Emergent Structure in Unstructured Epidemic Multicast[*]

Nuno Carvalho — U. Minho
José Pereira — U. Minho
Rui Oliveira — U. Minho
Luís Rodrigues — U. Lisboa

Abstract

In epidemic or gossip-based multicast protocols, each node simply relays each message to some random neighbors, such that all destinations receive it at least once with high probability. In sharp contrast, structured multicast protocols explicitly build and use a spanning tree to take advantage of efficient paths, and aim at having each message received exactly once. Unfortunately, when failures occur, the tree must be rebuilt. Gossiping thus provides simplicity and resilience at the expense of performance and resource efficiency.

In this paper we propose a novel technique that exploits knowledge about the environment to schedule payload transmission when gossiping. The resulting protocol retains the desirable qualities of gossip, but approximates the performance of structured multicast. In some sense, instead of imposing structure by construction, we let it emerge from the operation of the gossip protocol. Experimental evaluation shows that this approach is effective even when knowledge about the environment is only approximate.

1 Introduction

Epidemic multicast protocols, also known as probabilistic or gossip-based, operate by having each node relay every message to a set of neighbors selected at random [2, 5, 16, 13]. The procedure is repeated for a number of rounds such that the message is delivered to all destinations with high probability. As neighbors are uniform randomly chosen, the load is balanced among all nodes: Over time, all nodes send and receive approximately the same number of messages. This fails to take advantage of nodes and links with higher capacity. In addition, a large number of redundant message transmissions happen, leading to a poor latency/bandwidth tradeoff.

In sharp contrast, structured multicast protocols work by explicitly building a dissemination structure according to predefined efficiency criteria [8, 3, 19, 18, 23], and then use it to convey multiple messages. Therefore, nodes with higher capacity can offer a bigger contribution to the global dissemination effort by having larger degrees or being assigned closer to the root of the tree. Nodes in the leaves of the tree are not required to contribute to the message dissemination effort.

The tradeoff between gossip and structured approaches is clear: By avoiding the need to build and maintain a spanning tree, epidemic multicast provides extreme simplicity. The resulting evenly balanced load is a key factor to achieve resilience and scalability. On the other hand, structured multicast provides better resource usage (and thus higher performance when the network is stable) by optimizing the cost of the spanning tree according to efficiency criteria such as network bandwidth and latency. However, structured approaches have to deal with the complexity of rebuilding the tree when faults or network reconfiguration occurs.

In this paper, we aim at combining the best of both approaches, namely, the simplicity, scalability and resilience of epidemic multicast with the performance of structured multicast. The challenge is to incorporate efficiency criteria in a gossip protocol without affecting its probabilistic properties. Our current proposal builds on the ideas first introduced in [15], which point out that the combination of lazy and eager push gossip can promote an asymmetrical resource usage in

[*]This work was partially supported by project "P-SON: Probabilistically Structured Overlay Networks" (POS_C/EIA/60941/2004). Parts of this report were published in the Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. Edinburgh, UK, June, 2007.

a gossip protocol. By combining this technique with knowledge about the environment, one can make transmissions that lead to message deliveries to use preferably the nodes and links that match a target efficiency criteria. In some sense, instead of imposing a multicast structure by construction, we let this structure to emerge probabilistically from the operation of the gossip protocol, without altering the original gossip pattern.

The main contributions of the paper are the following: *(i)* it proposes a technique to exploit knowledge about the network topology to obtain an emergent structure that results in actual performance improvements in the data dissemination process, and *(ii)* it offers an extensive evaluation of the proposed technique in a realistic network setting.

The rest of this paper is structured as follows. Section 2 presents the rationale underlying our proposal. Section 3 proposes an epidemic multicast protocol that enables emergent structure according to a configurable strategy module. Section 4 introduces several sample strategies. Section 5 presents the experimental environment, used in Section 6 to evaluate the approach. Section 7 compares our proposal with related work. Finally, Section 8 concludes the paper.

2 Overview

2.1 Background

Gossip-based multicast protocols are often based on an *eager push gossip* approach [5, 16, 13]: A gossip round is initiated by a node that has received a message, relaying it to a number of targets. This simple strategy has however a large impact on bandwidth used, as the fanout required for atomic delivery leads to multiple copies of each message being received by each destination.

A different tradeoff can be achieved by using a *lazy push* strategy, which defers the transmission of the payload. In detail, during a gossip round a node will send only an advertisement of the new message. Transmission of the payload is initiated only if the message is unknown by the recipient. This allows message payload to be transmitted once to each destination, at the expense of an additional round-trip. Lazy transmission has also an impact on the reliability, as the additional round-trip and resulting increased latency widens the window of vulnerability to network faults. The impact is however small for realistic omission rates and can be compensated by a slight increase in the fanout [15]. The net effect is still a much lower bandwidth usage.

In fact, one can mix both approaches in a single gossiping round [15], thus providing different latency/bandwidth tradeoffs depending on how many messages are eagerly transmitted.

