Araneola: A scalable reliable multicast system for dynamic environments

Conference Paper · January 2004

DOI: 10.1109/NCA.2004.1347755 · Source: IEEE Xplore

2 authors:

Roie Melamed
IBM
22 PUBLICATIONS **451** CITATIONS
SEE PROFILE

Idit Keidar
Technion - Israel Institute of Technology
237 PUBLICATIONS **3,957** CITATIONS
SEE PROFILE

Araneola: A Scalable Reliable Multicast System for Dynamic Environments*

Roie Melamed†
IBM Haifa Research Laboratory

Idit Keidar‡
EE Department, Technion

*A preliminary version of this paper appears in Proceedings of the 3rd IEEE International Symposium on Network Computing and Applications (IEEE NCA04), August 2004, pp. 5–14.

†Roie Melamed, IBM Haifa Research Laboratory, Mount Carmel, Haifa 31905, Israel; email: roiem@il.ibm.com; phone: 972-52-6881255l; fax: 972-4-829-6114. **This work was done while the author was a graduate student in the Department of Computer Science, Technion - Israel Institute of Technology.**

‡Idit Keidar, Department of Electrical Engineering, Technion - Israel Institute of Technology, Technion City, Haifa, 32000, Israel; idish@ee.technion.ac.il

Abstract

This paper presents Araneola[1], a scalable reliable application-level multicast system for highly dynamic wide-area environments. Araneola supports multi-point to multi-point reliable communication in a fully distributed manner while incurring constant load (in terms of message and space complexity) on each node. For a tunable parameter $k \geq 3$, Araneola constructs and dynamically maintains a basic overlay structure in which each node's degree is either k or $k+1$, and roughly 90% of the nodes have degree k. Empirical evaluation shows that Araneola's basic overlay achieves three important mathematical properties of k-regular random graphs (i.e., random graphs in which each node has exactly k neighbors) with N nodes: (i) its diameter grows logarithmically with N; (ii) it is generally k-connected; and (iii) it remains highly connected following random removal of linear-size subsets of edges or nodes. The overlay is constructed and maintained at a low cost: each join, leave, or failure is handled locally, and entails the sending of only about $3k$ messages in total, independent of N. Moreover, this cost decreases as the churn rate increases.

The low degree of Araneola's basic overlay structure allows for allocating plenty of additional bandwidth for specific application needs. In this paper, we give an example for such a need — communicating with nearby nodes; we enhance the basic overlay with additional links chosen according to geographic proximity and available bandwidth. We show that this approach, i.e., a combination of random and nearby links, reduces the number of physical hops messages traverse without hurting the overlay's robustness as compared to completely random Araneola overlays (in which all the links are random) with the same average node degree.

Given Araneola's overlay, we sketch out several message dissemination techniques that can be implemented on top of this overlay. We present a full implementation and evaluation of a gossip-based multicast scheme with up to 10,000 nodes. We show that compared to a (non-overlay-based) gossip-based multicast protocol, gossiping over Araneola achieves substantial improvements in load, reliability, and latency.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**] Distributed Systems—*Distributed applications*

General Terms

Algorithms, Design, Measurement, Performance, Reliability

Keywords

Application-level multicast, dynamic wide-area environments, fault-tolerance, gossip, k-regular random graphs, load balancing, network proximity, peer-to-peer, reliability, scalability, unstructured overlay networks

[1] Araneola means "little spider" in Latin.

1 Introduction

Our goal is to provide a scalable multi-point to multi-point reliable multicast service for large groups in wide-area networks. Examples of applications that require such a service include publish-subscribe applications with a small number of topics [9, 12], distributed parallel processing [33], and collaboration applications such as shared document editing [26], whiteboards [36], chat, distributed interactive simulations [35], and multi-player games [1] [55]. Traditionally, IP multicast [20] has been advocated as a solution to scalable multicast. However, due to scalability, reliability, security, and congestion and flow control problems, nowadays IP multicast is mostly unavailable over the Internet [30, 45]. In recent years, Application Level Multicast (ALM) systems have emerged as a promising alternative to IP multicast for scalable wide-area multicast [6, 10, 13, 14, 16, 21, 24, 30, 32, 37, 42, 45, 47, 48, 59, 60]. In such systems, the multicast is supported at the application level, and hence these systems can be deployed over any network without requiring router support.

A protocol deployed in wide-area networks must be able to withstand frequent node failures as well as non-negligible message loss rates [52]. Moreover, studies have shown that, typically, users frequently join and leave multicast sessions [3]; such behavior is called *churn*. A major design goal for our work is therefore coping efficiently with churn. Specifically, we address the following challenges: (i) providing high reliability despite considerable message loss and failure rates while incurring constant load on each node; (ii) incorporating joining nodes and removing leaving (or failing) ones with a low *constant* overhead and within low latency; and (iii) providing an undisrupted service to nodes that are up despite high churn rates.

We present Araneola, a scalable reliable ALM system for dynamic wide-area environments. Araneola does not rely on any infrastructure such as dedicated servers nor requires router support. Reliability is achieved by constructing a richly-connected overlay and disseminating pertinent information on multiple *disjoint* paths in this overlay. The number of paths in the overlay can be tuned according to the expected failure and loss rates. Araneola is designed to incur small constant load on each node. To this end, it builds a basic overlay in which each node's degree is bounded by a small constant. Then, this basic overlay can be extended with additional links according to specific application needs, e.g., network proximity. This approach has three advantages: (i) all nodes, including low bandwidth ones, are capable of participating in the basic overlay; (ii) the load on all nodes is similar, so no user is required to contribute more bandwidth than its fair share for the basic overlay; and (iii) nodes have ample remaining bandwidth for connecting to additional nodes according to application needs.

Our search for a robust constant degree overlay leads us to consider k-regular random graphs. A *k-regular random graph* with N nodes is a graph chosen uniformly at random from the set of k-regular graphs with N nodes. In contrast to a normal random graph [11], where node degrees vary, in a k-regular graph, each node's degree is exactly k. For $k \geq 3$, a k-regular random graph is almost always a good expander [25], which implies that (i) its diameter grows logarithmically with N [65]; and (ii) it remains connected after random failures of a linear subset of its nodes and/or edges [27]. In addition, such a graph is generally k-connected, i.e., there are k *disjoint* paths between every two nodes in the graph[2]. In contrast, in order for a normal random graph to be even connected (with high probability), its average degree must be at least logarithmic in N [11]. Note that the diameter, which increases logarithmically, is the only feature of a k-regular random graph that depends on the system size. All the remaining characteristics of k-regular random graphs (connectivity, degree, robustness to random edge and node removals) are independent of N.

[2]The probability that a k-regular random graph is not k-connected is $O(N^{2-k})$.

