the reception of new messages on StaN optimized overlays. As observed, hop-counts are almost unaffected and even reduced in some situations. This is because when a crossposted message is received on a given topic, it is immediately delivered and relayed to all the relevant topics which acts as a shortcut to the normal per-topic relaying process.

A.2 Discussion

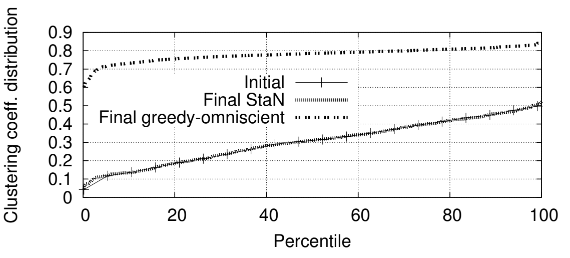
The crosspost-aware nature of CrosspostFlood when combined with the physical link sharing obtained by StaN enables improved resource usage in terms of bandwidth at virtually no local or global cost. We note that these savings are only possible due to link sharing, otherwise it degenerates to a simple flooding dissemination protocol. Moreover, CrosspostFlood is not specific to StaN as it may be combined with other protocols that promote link sharing such as SpiderCast [7].

The results are interesting because the improvements are obtained at virtual now cost. Nonetheless, a deeper analysis of this protocol and its combination with link sharing protocols like StaN is needed, namely by considering more complex workloads and other phenomena, such as message re-crossposting, i.e. when a node receives a message on a topic and locally decides to repost it on another topic.

APPENDIX B GREEDY-OMNISCIENT COMPARISON



(a) View improvement.



(b) Clustering coefficient distribution.

Fig. 11: Comparison of view improvement and clustering coefficient distribution for StaN and a greedy-omniscient approach for $\mathcal{W}_8$. Lines "Initial" and "Final StaN" overlap in Figure 11(b).

StaN promotes link sharing by relying only on local knowledge and without explicitly considering node interests. We now compare StaN against a greedy-omniscient implementation with global knowledge that explicitly takes into account node interests to try to maximize link sharing. Our goal is to compare StaN's performance with an approach based on global knowledge and at the same assess the impact on fitness of optimizing to an inherently clustered metric, the node's interests. The greedy-omniscient implementation works as follows: each node sorts all other known nodes according to the most topics in common (by computing the cardinality of the intersection of its subscriptions with the other node's subscriptions); and then picks the $viewSize$ best ones, where $viewSize$ is computed as in Section 3.3 in [1]. Because nodes have global knowledge, the node's local choices are the best possible but, as in typical greedy approaches, there is no guarantee that the global solution is optimal. We note that this strategy is similar to SpiderCast [7] with global knowledge and with the random selection disabled ($K_r = 0$).

Figure 11(a) depicts the IPVS vs FPVS for both StaN and the greedy-omniscient implementation. Each point in the scatter plot represents the IPVS and FPVS for each node. As expected, the greedy-omniscient implementation outperforms StaN in terms of PVS reduction. This is because the greedy-omniscient optimization criteria is precisely the view size whereas StaN's criteria is a weight metric unrelated to the view size. Nonetheless, the absolute reduction in PVS obtained by StaN is still considerable. For instance, for an IPVS of 600 StaN achieves a reduction of around 400. The tradeoff is increased clustering because by optimizing to the view size, the overlay tends to approximate the inherent subscriptions clustering, as observed in Figure 11(b). Note that lines *Initial* and *Final StaN* overlap indicating that StaN's impact on clustering is negligible.

B.1 Discussion

The comparison with a greedy-omniscient approach shows that optimizing to an inherently clustered metric, the node's interests, implies, as expected, an increase in the clustering of the overlay because neighbor selections are no longer random. Such high clustering negatively affects the robustness and dissemination efficiency of the overlays [8] and is thus an inherent limitation of approaches based on selecting neighbors according to subscription overlap.

Despite the greater improvement on performance of the greedy-omniscient implementation with respect to StaN, we believe StaN presents an interesting compromise between performance and fitness introducing thus a new point in the design space of topic-based publish-subscribe systems.

APPENDIX C DYNAMICS

In this section we study the behavior of StaN under conditions likely to emerge in large-scale scenarios, namely: message loss, node churn and growing scenarios where nodes continue to join after the initial bootstrap.

C.1 Message Loss.

We analyze the behavior of StaN under message loss by observing its convergence speed under increasing loss rates. Results can be observed in Figure 12. As expected, convergence speed is slowed down by message loss but StaN is still able to converge under moderate message loss rates. For instance, for a loss rate of 10%, the convergence at round 9 is almost

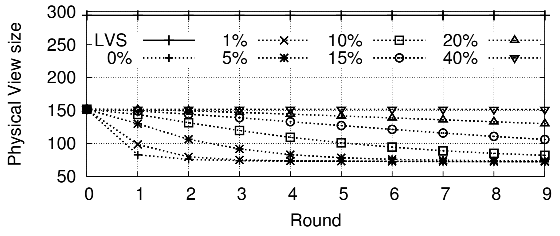


Fig. 12: View evolution under message loss for universe $\mathcal{W}_8$ (percentages indicate message loss rates).

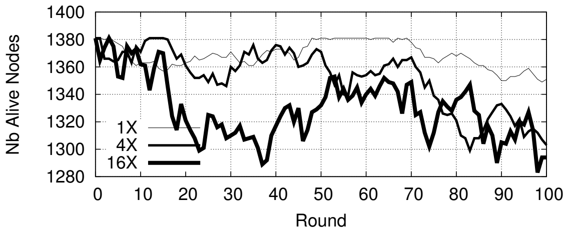
indistinguishable from a loss-free environment. As a matter of fact, only with loss rates greater than 15%, do we observe that the convergence speed is too slow to be useful.