2.2 Approach

The approach proposed in this paper stems from the observation that, in an eager push gossip protocol, paths leading to deliveries of each message implicitly build a random spanning tree. This tree is embedded in the underlying random overlay. If one knew beforehand which transmissions would lead to deliveries, one could use eager push gossip for those paths and lazy push gossip for all others. This would achieve exactly once transmission for each destination. Unfortunately, this is not possible, as one cannot predict which paths lead to delivery.

There is however one alternative strategy that is feasible: If one of the embedded trees is selected beforehand for eager push gossip, one increases the likelihood that this tree will lead to more deliveries. This happens because lazy push has additional latency, and paths that use it will be outran by paths that use solely eager push. If one assigns nodes and links with higher capacity to such tree, the performance of the protocol should approach that of a structured overlay. Note that by keeping redundant lazy transmissions, one retains the resilience of the gossip protocol. On the other hand, this strategy requires the explicit maintenance of a tree structure which imposes additional overhead and complexity.

Instead of selecting a single embedded tree, in this paper we aim at increasing the likelihood that implicitly created trees in a gossip protocol include nodes and links with higher capacity. Structure is therefore probabilistic: Nodes and links are selected with different probabilities for

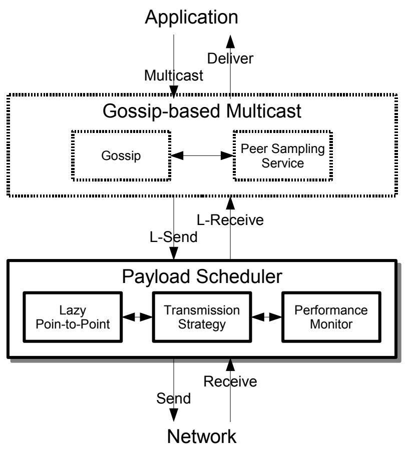


Figure 1: Protocol architecture overview.

payload transmission. Therefore, structure emerges from the strategy used for scheduling message payloads in a combined eager/lazy push gossip protocol. The main challenge is thus to achieve emergent structure without global coordination, while at the same time obtaining a meaningful performance advantage.

3 Architecture

The architecture that we propose to implement our approach is depicted in Fig. 1. It uses an additional layer that is inserted below a pure eager push gossip protocol. This layer, called the Payload Scheduler, selects when to transmit the message payload (by using a combination of eager push and lazy push) in a transparent manner for the gossip protocol above. The Payload Scheduler layer can be decomposed into three separate components, also depicted in Fig. 1:

- The Lazy Point-to-Point module is in charge of intercepting the interaction between the gossip layer above and the transport protocol below. It queries the Transmission Strategy module to decide whether to send the payload immediately (in the case, the exchange is performed in pure eager push mode) or to delay the payload transmission until a request is received. As we will later describe, this module is also in charge of generating and replying to payload requests.

- The Transmission Strategy module is the core component of the Payload Scheduler. It defines the criteria that is used to defer payload transmission at the sender and, at the receiver, when to request a transmission. Note that different strategies may be implemented, according to the criteria that one wants to optimize. We recall that, with each strategy, we aim at having the combined protocol (push gossip plus scheduler) to approximate the behavior of a structured multicast approach.

- The last component of the Payload Scheduler is a Performance Monitor. The role of the performance monitor is to extract, in run-time, performance data about the operation of the system, for instance, by computing round-trip delays. The performance data is used to feed the Transmission Strategy module.

In the remainder of this section, we will describe each component of our architecture in detail as well as all interfaces among them.

```
1   initially
2       $K = \emptyset$    /* known messages */
3   proc Multicast($d$) do
4       Forward(MkId(), $d$, 0)
5   proc Forward($i, d, r$) do
6       Deliver($d$)
7       $K = K \cup \{i\}$
8       if $r < t$ do
9           $P$ = PeerSample($f$)
10          for each $p \in P$ do
11              L-Send($i, d, r + 1, p$)
12  upon L-Receive($i, d, r, s$) do
13      if $i \notin K$ then
14          Forward($i, d, r$)
```

Figure 2: Basic gossip protocol.

3.1 Gossip Protocol Layer

As noted before, a fundamental aspect of our approach is that the Payload Scheduler can operate in a manner that is transparent for the operation of the push gossip protocol that lies above. Therefore, our approach can be applied to different gossip protocols, such as [5, 13, 16].

Nevertheless, for self containment, we depict in Fig. 2 a typical push gossip protocol. This implementation assumes the availability of a peer sampling service [10] providing an uniform sample of f other nodes with the PeerSample(f) primitive. It assumes also an unreliable point-to-point communication service, such that a message m can be sent to a node p using the L-Send(m, p) primitive. A message m is received from a peer p by handling the L-Receive(m, p) up-call. The gossip protocol maintains a set K of known messages (line 2), initially empty. This set is used to detect and eliminate duplicates. In detail, the algorithm works as follows.