We present for the first time an algorithm for constructing and maintaining an overlay that resembles a k-regular random graph in a distributed and efficient manner in dynamic settings. For a given parameter $k \geq 3$, Araneola's basic overlay converges to a graph in which each node has a degree of either k or $k+1$, and no two neighboring nodes have a degree of $k+1$. Empirically, we show that Araneola's overlay achieves the desired properties of k regular random graphs, namely logarithmic diameter, k-connectivity, and high robustness. In particular, we show that Araneola's overlay has a similar diameter and is as robust as graphs generated using a known centralized construction of k-regular random graphs. At the same time, Araneola's overlay construction algorithm is fully distributed and efficient, as each join or leave (or failure) incurs sending roughly about $3k$ messages in a k-degree overlay, regardless of N. Remarkably, in dynamic settings, the cost of handling a single join or leave operation *decreases* as the churn rate increases. This is in contrast to virtually all existing structured peer-to-peer overlays, with which the overhead for handling joins grows at least logarithmically with the number of nodes.

The low maintenance cost is achieved due to the facts that: (i) each join, leave, or failure is handled locally; and (ii) the selection of random neighbors uses partial membership views maintained by a distributed low cost membership service similar to the ones in [21, 59]. The overhead of the membership service is independent of the number of nodes and of the churn rate.

Many wide-area applications need low-diameter robust overlays. Beyond these general needs, different applications have additional specific needs such as communication with near-by nodes, proximity to content, and exploitation of bandwidth heterogeneity. We believe in separation of concerns between generic requirements on one hand, and needs that vary from application to application on the other. Araneola addresses the former using a low degree overlay, and thus leaves ample bandwidth for applications to address the latter. We illustrate this approach by extending Araneola's basic overlay with links added according to geographical proximity. We show that if each node has three random neighbors and the rest of its neighbors are chosen according to a geographical proximity metric, then the links in Araneola's overlay traverse substantially fewer physical hops on average compared to an Araneola overlay with the same average node degree in which all the links are random. Moreover, we show that if each node in the basic overlay is connected to as little as three or four random nodes, extending the basic overlay with links chosen according to geographical proximity creates an overlay that is as robust to random failures as a basic Araneola overlay with the same average degree, in which all the links are random.

Given Araneola's overlay, it is possible to multicast (broadcast) messages by simply flooding the overlay [47]. This approach yields low latency but also incurs fairly high overhead, as several duplicates of each message are sent to each node. This can be effective if data messages (i.e., payload messages sent by the application) are small, or if bandwidth is abundant, but otherwise, it is wasteful. Alternatively, message identifiers can be flooded instead of the data messages, and each node can request each message that it is missing from exactly one neighbor. This ensures that, in the absence of message loss, each data message is sent exactly $N-1$ times, although many duplicate messages carrying its identifier are sent.

In order to reduce the number of messages sent, one can bundle message identifiers together: each node can locally divide its time into *gossip rounds*, and send one *gossip message* to each of its neighbors in each round, where gossip messages include identifiers of recently received messages. Nodes can then request missing messages from other nodes that have them. This approach is appropriate for software update dissemination, video streaming, and file sharing applications like BitTorrent [18]. Note that with all of the above dissemination techniques, Araneola achieves full reliability of data delivery as long as there are no partitions in the overlay graph. Since, empirically, Araneola's overlay is an expander, Araneola achieves full reliability even under message loss and churn rates that are substantial higher than the ones measured

over the Internet.

We empirically evaluate Araneola with the latter (gossip-based) approach, and compare it to a gossip-based scalable reliable multicast protocol that is not based on an overlay network. Gossiping with Araneola differs from non-overlay-based gossip protocols in that with non-overlay-based gossip protocols (e.g., [19, 47]), each node chooses different random nodes to gossip with in each round, whereas in Araneola, each node always gossips with its neighbors in the overlay structure. We show that this difference leads to substantial improvements in load, reliability, and latency.

Contributions. In summary, this paper makes the following contributions:

- It presents the first efficient distributed algorithm for constructing and maintaining a graph structure that resembles a k-regular random graph and achieves its good properties in dynamic settings.
- It introduces an algorithm that constructs and maintains a richly-connected low degree overlay in which each join or leave operation incurs a constant overhead.
- It describes an overlay-based ALM system to provide an undisrupted multicast service in highly dynamic settings while incurring constant load on each node.
- It features a complete implementation and a thorough evaluation of Araneola running up to $10,000$ nodes on up to 125 machines, in both LAN and WAN, including extensive evaluation of the impact of churn on an ALM system.
- Finally, it constructs an overlay that designates ample bandwidth for each node to communicate with nodes chosen according to application needs, e.g., proximate nodes.

Roadmap. This paper proceeds as follows: Section 2 discusses related work. In Section 3, we summarize our design goals. Section 4 presents the design and pseudo code of Araneola's basic overlay, and Section 5 evaluates this overlay. Section 6 gives an example how Araneola's basic overlay can be enhanced with additional links chosen according to geographic proximity in order to reduce communication costs, and evaluates this extension over the Internet. Section 7 discusses multicasting over Araneola's overlay, and Section 8 evaluates a gossip-based multicast implementation. Section 9 concludes the paper.

2 Related Work

2.1 ALM Systems

In recent years, two leading approaches for supporting scalable ALM in dynamic failure-prone networks have emerged: gossip-based (or epidemic) multicast protocols, e.g., [10, 19, 21, 28, 40, 41, 42] (see Section 2.1.1), and dynamic overlay networks, e.g., [5, 14, 15, 16, 29, 32, 37, 53, 56, 62, 67] (see Section 2.1.2).

2.1.1 Gossip-based protocols

With gossip-based protocols, each node periodically chooses other random nodes to propagate the information to. Gossip-based protocols usually do not use any centralized infrastructure such as membership servers. Such protocols are highly robust in the presence of failures, and their reliability degrades gracefully

as failures amount [21, 38, 47]. They can achieve an average latency of $O(\log N)$ rounds [10, 21]. Moreover, they achieve good reliability (close to 100%) even in dynamic failure-prone settings. However, these protocols also have shortcomings. First, they generally require each node to send each message $O(\log N)$ times [42, 47], which induces a high load. Second, their reliability guarantees are probabilistic, and they generally provide less than 100% reliability even in static failure-free settings [47, 21]. Gossiping over Araneola eliminates these shortcomings: it has each node send information only k or k+1 times, and guarantees 100% reliability, assuming there is no message loss. This is since each node gossips with its overlay neighbors and all the nodes are organized in a highly-connected overlay (as opposed to a non-overlay-based gossip protocol). In addition, experimentally, even with a high message loss of about 10%, Araneola can still achieve perfect reliability. This is since in the Araneola overlay there are k-disjoint paths between every pair of nodes, and aggregated message identifiers are flooded on all of the overlay's links. We compare the performance of gossiping over Araneola with that of a non-overlay based gossip protocol in Section 8.1.3 below, and show that Araneola achieves higher reliability than the gossip protocol while incurring less overhead.

2.1.2 Tree/Mesh-based systems

Most overlay-based ALM systems are tree-based, e.g., [14, 16, 32, 37, 59]. With such systems, no duplicate messages are sent. If the tree topology is mostly stable, and loss-rates are low, then such systems can achieve great performance. However, in the presence of churn, the tree structure frequently becomes partitioned, causing a significant portion of the multicast messages to be lost. Therefore, in order to achieve reliability, such protocols need to detect message loss and recover from it. This can cause recovered messages to be significantly delayed; can induce substantial overhead, especially if failures are frequent; and can inhibit scalability. A second problem with tree-based multicast is uneven load distribution: as recently argued in [13], inner nodes in the tree carry almost all the burden for the multicast, whereas leaf nodes do not share the load.