C.2 Node churn.

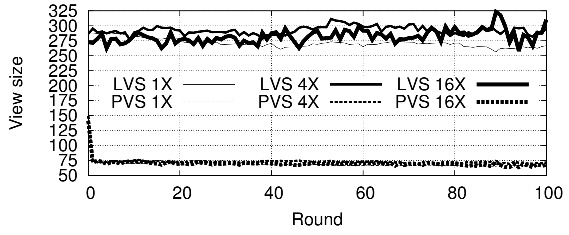
We now study StaN's behavior under node dynamics by reproducing a churn trace gathered from the Overnet network [9]. For each run, we generate a trace with 1381 nodes ($\mathcal{W}_8$'s universe size) and map 60 seconds of the trace time to a cycle, adding and removing nodes as appropriate. We experiment with higher churn rates by *speeding up* the trace by a given factor, i.e., mapping a longer trace time to each cycle. For instance, mapping 120 seconds to a cycle yields a factor of $2X$. Figure 13 presents the evolution of the universe size (top) and PVS and LVS (bottom) for factors 1, 4 and 16. As expected, increasing the churn rate increases the magnitude and amplitude of the variations in the node population (Figure 13(a)). The same behavior is observed for the LVS and PVS which grow and shrink as the node population variates. For higher churn rates, we observe that a few nodes (less than 5 in all the experiments) got isolated from the overlay. The reason is that the view size evolves only due to churn, decreasing when neighbors fail and increasing as new nodes join. In some high churn cases, failures in the vicinity of one node are enough to depopulate the view without being compensated by joins, disconnecting the node from the overlay. We address this issue by triggering a random walk (Algorithm 2 in [1], COLLECTWALK()) to add new links, when the view size is smaller than a given threshold (5 in our experiments). This simple modification avoids disconnections even under high churn rates. We note that the decision to modify the view size and add links when necessary is typically the responsibility of the OMP, we do this here to focus exclusively on StaN's behavior.

C.3 Growing universe.

Finally, we study the behavior of StaN under a considerable universe growth. We start by randomly selecting 50% of the $\mathcal{W}_8$ nodes and running StaN on that sub-universe. Then, every 10 rounds we add 10% of the remaining nodes until all nodes are in



(a) Universe size evolution under churn for universe $\mathcal{W}_8$.



(b) LVS and PVS evolution under churn for universe $\mathcal{W}_8$.

Fig. 13: Universe and View evolution under churn for universe $\mathcal{W}_8$. (Numbers represent the churn speedup factor.)

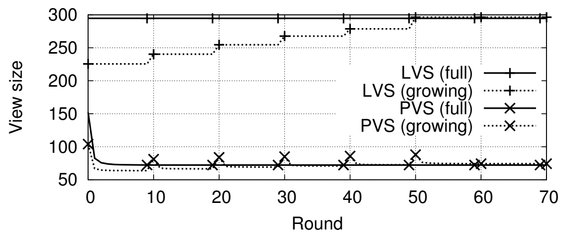


Fig. 14: View evolution for growing $\mathcal{W}_8$ universe.

the system. Results are presented in Figure 14. When adding nodes, the PVS grows quickly to accommodate the new nodes which is then reduced by StaN in a few rounds. Most interestingly, some rounds after the universe is fully grown (round 60), both LVS and PVS are almost indistinguishable from a universe fully bootstrapped from scratch.

C.4 Discussion

The decentralized and gossip-based nature of StaN allows it to perform well on adverse scenarios with message loss, node churn and growing universes.

Although not considered in this paper, robustness to these adverse conditions can be improved by leveraging messages received on the COLLECTWALK(). This would allow nodes to leverage random walks initiated by other nodes and thus be more robust to message loss as every message relayed enables the discovery of new nodes which also improves convergence time. The same applies to node churn and growing universes allowing nodes to react quicker to changes.

REFERENCES

[1] M. Matos, P. Felber, R. Oliveira, J. Pereira, and E. Rivière, "Scaling up publish/subscribe overlays using interest correlation

for link sharing," *(TO BE COMPLETED FOR THE CAMERA READY)*, vol. –, –.

[2] P. Fraigniaud, P. Gauron, and M. Latapy, "Combining the use of clustering and scale-free nature of exchanges into a simple and efficient P2P system," in *International Conference on Parallel and Distributed Computing*, 2005.

[3] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proceedings of Multimedia Computing and Networking*, 2002.

[4] S. Handurukande, A.-M. Kermarrec, F. Le Fessant, L. Massoulié, and S. Patarin, "Peer sharing behaviour in the eDonkey network, and implications for the design of server-less file sharing systems," *ACM Eurosys*, 2006.

[5] S. Whittaker, L. Terveen, W. Hill, and L. Cherny, "The dynamics of mass interaction," in *Conference on Computer supported cooperative work*, 1998.

[6] M. McGlohon, "Structural analysis of large networks: Observations and applications," Ph.D. dissertation, Carnegie Mello University, 2010.

[7] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication," in *International Conference on Distributed Event-Based Systems*, 2007.

[8] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems*, vol. 25, no. 3, aug 2007.

[9] R. Bhagwan, S. Savage, and G. Voelker, "Understanding availability," in *Proc. of IPTPS: international workshop on Peer-to-Peer Systems*, Feb. 2003.