- The application calls procedure Multicast(d) to multicast a message with payload d (line 3). This simply generates an unique identifier and forwards it (line 4). The identifier chosen must be unique with high probability, as conflicts will cause deliveries to be omitted. A simple way to implement this is to generate a random bit-string with sufficient length.

- Received messages are processed in a similar manner (line 12), with the difference that it is necessary to check for and discard duplicates using the set of known identifiers K (line 13) before proceeding.

- The forwarding procedure Forward(i, d, r) (line 5) uses the message identifier i, the payload d and the number of times, or rounds, the message has already been relayed r, which is initially 0. It starts by delivering the payload locally using the Deliver(d) up-call. Then the message identifier is added to the set of previously known messages K (line 7). This avoids multiple deliveries, as described before. Actual forwarding occurs only if the message has been forwarded less than t times (line 8) [13] and consists in querying the peer sampling service to obtain a set of f target nodes and then sending the message, as in lines 9 and 11. Constants t and f are the usual gossip configuration parameters [6].

For simplicity, we do not show how identifiers are removed from set K, preventing it from growing indefinitely. This problem has been studied before, and efficient solutions exist ensuring with high probability that no active messages are prematurely garbage collected [5, 13].

3.2 Payload Scheduler Layer

The Lazy Point-to-Point module is the entry point to the Payload Scheduler. It controls the transmission of message payload using a simple negative acknowledgment mechanism. The policy

```
initially
    $\forall i : C[i] = \perp$    /* cached data */
    $R = \emptyset$    /* known messages */
Task 1:
    proc L-SEND$(i, d, r, p)$ do
        if EAGER?$(i, d, r, p)$ then
            SEND(MSG$(i, d, r, p)$
        else
            $C[i] = (d, r)$
            SEND(IHAVE$(i), p)$
    upon RECEIVE(IHAVE$(i), s)$ do
        if $i \notin R$ then
            QUEUE$(i, s)$
    upon RECEIVE(MSG$(i, d, r), s)$ do
        if $i \notin R$ then
            $R = R \cup \{i\}$
            CLEAR$(i)$
        L-RECEIVE$(i, d, r, s)$
    upon RECEIVE(IWANT$(i), s)$ do
        $(d, r) = C[i]$
        SEND(MSG$(i, d, r),\ p)$
Task 2:
    forever do
        $(i, s) =$ SCHEDULENEXT()
        SEND(IWANT$(i), s)$
```

Figure 3: Point-to-point communication.

used for each individual message is obtained from the Transmission Strategy module using a pair of primitives:

- EAGER?(i, d, r, p) is used to determine if payload d for message with identification i on round r should be immediately sent to peer p. Note that if the method always returns true the protocol operates on pure eager push mode. If the method always returns false, the protocol operates on pure lazy push mode.
- $(i, s) =$ SCHEDULENEXT() blocks until it is the time for some message i to be requested from a source s. From the correctness point of view, any schedule is safe as long as it eventually schedules all lazy requests that have been queued.

The Lazy Point-to-Point module also informs the Transmission Strategy of known sources for each message and when payload has been received using the following primitives:

- QUEUE(i, s) queues a message identifier i to be requested from source node s. The Transmission Strategy module must keep an internal queue of known sources for each message identifier in order to schedule them eventually, unless payload is received first.
- CLEAR(i) clears all requests on message i. Note also that a queue eventually clears itself as requests on all known sources for a given message identifier i are scheduled.

The Lazy Point-to-Point module is depicted in Fig. 3 and uses two separate tasks. Task 1 is responsible for processing transmission requests from the gossip layer and message deliveries from the transport layer. Task 2 runs in background, and performs requests for messages that are known to exists but whose payload has not been received yet. Furthermore, the module maintains the following data structures: a set R of messages whose payload has been received and; a map C, holding the payload and round number for the message (if known).

The module operates as follows. When a message is sent (line 19), the Transmission Strategy module is queried to test if the message should be immediately sent (line 21). If not, an advertisement without the payload is sent instead (line 24). Upon receiving a message advertisement for an unknown message, the Transmission Strategy module is notified (line 27). Upon receiving full message payload, the strategy module is informed (line 31) and it is handed over to the gossip layer (line 32).

Task 2 executes the following loop. The Transmission Strategy module is invoked to select a message to be requested and a node to request the message from (pair (i, s) in line 38). This invocation blocks until a request is scheduled to be sent by the Transmission Strategy module. A request is then sent (line 39). Finally, when a node receives a request (line 33) it looks it up in the cache and transmits the payload (line 35). Note that a retransmission request can only be received as a consequence of a previous advertisement and thus the message is guaranteed to be locally known.

For simplicity, we again do not show how cached identifiers and payloads are removed from C and R, preventing them from growing indefinitely. This is however similar to the management of set K, discussed in the previous section, and thus the same techniques apply.