Pbcast [10] combines best-effort tree-based dissemination, e.g., using IP Multicast, with gossip-based recovery. This approach has the advantages of tree-based ALMs, including fast dissemination and no duplicates in failure-free cases, as well as the robustness of gossip-based protocols. It is therefore effective if a stable tree-based multicast service is available. However, it is also hampered by the difficulty of maintaining stable trees in the presence of high churn.

Mesh-based overlay systems can achieve load balancing and robustness to failures and message loss by including multiple paths between every pair of nodes. SplitStream [13] constructs and maintains a forest of multicast trees, and evenly distributes the forwarding load among all participating nodes. In Bullet [45], nodes self-organize into an overlay mesh, and data packets are distributed to strategic points in the overlay. Nodes are then responsible for locating and retrieving the data packets. Whereas Araneola focuses on providing undisrupted service to nodes that are up despite high churn rates and considerable message loss and failure rates, SplitStream and Bullet are designed for content streaming. Therefore, neither SplitStream nor Bullet were evaluated under the high churn rates we evaluate Araneola. These two systems can induce high overhead: in SplitStream, each join event can incur sending $(k+1)\log(N)$ messages where k is the number of trees in the forest, and in Bullet the average per-node control overhead is approximately 30 Kbps. Moreover, in Bullet, roughly 10% of received data packets are duplicates. In contrast, in Araneola each join or leave operation incurs sending roughly $3k$ messages, and no duplicates are sent. Moreover, in order to achieve high reliability under high churn rates, these two systems need to either use forward error correction

techniques, which incur additional overhead, or to employ heavy buffering, which incurs high delay and requires additional disk space. Finally, these two systems are intended for single-source multimedia transfer and do not support multi-point to multi-point reliable communication as we do.

In the Yoid project [24], nodes auto-configure into two topologies: a shared tree topology for efficient multicast, and a mesh topology for distributing membership information and application content when the tree topology is partitioned. This solution has several limitations. First, the tree configuration is fragile and the discovery of tree partitions may be slow. Second, Yoid trees can be lop-sided, with longer-than-necessary diameters, thus causing high message latency. Finally, membership information is flooded to all the nodes in the system, and hence Yoid is only appropriate for small multicast groups.

Snoeren et al. [60] construct an infrastructure of servers, and each node is connected to k servers from which it receives duplicate packet streams. While this approach achieves high reliability, it also incurs substantial overhead: each packet is sent to each node by k different servers. In contrast, in the absence of packet loss, each Araneola node receives each packet from exactly one node. In addition, Snoeren et al.'s solution is based on server infrastructure, which is not required by Araneola.

ODRI [48] is a dynamic overlay based on de Bruijn graphs that preserves the properties of these graphs namely an average constant in and out degree at each node, a diameter that grows logarithmically with the number of nodes, and good resilience to node and link failures. Whereas in a k-Araneola overlay each node is connected to either k or $k+1$ nodes, in ODRI with the same k parameter, each node has k incoming and between k and $O(k \ln N)$ outgoing links. Hence, Araneola achieves better load balancing than ODRI. In addition, the join overhead in ODRI is logarithmic in the number of nodes, whereas in Araneola the join cost is constant.

PRM (Probabilistic Resilient Multicast) [6] is a multicast data recovery scheme based on randomized data forwarding. In PRM, nodes self organize into an overlay network, and multicast messages are disseminated over a multicast tree. In order to overcome failures, PRM also includes a proactive component called randomized forwarding in which each node chooses a constant number of other nodes uniformly at random and forwards data to each of them with low probability, e.g., 0.01–0.03. In contrast, Araneola constructs and maintains a richly-connected overlay, in which there are k-disjoint paths between every pair of nodes, and floods aggregated message identifiers over all the overlay links. Compared to PRM, Araneola employs substantial higher redundancy, and hence achieves full reliability under substantially higher message loss and churn rates. For example, the PRM system incorporated into the NICE protocol [5] achieves reliability of roughly 97% in settings with message loss up to 5% and with up to 5 topology changes per second [6], whereas Araneola achieves perfect reliability in such a setting. In addition, the randomized forwarding mechanism creates message duplications, whereas Araneola does not create any message duplications. Additionally, all the parameters of the Araneola protocol are independent of the number of nodes, whereas the parameters of the PRM protocol (e.g., the number of nodes with which a node gossips and the probability with which the node forwards data to each of these nodes) does depend on the number of nodes in the system. Moreover, Araneola was tested in real LAN and WAN settings, whereas the PRM system was tested only in simulations.

Finally, we are unaware of a previous peer-to-peer multicast system that provides full reliability of message delivery in highly dynamic failure-prone environments. In addition, none of the aforementioned multicast systems was evaluated under the high churn and failure rates that we evaluate Araneola under.

2.2 Overlay Structures

Lin et al. [47] construct a static k-Harary graph [31]; such a graph has a logarithmic diameter, a degree of k, and a connectivity level of k, and is therefore an attractive structure for supporting reliable multicast. Lin et al. study the tradeoffs between a gossip protocol and flooding messages on a static overlay structured like a 4-Harary graph in small fixed networks. Their measurements show that at moderate failure rates, flooding a small overlay achieves the same reliability with a substantially smaller overhead than a gossip protocol. As the failure rate increases, however, the overlay can become partitioned, and the gossip protocol exhibits a much more graceful degradation. This motivates a solution like Araneola, based on a dynamic overlay that detects failures and continuously heals itself.

Recently, several dynamic peer-to-peer overlays with logarithmic diameters and bounded node degrees have been suggested, e.g., emulating the Butterfly [49], de Bruijn graphs [39], Small Worlds graphs [44], or random expander graphs with degrees≥ 8 [46]. However, none of these systems can guarantee, with high probability, a lower cost than $O(\log N)$ messages and time for handling joins, since a joining node must search and locate its (random or hashed) joining location prior to joining the system. Chawathe et al. [17] have argued that this logarithmic cost inhibits the scalability of such systems assuming the churn rates measured in Gnutella and Napster [58]. Moreover, the algorithm in [49] is complicated, and the overlay in [46] does not support as many concurrent leave operations as Araneola does.

Several overlay structures, e.g., [50, 59], reduce message delivery latency and communication costs by incorporating links between nearby nodes in addition to the random links required for achieving a good overlay. Other overlays, e.g., Pastry [57] and Tapestry [66], achieve local routing by selecting nearby nodes among a large collection of random ones. Land's [2] lookup algorithm achieves a worst case stretch bound of $1 + \epsilon$ by adding local links that increase node degrees by a constant expected factor.