Finally, the goal of the performance monitor module is to measure relevant performance metrics of the participant nodes and to make this information available to the strategy in an abstract manner. The exported interface of this module is simply Metric(p), that returns a current metric for a given peer p. This metric is used by the Transmission Strategy to select whether to immediately schedule an eager transmission or when to request lazy transmission from each source. Note that, the performance monitor module may be required to exchange messages with its peers (for instance, to measure roundtrip delays).

The next section discusses different implementations of the Transmission Strategy and of the Performance Monitor modules that aim at achieving different dissemination structures.

4 Strategies and Monitors

The definition of a Transmission Strategy has two main objectives: *(i)* avoid as much as possible the redundant transmission of the same payload to any given target node; and *(ii)* decrease the latency of message delivery. These goals are however conflicting. The first goal can be achieved by using a lazy push strategy in all peer exchanges. Since nodes only gossip IHAVE commands, the recipient can request the payload only once. Unfortunately, each lazy push exchange adds one additional roundtrip to the final delivery latency. On the other hand, a pure eager push strategy minimizes latency at the cost of adding a significant amount of redundancy.

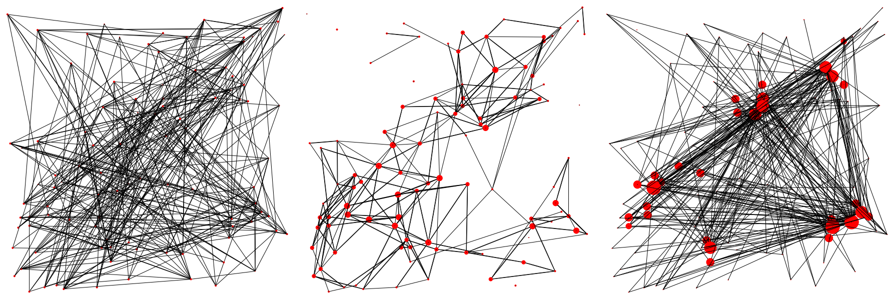
The key to obtaining a better latency/bandwidth tradeoff is thus to select nodes and links that should be preferred in a decentralized fashion. We start however by proposing a couple of strategies that do not take advantage of knowledge about the environment for use as a baseline.

4.1 Strategies

Flat This strategy is defined as Eager?(i, d, r, p) returning true with some probability π or false with probability $1 - \pi$. When $\pi = 1$, this defaults to a fully eager push gossip. With $\pi = 0$, it provides pure lazy push gossip. In between, it provides different latency/bandwidth tradeoffs, as a different share of gossip messages is handled in a lazy fashion.

When a lazy strategy is used (and IHAVE messages are sent), we need also to consider how retransmissions are scheduled within ScheduleNext() by receivers. In the Flat strategy, the first retransmission request is scheduled immediately when queued, which means that an IWANT message is issued immediately upon receiving an IHAVE advertisement. Further requests are done periodically every T, while sources are known.

Time T is an estimate of maximum end-to-end latency. This avoids issuing explicit transmission requests until all eager transmissions have been performed, thus optimizing bandwidth. Note that, unless there is a network omission or an extreme transmission delay, there is usually no need to



(a) Flat (7% of traffic). (b) Radius (37% of traffic). (c) Ranked (30% of traffic).

Figure 4: Emergent structure defined by the top 5% most used connections.

issue a second request. Thus the value of T has no practical impact in the final average latency, and can be set only approximately.

Time-To-Live (TTL) This strategy uses eager push until some round u and is thus defined as $\text{EAGER?}(i, d, r, p)$ returning true iff $r < u$. When $u > t$, this defaults to common eager push gossip. With $u = 0$, it provides pure lazy push gossip. In between, it provides different latency/bandwidth tradeoffs, as a different share of gossip messages is handled in a lazy fashion. $\text{SCHEDULEDNEXT}()$ is defined exactly as in the Flat strategy.

We propose this because it is intuitively useful: During the first rounds, the likelihood of a node being targeted by more than one copy of the payload is small and thus there is no point in using lazy push.

Radius This strategy is defined as $\text{EAGER?}(i, d, r, p)$ returning true iff $\text{METRIC}(p) < \rho$, for some constant radius ρ and some metric $\text{METRIC}(p)$ provided by a Performance Monitor module for node p. $\text{SCHEDULEDNEXT}()$ delays the first retransmission by some time T_0 that is an estimate of the latency to nodes within radius ρ in the given metric. Further retransmissions are scheduled periodically with period T, as in the Flat strategy. However, if multiple sources are known, the nearest neighbor (according to the Performance Monitor) is selected.

This follows the intuition of gossiping first with close nodes to minimize hop latency. The expected emergent structure should approximate a mesh structure, with most of payload being carried by links between neighboring nodes.

Ranked This strategy aims at achieving a hubs-and-spokes structure by selecting a set of *best nodes* to serve as hubs, bearing most of the load. Therefore, at some node q, $\text{EAGER?}(i, d, r, p)$ returns true iff either q or p are considered to be best nodes, meaning that eager push is used whenever a best node is involved. $\text{SCHEDULEDNEXT}()$ is defined exactly as in the Flat strategy.

Although some nodes can be explicitly configured as *best nodes*, for instance, by an Internet Service Provider (ISP) that wants to improve performance to local users, a ranking can also be computed using local Performance Monitors and a gossip based sorting protocol [11]. As shown later, this is greatly eased by the fact that the protocol still works even if ranking is approximate.

4.2 Monitors

Latency Monitor This monitor measures the latency to all neighbor nodes. Real-time monitoring of latency has been addressed a number of times, in fact, every TCP/IP connection implicitly estimates round-trip time in order to perform congestion control [7]. This estimate can be retrieved by applications and used for other purposes.

Distance Monitor This monitor measures geographical distance to all neighbor nodes. This is useful mostly for demonstration purposes, as it allows us to plot network usage graphs such that emergent structure is understandable by the reader. Otherwise, it is not useful in optimizing network parameters.

4.3 Approximation and Noise

When evaluating the proposed protocol on the ModelNet emulated network as described in the following sections, the proposed strategies and monitors are simplified by relying on global knowledge of the network that is extracted directly from the model file. This has two goals: First, it allows us to separate the performance of the the proposed strategy from the performance of the monitor. Second, it allows us to arbitrarily introduce noise in Transmission Strategy in order to evaluate its robustness.

In detail, we intercept each query to Eager? and use a temporary variable v as follows. If Eager? would return true, we set $v = 1.0$; if it would return false we set $v = 0.0$. We then compute $v' = c + (v - c)(1 - o)$. We then generating a random boolean with probability v' of being true as the outcome of the query. Constant c is set such that the overall probability of Eager? returning true is unchanged, thus leading to the same amount of eager transmissions although scheduled in different occasions.

When noise ratio $o = 0.0$, the original result is unchanged. When noise ratio $o = 1.0$, any Transmission Strategy defaults to Flat with $\pi = c$, thus completely erasing structure. In between, the Transmission Strategy produces an increasingly blurred structure.

5 Experimental Environment

5.1 Network Emulation

Experimental evaluation of the proposed protocol is done using the ModelNet large-scale emulation infrastructure [21] with a realistic network model generated by Inet-3.0 [22]. In detail, ModelNet allows a large number of virtual nodes running unmodified programs to be configured in a smaller number of physical nodes in a LAN. Traffic is routed through emulator nodes thus applying delay, bandwidth, and loss as specified in the network model. Inet-3.0 generates realistic Autonomous System level network topologies using a transit-stub model.

ModelNet is deployed in a cluster of 5 workstations connected by switched 100Mbps Ethernet. Each workstation has a 2.4GHz Intel Celeron CPU, 512MB RAM, and a RealTek Ethernet controller. When hosting virtual nodes, they run Linux kernel 2.6.14 and IBM Java2 1.5 runtime. When running as an emulator, FreeBSD 4.11 is used.

The network model is generated using Inet-3.0 default of 3037 network nodes. Link latency is assigned by ModelNet according to pseudo-geographical distance. Client nodes are assigned to distinct stub nodes, also with the default 1 ms client-stub latency. A typical network graph has the following properties: average hop distance between client nodes is 5.54, with 74.28% of nodes within 5 and 6 hops; average end-to-end latency of 49.83 ms, with 50% of nodes within 39 ms and 60 ms.

5.2 Implementation and Configuration

The proposed protocol was implemented by modifying an open source and lightweight implementation of the NeEM protocol [16] that uses the java.nio API for scalability and performance [20]. Briefly, NeEM uses TCP/IP connections between nodes in order to avoid network congestion. When a connection blocks, messages are buffered in user space, which then uses a custom purging strategy to improve reliability. The result is a virtual connection-less layer that provides improved guarantees for gossiping.

This implementation was selected as NeEM 0.5 already supports eager and lazy push, although the later is selected only based on a message size and age threshold. The change required was to remove the hard-coded push strategy and insert the scheduler layer. Message identifiers are probabilistically unique 128 bit strings.

The protocol was configured with gossip fanout of 11 and overlay fanout of 15. With 200 nodes, these correspond to a probability 0.995 of atomic delivery with 1% messages dropped, and a probability of 0.999 of connectedness when 15% of nodes fail [6]. A retransmission period of 400 ms was used, which is the minimal that results in approximately 1 payload received by each destination when using a fully lazy push strategy.

5.3 Traffic and Measurements

During each experiment, 400 messages are multicast, each carrying 256 bytes of application level payload. To each of them, a NeEM header of 24 bytes is added, besides TCP/IP overhead. Messages are multicast by virtual nodes in a round-robin fashion, with an uniform random interval with 500 ms average. All messages multicast and delivered are logged for later processing. Namely, end-to-end latency can be measured when source and destination share the same physical node, and thus a common clock. Payload transmissions on each link are also recorded separately.