Although adding links to proximate nodes has many benefits, we believe that proximity requirements vary among applications. We therefore advocate a separation of concerns between such specific application needs and generic requirements of wide-area applications. The basic overlay of Araneola addresses the generic needs while incurring a low load, and thus leaves ample bandwidth for the application to address additional needs such as proximity, bandwidth heterogeneity, and so forth. We illustrate this approach in Section 6 by extending Araneola's basic overlay with links chosen according to network proximity in order to reduce the latency of message delivery and communication costs. The resulting extended overlay achieves a smaller average degree than [50, 59] and better load balancing than [59] in terms of the average number of times a node is required to forward a message.

2.3 Normal Random Graphs

A random graph is a graph obtained by starting with a set of N vertices and adding edges between them at random. There are two common models for normal random graphs. In the Erdos−Renyi model, called $G(N, p)$, each of the $\binom{n}{2}$ possible edges occurs independently with probability p. A closely related model, $G(N, M)$ assigns equal probability to all graphs with exactly M edges. As opposed to a k-regular random graph, a normal random graph can be easily constructed in a distributed fashion. However, k-regular random graphs have substantial better connectivity, and therefore robustness, characteristics. Specifically, for $k \geq 3$, a k-regular random graph is generally k-connected. In addition, such a graph remains highly connected following random removal of linear-size subsets of edges or nodes. In contrast, in order for a normal random graph to be even connected (with high probability), its average degree must be at least logarithmic

in N [11]. Finally, the diameter of both k-regular and normal random graphs grows logarithmically with N.

An Araneola overlay combines the advantages of both normal and k-regular random graphs. On the one hand, as we show throughout Section 5, Araneola's basic overlay achieves the important mathematical properties of k-regular random graphs, namely logarithmic diameter, k-connectivity, and high robustness. On the other hand, similarly to the construction of normal random graphs, the protocol for constructing an Araneola overlay is simple and fully distributive. In addition, the construction overhead is small, and each join, leave, or failure event is handled locally and entails the sending of only about $3k$ messages.

2.4 Centralized Constructions of k-Regular Random Graphs

Araneola builds an overlay structure that approximates a k-regular random graph using a distributed protocol in dynamic environments. Previous algorithms for generating k-regular random graphs were centralized and static. For example, Bollobas [11] and Bender and Canfield [7] give a centralized construction of a k-regular random graph on N vertices, which works roughly as follows: it duplicates each vertex k times and creates a uniform random perfect matching[3] on these Nk copies of vertices. The resulting graph contains an edge between two vertices, i and j, if the matching contains an edge between copies of i and j. The resulting graph may not be simple, i.e., it may contain self-loops and/or parallel edges. It has been shown [65] that the probability of such a graph being simple is $\exp(-k^2/4)$, and the expected time to obtain a simple k-regular random graph with this algorithm is $O(Nke^{k^2/4})$. McKay and Wormald [51] improve this expected time to $O(N^2k^4)$ using a simple algorithm, and to $O(Nk^3)$ using a complicated and hard to implement algorithm.

Steger and Wormald [61] propose a faster algorithm based on Bollobas's [11] and Bender and Canfield's [7] constructions. This algorithm creates a perfect matching that does not contain self-loops and parallel edges, and hence the resulting graph is always simple. The running time of this algorithm is $O(Nk^2)$. Steger and Wormald prove that if $k = o(N^{1/28})$ then the distribution of the generated graphs is asymptotically uniform[4]. Recently, Kim and Vu [43] have proven that the distribution of graphs generated using this algorithm is asymptotically uniform with k up to $N^{1/3}$.

Araneola is the first distributed and efficient approximation of a k-regular random graph that we are aware of. As opposed to the centralized constructions mentioned above, in which each addition or removal of a single vertex or edge from the graph requires the reconstruction of the graph from scratch, Araneola incrementally incorporates joining nodes and removes leaving ones from the graph, while sending only about $3k$ messages for each such change. In Section 5.5, we show that the overlays generated by Araneola have the same diameter and are as robust as the graphs generated using the centralized construction of [61, 43].

3 Design Goals

The purpose of Araneola is to support scalable reliable multi-point to multi-point communication in dynamic wide-area settings where nodes frequently join and leave (or fail). We have set the following requirements for our service:

- High reliability – high reliability in face of high failure and message loss rates, and graceful degradation in the face of increasing failure rates. The reliability should be independent of the number of nodes.

[3] A matching on a graph G is a set of vertex-disjoint edges of G. A perfect matching is a matching that covers all vertices.

[4] The distribution of the generated graphs approaches a uniform distribution as $N \to \infty$.

- Low latency, increasing at most like $O(\log N)$; the latency should remain low while multiple nodes are joining and leaving (or failing).
- Low constant load on each node.
- Low constant cost for handling joins and failures.
- Quick failure recovery and prompt incorporation of joining nodes.

In addition, Araneola is designed to be suitable for a variety of wide-area applications. We believe that each application has its own considerations for link selection based on application-defined proximity metrics and available bandwidth. Therefore, we construct a generic low-degree overlay that incurs low load on each node, leaving ample bandwidth for each node for communication with additional nodes chosen in an application-specific manner.

Araneola is designed to achieve all the above goals without using any infrastructure, servers, or any elaborate communication mechanism beyond point-to-point UDP communication between pairs of nodes — we assume that every pair of nodes can communicate with each other.

In order to achieve these goals, Araneola strives to build a basic overlay structure with the following characteristics: (i) multiple disjoint paths between every pair of nodes, where the number of paths is a configurable parameter k; (ii) robustness to random removal of a certain percentage of the nodes or edges; (iii) low diameter and average distance, increasing at most like $O(\log N)$; (iv) low bounded degree (3 or more), which leaves plenty of bandwidth for communication with additional nodes according to application needs; and (v) support for local addition and removal of nodes at a constant cost.

As we have seen, k-regular random graphs naturally achieve these goals for $k \geq 3$. Therefore, Araneola strives to construct and maintain an overlay that approximates a k-regular random graph. As noted in Section 2.4, creating a perfect k-regular random graph could be difficult and costly. Instead, Araneola is designed to converge to a random graph in which each node has a degree of either k or $k+1$ and no two neighboring nodes have a degree of $k+1$. We show that the desirable graph properties of k-regular random graphs carry over to graphs with this structure.

4 Araneola's Overlay

Araneola's protocol has three components: one implements a randomized partial membership service (see Section 4.1), the second constructs and maintains the basic overlay (see Section 4.2), and the third implements the multicast service (see Section 7). All Araneola nodes run these three components. Araneola handles each multicast group independently, i.e., it builds an overlay structure for each multicast group. Since each group is handled independently, we present the protocol for a single group, and omit the group's name.

4.1 The Membership Service

When joining the overlay, a node randomly selects several other nodes to connect to. This requires each node to know some other nodes' identities. To this end, we implement a scalable randomized membership protocol similar to [59], where membership information is gossiped over the overlay's links. Each node maintains a small set of node identities, called a *membership view*, which evolves over time. The size of the

membership view S is a predefined parameter. Each node has a log file that contains random node identities received in a previous session. When a new node joins Araneola for the first time it can ask another node for its membership view, and use that as its initial view.

Periodically, each node's membership protocol piggybacks a small amount of membership information on messages sent to the node's neighbors. Specifically, the node sends a certain number of random node identities from its membership view to each neighbor. Upon receiving such membership information from a neighbor, the node adds these node identities to its membership view. Then, if the membership view includes more than S node identities, then random node identities are removed from the membership view until it includes only S node identities.