Results presented in the following sections used 25 virtual nodes on each workstation, thus obtaining 100 virtual nodes. The reason for this limitation is that an epidemic multicast protocol produces a bursty load, in particular when using eager push gossip: Network and CPU load occurs only instants after each message is multicast. Using a larger number of virtual nodes was observed to induce additional latency which would falsify results. The configurations that result in lower bandwidth consumption, which are the key results of this paper, were also simulated with 200 virtual nodes.

5.4 Statistics

Consider the following statistics of each experiment with 100 virtual nodes using an eager push strategy: 40000 messages delivered, 440000 individual packets transmitted. This amounts to 2200 Kpkts/s and thus approximately 6 MBps. As the overlay evolves, TCP/IP connections are created and tear down. During each run, approximately 550 simultaneous and 15000 different connections are used. The experiments presented in the next sections, when automated, take almost 7 hours to run. The total amount of resulting logs is 1Gb, that has then to be processed and rendered in plots.

Care was taken to consider variance of each measure taken. When in the following sections we affirm that a performance difference is relevant, this was confirmed by checking that confidence intervals with 95% certainty do not intersect. In fact, the large number of samples used are sufficient to make such intervals very narrow.

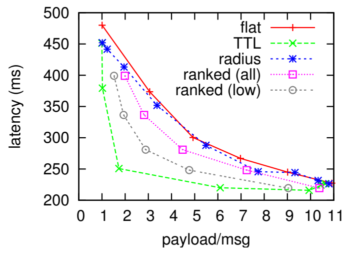
6 Experimental Results

6.1 Pseudo-Geographical

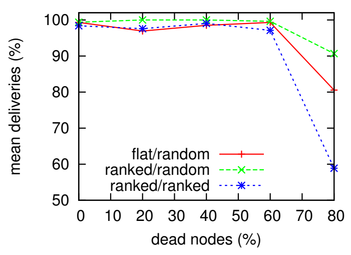
To strengthen the intuition, we start by evaluating the impact of the proposed strategies with an oracle that considers pseudo-geographical position of nodes, as generated by Inet-3.0. Although this cannot be used to assess the performance of the protocol, as geometrical distance does not directly map to end-to-end distance, it allows the resulting emergent structure to be plotted and understood.

Fig. 4 shows the result of running 100 node configurations with different strategies and then selecting the top 5% connections with highest throughput. The size of each red circle is proportional to the amount of payload transmitted by the node. Note that each connection is used for a brief period of time, as the membership management algorithm periodically shuffles peers with neighbors. This means that connections shown may have not existed simultaneously.

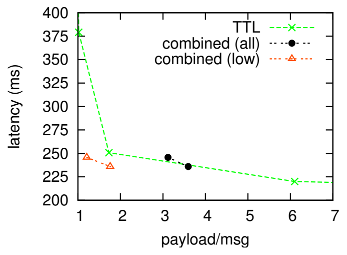
As a baseline, Fig. 4(a) shows an eager push configuration, where no structure is apparent. A confirmation of this is given by the fact that the top 5% connections account for only 7% of



(a) Latency/bandwidth tradeoff.



(b) Average deliveries.



(c) Hybrid strategy.

Figure 5: Performance.

all traffic, i.e. payload transmissions are evenly spread across all connections. In sharp contrast, Fig. 4(b) shows an obvious emergent mesh structure as a result of the Radius strategy, in which the 5% connections account for 37% of all payload transmissions. Finally, Fig. 4(c) shows a sub-set of nodes emerging as super-nodes, accounting for a large share of links and also transmitting a higher number of payloads. Again, the emergent structure is confirmed by the fact that 5% of the connections account for 30% of total payloads transmitted.

6.2 Latency vs Bandwidth

The primary evaluation criterion is the latency/bandwidth tradeoff provided by each configuration. Assuming an eager push protocol, one expects that each payload is approximately transmitted f times for each delivery. This makes gossip protocols notorious for wasted bandwidth. On the other hand, assuming that one uses a lazy push approach, one expects on average a single payload transmitted for each messages delivery at the expense of an additional round-trip, thus impacting latency.

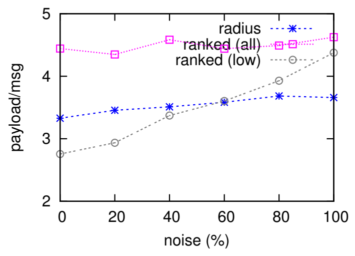
The measurements of strategy Flat in Fig. 5(a) illustrate this tradeoff. Latency can be improved from 480 ms down to 227 ms by increasing parameter π. This does however increase the number of payloads transmitted for each message delivered from the optimal 1 up to 11, the fanout value used. A much better tradeoff can be obtained with the TTL strategy, which achieves 250 ms with 1.7 transmitions of payload for each message delivered.

Considering that a large majority of links is between 39 ms and 60 ms (with mean close to 50 ms), and that each message is delivered on the average after being gossiped 4.5 times, it is not realistic to expect that link selection provides an improvement beyond a 45 ms. It is therefore interesting to observe that the Ranked strategy does indeed improve average latency when compared to the Flat strategy that produces a similar amount of traffic. On the other hand, the Radius strategy does not produce such improvement. This is explained by noting that the reduced round latency is compensated by an average larger number of rounds required.

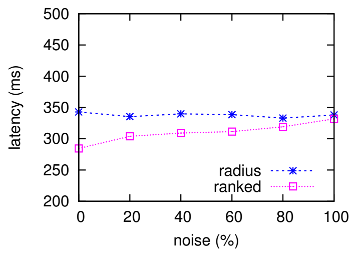
6.3 Reliability

We are interested in confirming that the proposed approach does not impact reliability. We simulate failed nodes by silencing them with firewall rules after letting them join the overlay and warm up, *i.e.* immediately before starting to log message deliveries.

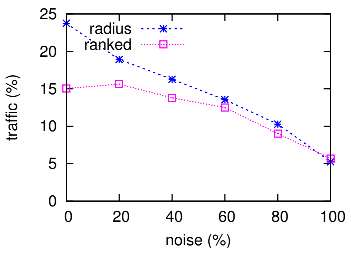
As a baseline in Fig. 5(b), we fail an increasing number of nodes selected at random while using a pure eager push configuration of the protocol. As expected, when no node fails one observes perfect atomic delivery of all messages. With 20% or more nodes failing, one observes that atomic delivery does not always happen, although as expected there is a large number of nodes that deliver each message. Finally, with more than 80% of nodes failing, the protocol breaks down as



(a)



(b)



(c)

Figure 6: Degradation of structure.

an arbitrary number of nodes get disconnected from the overlay.

Then we do the same with the Ranked strategy. We run experiments twice: First, we select failed nodes at random. Then we select the nodes with the best ranks, precisely those that are contributing more to the dissemination effort. As can be observed also in Fig. 5(b), there is no noticeable impact in reliability in both situations. With more that 80% of nodes failing, although there is an apparent difference, the observed high variance makes it impossible to conclude on a which configuration is the best.

6.4 Hybrid Strategy

A major advantage of the proposed approach is that one can easily try new strategies without endangering the correctness of the protocol. Namely, this can be used to try complex heuristics. We do this now by trying to leverage the contribution of TTL, Radius, and Ranked in a single hybrid strategy. In detail, $\text{EAGER?}(i, d, r, p)$ returns true iff one of the involved nodes is considered a best node; or if $\text{METRIC}(p) < 2\rho$ when $r < u$; or $\text{METRIC}(p) < \rho$ otherwise. i.e. radius shrinks with increasing round number. Parameters ρ and u are the same as in Radius and TTL strategies. $\text{SCHEDULEDNEXT}()$ is defined exactly as in the Radius strategy.

The result is shown in Fig. 5(c), which provides a new interesting tradeoff for regular nodes: Latency can be reduced from 379 ms to 245 ms while increasing average payload to 80% of nodes from 1.01 to 1.20 payload transmissions for each message (see “combined (low)” in Fig. 5(c)). This is achieved at the expense of a contribution of 10.77 payload/message by the remaining 20% (overall average of 3.11). Eager gossip would require an overall average of 11 payload/message to achieve 227 ms.

6.5 Noise

Finally, we evaluate the protocol when the accuracy of the performance monitoring module deteriorates. As described in Section 4.3, we do this in a way that carefully preserves the amount of data transmitted. This can be confirmed in Fig. 6(a). Note however that the amount of data transmitted by regular nodes (i.e. other than the *best nodes*) increases with the amount of noise until it converges with the overall average.

Fig. 6(b) shows the impact of degrading structure in latency of the Ranked strategy, as Radius does not provide a latency advantage. Finally, Fig. 6(c) uses the amount of data conveyed by the top 5% links as a measure of emergent structure. It can be observed that it converges to the expected 5% of traffic as noise increases, thus showing that structure is increasingly blurred.

7 Related Work

Reliable multicast in large scale networks has been addressed a number of times using a different approaches. Some proposals build on IP multicast, and have thus been hard to deploy [8]. More recently, a number of application-level multicast protocols have been proposed, namely, by building on scalable structured overlay networks [3, 19, 18, 23]. It would be interesting to compare the performance of such protocols with the proposed approach, as well as the amount of code required to implement them. Nevertheless, by relying on explicit garbage collection to ensure reliability, these approaches should be vulnerable to the throughput stability problem [1].

The first reliable multicast protocol to exploit gossiping in order to provide simultaneously atomic delivery and throughput stability is Bimodal Multicast [2]. This protocol does however rely on an optimistic dissemination phase to avoid a large number of redundant transmissions. This makes it hard to scale it to wide area networks, as the low adoption of IP multicast in WAN has shown.

It has also been proposed that instead of using a random overlay, one can build a mesh according to some configurable metric using gossiping itself [14]. In fact, it has been pointed out a number of different structures can be built using gossiping [9]. These proposals share with ours the requirement of a performance monitor, but do not preserve the random nature of an unstructured overlay which is key to reliability [10]. Similar techniques can however be used to derive knowledge about the environment for use by the Transmission Strategy module.