Whereas in a gossip-based multicast protocol, e.g., Lpbcast [22], each node uses its membership information in every round in order to disseminate multicast data, in Araneola, membership information is used infrequently, only for overlay maintenance. Specifically, a node consults its membership service only when its degree drops below a predefined threshold. Note that, similarly to gossip-based protocols, we could have implemented a gossip-based multicast layer directly on top of the membership service. However, as we explain in Section 2.1.1 and experimentally show in Section 8.1.3, gossiping over Araneola's overlay eliminates the shortcomings of gossip-based protocols, and further improves the scalability of these protocols. Since membership information is used infrequently in Araneola, it can also be disseminated infrequently. Empirically, even under churn rates exceeding those measured over the Internet [58] and the Mbone [3], disseminating membership information once a minute suffices for creating a robust overlay and achieving full reliability of message delivery (see Section 8.2). In Section 4.3.1, we calculate the overhead incurred by the membership service. In a typical setting, the per-node membership overhead is 300 bytes per-minute, regardless of the churn rate.

In Section 5.4, we evaluate the effect of the membership service on the overlay. We show that the initial distribution of the membership views has a small effect on the quality of the constructed overlay: a k-Araneola overlay is at least $k-1$-connected even when the initial distribution of the membership views is skewed.

4.2 Building and Maintaining the Overlay

The protocol for constructing and maintaining the overlay is composed of three tasks: (i) the *connect task* (see Section 4.2.1) adds new connections when a node's degree is below a configurable parameter called L, which determines the graph's target degree (k); (ii) the *disconnect task* (see Section 4.2.2) tries to reduce a node's degree if it is above L, without causing any node's degree to drop below L; and (iii) the *failure detector* task detects neighboring node failures. This task simply generates an fd_suspect event when messages from a given neighbor fail to arrive for a certain period of time. The failure detector is straightforward and we do not describe it in pseudo-code.

Araneola's data structures are presented in Figure 1. The set *neighbors* holds the node's current neighbors in the overlay, with their respective degrees. The degree of a node is the size of its neighbors set, i.e., $|neighbors|$. The set *next_round_connect* contains node identifiers received from redirections of CONNECT requests as explained below. The current time can be read from *clock*. The set *connect_to_node* and the boolean flag *rule2_flag* are used by the reduction task (see Figure 3), and are explained below. The parameter L determines the graph's target degree (k), and the parameter H defines the maximum allowed degree for a node. These are configurable parameters: L affects the connectivity and diameter of the overlay, while H affects the overhead of constructing the overlay. A number of timeout values are defined in order to control

the frequency at which different events occur.

Data structures:
id – this node's identifier.
neighbors – set of pairs ⟨id,degree⟩, initially ∅.
next_round_connect – set of pairs ⟨id,degree⟩, initially ∅.
clock – the current time.
connect_to_node – set of node identifiers, initially ∅.
rule2_flag – a binary flag, initially 0.
Parameters:
L – target number of neighbors.
H – upper bound on the number of neighbors.
Timeouts: connect_timeout, disconnect_timeout.

Figure 1: Araneola's data structures and parameters.

4.2.1 The connect task

When a node's degree is below L, the connect task (see Figure 2) periodically attempts to set up as many new connections as it is missing to randomly chosen nodes (lines 1–10). The target nodes are chosen either from the set *next_round_connect* or at random from the local membership view. For each attempted connection, the node sends a CONNECT request (line 9). At bootstrap time, the node issues CONNECT requests to L nodes, and then sleeps for *connect_timeout*. It is expected that during this period enough new connections will be formed, although since some of the chosen nodes may be faulty or overloaded, there may be a need to attempt more connections after the timer expires. The connect task can be awoken by other tasks before the timer expires (line 35).

A node that receives a CONNECT request (line 11) *accepts* it, by calling add_connection, provided that the sum of the sizes of the sets *neighbors* and *connect_to_node* is smaller than H, and otherwise it *redirects* the request, as will be explained shortly. The procedure add_connection adds the sender to neighbors (line 31) and responds with a CONNECT_OK. Upon receiving the CONNECT_OK (line 18), the requester registers the new connection if either its degree is still smaller than H, or the sender of the CONNECT_OK message is in *connect_to_node*. Otherwise, the requester sends a LEAVE message to the sender (line 25). A LEAVE message causes its receiver to remove its connection with the sender (lines 26–27), and wake up the connect task if necessary (lines 34–35).

Redirecting is done by sending a REDIRECT message to the requester, naming the sender's lowest degree neighbor l (line 15). This causes the requester to add l to its *next_round_connect* set (line 17). The next time the requester's connect_task will awaken, it will attempt to connect to l rather than to a random node (line 5). CONNECT and CONNECT_OK messages carry the sender's current degree for initializing the *degree* in the *neighbors* data structure. In addition, every node periodically sends its degree to its neighbors, in order to keep the neighbors data structure up-to-date (this is not shown in the code). This information (which is couple of bytes long) is piggybacked on gossip messages. A node that voluntarily leaves the system sends a LEAVE message to all its neighbors. An involuntary failure of a neighbor is detected using the failure detector, which generates an fd_suspect event (line 28). When a node detects a neighbor as faulty, it sends that neighbor a LEAVE message and removes the connection by calling remove_connection (line

30).

4.2.2 The disconnect task

With the connect task a node's degree can be as high as H. The disconnect task (see Figure 3), which is composed of two rules (*Rule 1* and *Rule 2*), reduces node degrees, so that, eventually, each node's degree is either L or L+1, and at most 50% of the nodes have degree L+1.

Rule 1. *Rule 1* removes the connection between a pair of nodes that both have degrees higher than L (Figure 3, lines 5–9). Specifically, if a node n's degree is $L+i$, then n attempts to remove i of its neighbors. Neighbors with degrees higher than L are candidates for removal; they are inserted into the set *cands* (line 5). If *cands* contains more than i nodes, the i lowest identifier ones are kept (line 6). If n has a higher id than a node c in *cands*, then n sends a DISCONNECT message to c (line 9). Upon receiving this message (line 17), if c's degree is still higher than L and n is in c's *cands* set, it removes the connection with n, and sends a DISCONNECT_OK message. By checking that n is in c's *cands* set, we ensure that parallel invocations of Rule 1 will not drop c's degree below L. Upon receipt of a DISCONNECT_OK (line 23), n removes the connection with c.

Rule 1 ensures that if from some point onward no nodes join the overlay, then eventually there are no two neighboring nodes that both have degrees higher that L. Every *disconnect_timeout*, each node's *cands* set is set with the i lowest identifier neighbors of the node among the neighbors with degrees higher than L. This ensures that as long as there are two neighboring nodes with degrees>L, each *disconnect_timeout* there are at least two neighboring nodes, a and b, so that $a \in b.cands$ and $b \in a.cands$ (e.g., when a is the lowest-identifier node with degree $> L$, and b is its lowest-identifier neighbor with degree $> L$). Thus, until there are no two neighboring nodes with degrees>L in the overlay, every *disconnect_timeout* at least one link between such two neighboring nodes is removed from the overlay graph, although usually almost all such links will be removed simultaneously. In any case, eventually all such links are removed.

Rule 2. With Rule 1 it is possible for a node to have degree H while all of its neighbors have degree L. This case is solved by *Rule 2*, which is invoked only at a node n when all of n's neighbors' degrees are $\leq$L. With Rule 2, node n chooses its two neighbors with the highest and lowest degrees, h and l, respectively (lines 12–13). If n's degree is at least $l.degree + 2$ and it is not involved in another invocation of Rule 2 (*rule2_flag* = *false*), then n tries to cause h to shift one of its connections from n to l. But before removing h's connection with n, we ensure that l is willing to accept h's connection. Therefore, n contacts l (rather than h) and asks it to try to connect to h, and to ask h to remove its connection with n. To this end, n sends a $\langle$CONNECT_TO,$h\rangle$ message to l. If upon receiving this message l's degree is still $\leq$L, and l's *rule2_flag* is false (line 26), then l inserts h to *connect_to_node*, and sends it a CHANGE_CONNECTION message. The recipient, h, connects to l by calling add_connection (line 33), provided that its *rule2_flag* is *false*, and sends a DISCONNECT message to n if its degree is higher than L (lines 34–35). Note that h's CONNECT request will be approved by l, since prior to sending the CHANGE_CONNECTION message to h l inserts h to its *connect_to_node* set. This connection with h can increase l's degree, but not to become higher than L+1 since l accepts a CONNECT_TO request only if its degree$\leq$L and its *rule2_flag* is *false*. Moreover, note that if l's degree will become higher than L, and n's degree will remain above L, then Rule 1 will eventually reduce l's degree back to L. Finally, each node's *rule2_flag* and l's *connect_to_node* set are set to *false* and $\emptyset$, respectively, after each of them is no longer involved in the current invocation of Rule 2 (Figure 2 line 22

Connect task:
1. **loop forever**
2. $gap \leftarrow \mathrm{L} - |neighbors|$
3. **for** $(i = 0; i < gap; i++)$
4. **if** $|next\_round\_connect| \neq \emptyset$ **then**
5. $n \leftarrow$ element in $next\_round\_connect$
6. remove n from $next\_round\_connect$
7. **else**
8. $n \leftarrow$ random node from membership service
9. send $\langle$CONNECT, $|neighbors|\rangle$ to n
10. sleep (connect_timeout)

Event handlers:
11. **upon** receive $\langle$CONNECT, $d\rangle$ from n **do**
12. **if** $(|neighbors| + |connect\_to\_node| < \mathrm{H})$ **then**
13. add_connection (n, d)
14. **else**
15. send $\langle$REDIRECT, lowest degree neighbor$\rangle$ to n

16. **upon** receive $\langle$REDIRECT, $n'\rangle$ from n **do**
17. $next\_round\_connect \leftarrow next\_round\_connect \bigcup \{n'\}$

18. **upon** receive $\langle$CONNECT_OK, $d\rangle$ from n **do**
19. **if** $(|neighbors| + |connect\_to\_node| < \mathrm{H} \vee n \in connect\_to\_node)$ **then**
20. $neighbors \leftarrow neighbors \bigcup \{n, d\}$
21. **if** $(n \in connect\_to\_node)$ **then**
22. $rule2\_flag \leftarrow false$
23. remove n from $connect\_to\_node$
24. **else**
25. send $\langle$LEAVE$\rangle$ to n

26. **upon** receive $\langle$LEAVE$\rangle$ from n **do**
27. remove_connection(n)

28. **upon** fd_suspect (node_id n) **do**
29. send $\langle$LEAVE$\rangle$ to n
30. remove_connection (n)

Procedures:
Procedure add_connection (node_id n, int d)
31. $neighbors \leftarrow neighbors \bigcup \{< n, d >\}$
32. send $\langle$CONNECT_OK, $|neighbors|\rangle$ to n

Procedure remove_connection (node n)
33. remove n from *neighbors*
34. **if** $(|neighbors| < \mathrm{L})$ **then**
35. wake up connect task

Figure 2: Overlay construction: the connect task.

and Figure 3 lines 22 and 36). rule2_flag ensures that at any moment each node is involved in at most one invocation of Rule 2, and hence no deadlock situations are possible.

Proposition 1. *If there is a time after which no nodes join, leave, fail, or are detected as faulty, then each node's degree is eventually either L or L+1, and at most* 50% *of the nodes have degree* $L+1$.

Proof. Rule 1 removes connections between every pair of neighbors with degrees higher than L, without adding new connections. Thus, Rule 1 ensures that eventually, no more than 50% of the nodes have degrees higher than L. Since nodes with degrees lower than H accept new connections, all joining nodes eventually succeed in forming connections with at least L other neighbors. Therefore, the connect task and Rule 1 ensure that eventually, each node's degree is between L and H, and no two neighboring nodes have degree>L. This implies that at least 50% of the nodes have a degree of L.

With Rule 1, it is still possible for a node to have a degree>L+1 when all of n's neighbors have a degree of L. In this case, Rule 2 is invoked at node n, reducing n's degree by one and increasing the degree of n's lowest degree neighbor l by one (in this case l is a random neighbor of n) without changing the rest of n's neighbors' degrees. Now, l's degree equals to L+1 and Rule 1 becomes enabled again, disconnecting the connection between n and l. Thus, after activating Rule 2 and Rule 1 consecutively, n's degree is reduced by 2 while the degrees of the rest of n's neighbors remain L. If n's degree still above L+1, further consecutive activations of the two reduction rules reduce n's degree each time by 2 until its degree becomes either L or L+1. □

Although the worst-case convergence time can be linear in N, in practice, in all of our experiments with up to 10,000 nodes, the overlay converged to a state in which each node's degree is either L or L+1 within less than 10 disconnect timeouts.

4.2.3 The probability for an overlay partition.

The probability that a k-regular random graph is not k-connected is $O(N^{2-k})$ [65]. Empirically, as we show in Section 5.5, a k-Araneola overlay achieves a slightly better fault-tolerance to node and link failures than a k-regular random graph. This is since in a k-Araneola overlay the degree of each node is between k and $k+1$ whereas in a k-regular random graph all the nodes have a degree of k. In addition, as we show in Section 5.4, the initial distribution of the membership views has a small effect on the overlay's fault-tolerance. In hundreds of runs, a k-Araneola overlay was always $k-1$ or k connected even when the initial distribution of the membership views was skewed. Moreover, as we show in Section 5.2, a 5-Araneola overlay is connected even after a random removal of up to 10% of its edges or after a random removal of up to 15% of its nodes. Therefore, for $k\geq 5$ and $N\geq 1000$, the probability that a k-Araneola overlay becomes partitioned is negligible.