Lazy push gossip can also be confused with pull gossip, as in both cases payload is transmitted only upon request. Pull gossip is however fundamentally different as it issues generic requests to a random sub-set of nodes, which might or not have new data, while lazy push gossip requests specific data items only from peers that have previously advertised them. In fact, unless performed lazily, pull gossip will result in multiple payload transmissions to the same destination as much as eager push gossip. A combination of eager push and eager pull gossip has been proposed as a means to reduce propagation latency [12]. A combination of lazy pull gossip and lazy push gossip has been proposed in the CREW protocol for fast file download by a large number of destinations [4], where the latency of lazy gossip is hidden by concurrently requesting multiple chunks of a large file.

Adaptation to heterogeneous network resources can also be done by adjusting node fanout according to available bandwidth [17, 4] and by biasing choices during gossiping and overlay maintenance [17]. Although in the current paper we have always used unmodified gossip to demonstrate viability, combining both approach should allow further improvements.

8 Conclusions

Epidemic multicast protocols are notorious for being resilient and scalable, but also for generating a large number of redundant message transmissions. In this paper, we have shown that by carefully scheduling the transmission of payload in a combined eager/lazy push gossip protocol, one can reduce the bandwidth required without otherwise impacting performance or reliability. Namely, we propose an heuristic configuration that achieves close to optimum bandwidth and latency in a large majority of participants.

Most interestingly, when we examine the resources used, we observe that those nodes and links on which most payload transmissions are scheduled emerge as a probabilistic structure embedded in an otherwise unstructured overlay. This shows that one can approximate the performance advantages of structured overlays while at the same time preserving the simplicity, resilience and scalability of gossip.

Furthermore, our evaluation shows that the resulting performance degrades gracefully as the knowledge about the environment becomes less accurate, thus demonstrating the robustness of the approach. In fact, the worst case scenario in which such knowledge is entirely random, the performance is bounded by the original pure lazy or eager push protocols.

Note also that, although best results are achieved when all nodes cooperate on a single strategy, correctness is ensured regardless of the strategy used by each peer. This makes the proposed approach a promising base for building large scale adaptive protocols, given that its operation does not require tight global coordination.

Acknowledgments

The authors thank Bruno Matos at the U. Minho for his work on setting up the experimental environment.

References

[1] K. Birman. A review of experiences with reliable multicast. *Software Practice and Experience*, 29(9), July 1999.

[2] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Computer Systems*, 17(2), May 1999.

[3] Y. Chu, S. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), 2002.

[4] M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, and Sharad Mehrotra. CREW: A gossip-based flash-dissemination system. In *Intl. Conf. Distributed Computing Systems (ICDCS)*, 2006.

[5] P. Eugster, R. Guerraoui, S. Handrukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *Proc. IEEE Intl. Conf. Dependable Systems and Networks (DSN)*, 2001.

[6] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. From epidemics to distributed computing. *IEEE Computer*, May 2004.

[7] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Trans. Networking*, 7(4), August 1999.

[8] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5, December 1997.

[9] M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay topology management. In *Proc. 3rd Intl. Ws. Engineering Self-Organising Applications (ESOA'05)*. Springer-Verlag, 2005.

[10] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *Proc. 5th ACM/IFIP/USENIX Intl. Conf. Middleware*, 2004.

[11] Márk Jelasity. A case study on gossip beyond gossip: Sorting. Ws. on Gossip Based Computer Networking, Lorent Center, Leiden, 2006.

[12] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *IEEE Symp. Foundations of Computer Science*, 2000.

[13] B. Koldehofe. Buffer management in probabilistic peer-to-peer communication protocols. In *Proc. IEEE Symp. Reliable Distributed Systems (SRDS)*, 2003.

[14] L. Massoulié, A.-M. Kermarrec, and A. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *Proc. IEEE Symp. Reliable Distributed Systems (SRDS'03)*, 2003.

[15] J. Pereira, R. Oliveira, and L. Rodrigues. Efficient epidemic multicast in heterogeneous networks. In *OTM Workshops*, number 4278 in Lecture Notes in Computer Science, 2006.

[16] J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec. NeEM: Network-friendly epidemic multicast. In *Proc. IEEE Symp. Reliable Distributed Systems (SRDS)*, 2003.

[17] J. Pereira, L. Rodrigues, A. Pinto, and R. Oliveira. Low-latency probabilistic broadcast in wide area networks. In *Proc. IEEE Symp. Reliable Distributed Systems (SRDS'04)*, October 2004.

[18] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*, 2001.

[19] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication*. Springer-Verlag, 2001.

[20] P. Santos and J. Pereira. NeEM version 0.5. http://neem.sf.net, 2006.

[21] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[22] J. Winick and S. Jamin. Inet-3.0: Internet topology generator. Technical Report CSE-TR-456-02, University of Michigan, 2002.

[23] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video*, 2001.