4.3 Maintenance Overhead

In Section 4.3.1, we calculate the overhead incurred in a steady state, i.e., in the absence of churn. In Sections 4.3.2 and 4.3.3, we calculate the overhead for the simple case where a single join or leave, respectively, occurs when the system is stable, i.e., each node's degree is either L or L+1, and no two neighboring nodes have a degree of L+1. In Section 5.3.2, we show that this analysis gives a good estimation for dynamic settings in which the churn rate is low. When the churn rate rises, the overhead decreases because when

Disconnect task:

1. **loop forever**
2. sleep (disconnect_timeout)
3. $i \leftarrow |neighbors| -$ L
4. **if** ($i > 0$) **then**
/ Rule 1 */*
5. $cands \leftarrow \{n \in neighbors : n.degree > L\}$
6. **if** ($|cands| > i$) **then** $cands \leftarrow i$ elements of $cands$ with lowest identifiers
7. **foreach** $c \in cands$
8. **if** ($c.id < id$) **then**
9. send $\langle$DISCONNECT$\rangle$ to c
/ Rule 2 */*
10. **if** ($cands = \emptyset \wedge !rule2\_flag$) **then**
11. $rule2\_flag \leftarrow true$
12. $h \leftarrow$ random neighbor among the neighbors with the highest degree
13. $l \leftarrow$ random neighbor among the neighbors with the lowest degree
14. **if** ($|neighbors| \geq l.degree + 2$) **then**
15. $cands \leftarrow cands \bigcup \{h\}$
16. send $\langle$CONNECT_TO,$h\rangle$ to l

Event handlers:

17. **upon** receive $\langle$DISCONNECT$\rangle$ from n **do**
18. **if** ($|neighbors| >$ L $\wedge\ n \in cands$) **then**
19. remove_connection(n)
20. send $\langle$DISCONNECT_OK$\rangle$ to n
21. **if** ($n \in cands$) **then**
22. $rule2\_flag \leftarrow false$

23. **upon** receive $\langle$DISCONNECT_OK$\rangle$ from n **do**
24. remove_connection(n)

25. **upon** receive $\langle$CONNECT_TO, $n'\rangle$ from n **do**
26. **if** ($|neighbors| \leq$ L $\wedge\ !rule2\_flag$) **then**
27. $rule2\_flag \leftarrow true$
28. $connect\_to\_node \leftarrow \{n'\}$
29. send $\langle$CHANGE_CONNECTION,$|neighbors|, n\rangle$ to n'

30. **upon** receive $\langle$CHANGE_CONNECTION,$d, n'\rangle$ from n **do**
31. **if** ($|neighbors| <$ H $\wedge\ !rule2\_flag$) **then**
32. $rule2\_flag \leftarrow true$
33. add_connection (n)
34. **if** ($|neighbors| >$ L) **then**
35. send $\langle$DISCONNECT$\rangle$ to n'
36. $rule2\_flag \leftarrow false$

Figure 3: Overlay construction: reducing node degrees.

many join and leave events occur concurrently their costs can be amortized. For example, a join event may increase a node's degree while a leave event is reducing it, eliminating the need for correcting the overlay.

In Sections 4.3.2 and 4.3.3, we denote by p the probability that a node has a degree of L, and the probability that a node has a degree of L+1 is $1 - p = q$.

4.3.1 Steady state and membership overhead

In a steady state, no control messages[5] are sent. In this case, the overhead is composed out of the membership overhead only. Recall that, every predefined period, each node's membership protocol piggybacks a small number of random node identities on messages sent to the node's neighbors. Specifically, in all of our experiments, every minute, the membership protocol sends 10 random node identities to each of the node's neighbors. We represent each node identity as a 6-cell byte array. Assuming each node has 5 neighbors, the per-node membership overhead is $5{\cdot}10{\cdot}6{=}300$ bytes per-minute. As explained in Section 4.1, the per-node membership overhead is fixed, and does not depend on the churn rate.

4.3.2 The overhead for join

We begin by calculating the expected overhead for a single CONNECT request. Assume that node c issues a CONNECT request to node t. We distinguish between three possible cases: (i) t and all of its neighbors have a degree of L; (ii) t has a degree of L and at least one of its neighbors has a degree of L+1; or (iii) t has a degree of L+1. In the latter case, all of t's neighbors have a degree of L. The probability for case (i) is p^{L+1}, the probability for case (ii) is $p(1 - p^L)$, and the probability for case (iii) is $1 - p = q$.

In case (i), c sends one CONNECT message to t and in return, t sends one CONNECT_OK message to c, for a total of two messages. In case (ii), in addition to the CONNECT and CONNECT_OK messages, two additional messages (DISCONNECT and DISCONNECT_OK) are later sent (by Rule 1) in order to reduce the degree of t and one of its neighbors from L+1 to L. Thus, a total of four control messages are sent. In case (iii), after sending the CONNECT and CONNECT_OK messages, t's degree becomes L+2 while the rest of t's neighbors still have degrees of L. In this case, t activates Rule 2. First, t sends to its lowest degree neighbor, l, a CONNECT_TO message with the identity of its highest degree neighbor, h. Then, l sends a CHANGE_CONNECTION message to h with t's identity. In return, h sends a CONNECT message to l and a DISCONNECT message to t. Finally, l sends a CONNECT_OK message to h and t sends a DISCONNECT_OK message to h. Now, the degrees of t and l become L+1 and the degree of h remains L. In the next iteration of the reduce algorithm Rule 1 is applied and either t or l sends a DISCONNECT message to the other and the other replies with a DISCONNECT_OK. The total number of messages sent in case (iii) is thus ten.

The expected number of control messages sent for a single CONNECT request is therefore:

$$2p^{L+1} + 4p(1 - p^L) + 10q = 4p + 10q - 2p^{L+1}.$$

Since a joining node sends L CONNECT messages, the expected overhead associated with a single join operation during a stable period is:

$$L(4p + 10q - 2p^{L+1}).$$

[5]Control messages are messages that are sent by the Connect and Disconnect tasks (see Figure 2 and Figure 3).

The above analysis of the join overhead ignores the possibility for cascading reconnections. In Section 5.3.2, we compare this analyzed join/leave overhead with the measured join/leave overhead, and find that they are very close. That is, cascading reconnections do not have a significant impact on the join overhead.

4.3.3 The overhead for leave

Assume that node l sends a LEAVE message to node t. There are two possible cases: either (i) t has a degree of L; or (ii) t has a degree of L+1. The probability for case (i) is p, and the probability for case (ii) is q. In the first case, l sends a LEAVE message to t. Subsequently, t sends a CONNECT request to a random new node. We showed above that the expected overhead associated with a CONNECT request is $4p + 10q - 2p^{L+1}$. Thus, the expected number of messages sent in the first case is $1 + 4p + 10q - 2p^{L+1}$. In the second case, l sends a LEAVE message to t. However, in this case, t does not send any messages as its degree becomes L. Thus, the total expected overhead for sending a LEAVE message is:

$$p(1 + 4p + 10q - 2p^{L+1}) + q.$$

The expected number of LEAVE messages a node sends upon leaving the system is:

$$pL + q(L + 1) = L + q.$$

Thus, the expected number of messages sent upon a node leaving the system is:

$$(L + q) * [p(1 + 4p + 10q - 2p^{L+1}) + q].$$

5 Evaluation of Araneola's Overlay

We have implemented the code for constructing and maintaining Araneola's overlay in Java using UDP/IP. In our experiments, we set the connect_timeout to 5 seconds and the disconnect_timeout and the connect_to_timeout to 30 seconds. Membership information is gossiped once a minute. At bootstrap, each node's membership view contains ten node identities chosen uniformly at random. In this section, we evaluate Araneola's overlay on a single LAN in Netbed [64]. In Section 6, where we extend Araneola to exploit network proximity, we evaluate Araneola's overlay also on a WAN. We begin our study, in Section 5.1, by evaluating Araneola's overlay in a static setting; we study the impact of L and H on the overlay as well as the overlay's scalability. In Section 5.2 we study the overlay's fault-tolerance. In Section 5.3, we measure the join and leave overhead in experiments with high churn. In Section 5.4, we evaluate the effect of the membership service on the overlay. Finally, in Section 5.5, we compare Araneola's overlay with k-regular random graphs constructed using a centralized algorithm.

5.1 Static Evaluation

In our static evaluation, all the nodes are created simultaneously, and remain up throughout the experiment. Each experiment lasts 5 minutes. Empirically, we saw that within this time the overlay converges to a stable state, in which each node's degree is either L or L+1 and no two neighboring nodes have a degree of L+1.

Each experiment (with a given number of nodes and choice of parameter settings) was run at least 4 times, for a total of several dozens.

5.1.1 The impact of L

Araneola's parameter L affects the load imposed on each node. In Section 4.3 above we have shown that the join/leave overhead grows roughly linearly with L. Additionally, increasing L increases the multicast overhead, since data or gossip messages are sent on all links.

Nevertheless, increasing L yields a number of benefits. First, it improves the overlay's connectivity and robustness. In Section 5.2, we show that when L$=5$, the overlay generally remains connected after random removal of 15% of its edges or nodes, while when L$=4$, it remains connected after the removal of only about 10% of the edges or 7% of the nodes. Second, increasing L reduces the overlay's diameter. Note that the connectivity and robustness of a k-regular random graph with a given k is independent of the number of nodes. Therefore, we can set the value of L regardless of the number of nodes in the system. The value of L, however, has a small effect on the overlay's diameter. Below, we examine the relationship of Araneola's overlay's diameter with that expected in a k-regular random graph for different group sizes.

Wormald [65] gives the following formula for the expected diameter of a k-regular random graph: the diameter D *asymptotically almost surely (aas)*[6] satisfies:

$$1 + \lfloor \log_{k-1} N \rfloor + \lfloor \log_{k-1}(\frac{(k-2)}{6k} \log N) \rfloor \leq D \leq 1 + \lceil \log_{k-1}((2+\epsilon)kN \log N) \rceil.$$

To understand the impact of L, we experiment with 8000 nodes (on 100 Netbed machines) for values of L ranging from 3 to 10, and measure the diameter of each overlay. In Table 1, we report the highest overlay diameter measured for each value of L, and compare it to the formula above. We see that the highest diameter of Araneola's overlay occurs in the range predicted by the formula.

L	Expected diameter range in L-regular random graphs [65]	Highest measured Araneola diameter
3	13–19	13
4	9–13	9
5	7–11	8
6	6–9	7
7	6–9	6
8	5–8	6
9	5–8	6
10	5–8	6

Table 1: The impact of L on Araneola's diameter versus Wormald's formula, 8000 nodes.

Increasing the value of L has a third benefit— it constructs an overlay that more closely approximates a regular graph, in that a higher percentage of the nodes have a degree of L, as shown in Figure 4.

In most of the experiments we present below, we set L to 5. We chose this value because it provides a good balance between the desired properties: the load imposed on each node is still modest, and the overlay's diameter is small. Moreover, as we shall see below, it yields a robust overlay (twice as resilient to

[6] A property holds aas if the probability that it holds approaches 1 as $N \to \infty$.

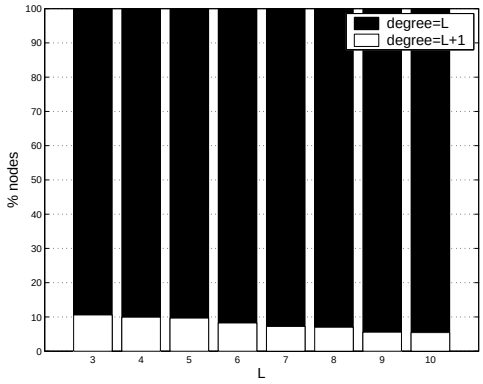


Figure 4: Degree distribution, 8000 nodes.

node failures as with L$= 4$) and achieves 100% reliability at join and leave rates exceeding those measured on the MBone [3].

5.1.2 The impact of H

Next, we examine the impact of the parameter H. Like L, the choice of H does not depend on the system size. This is since the expected number of control messages received by each node does not vary with the number of nodes. We experiment with $N = 1000$, L$= 5$, and different values of H. We observe that in order to obtain a reasonable overhead, H needs to be at least L$+5 = 10$. When H is lower than 10, we get a high overhead— some nodes send hundreds of CONNECT requests before finding a node with degree lower than H. This occurs since nodes run the reduce task only once in 30 seconds, in the interim, many node's degrees can rise above L, especially in our static experiments where all nodes are created simultaneously. When H is set to L$+5 = 10$, however, this problem is eliminated and the average number of control messages received by each node is between 8 and 9, independently of N. The number of control messages received by each node is normally distributed. We did not observe significant differences among values of H ranging from 10 to 20: for all values of H between 10 and 20, the average number of control messages received was between 8.3 and 8.6. Similar results were obtained for L equal to 4 or 6. We therefore henceforth fix the value of H to be L$+5$.

5.1.3 Overlay properties and scalability

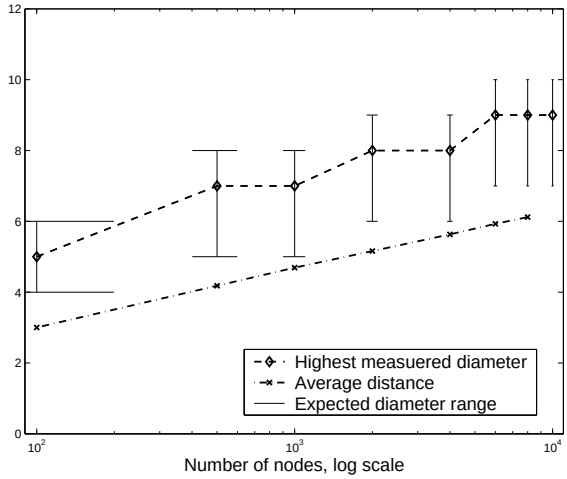
In order to understand Araneola's scalability, we vary N, the group size, from 500 nodes (on 10 Netbed machines) to $10,000$ nodes (on 125 Netbed machines). L and H are set to 5 and 10, respectively. At the end of each experiment, we take a snapshot of the overlay structure, and then analyze its properties offline. We measure node degrees as well as the overlay's diameter, average distance, and connectivity. The results are summarized in Figure 5[7]. The first column in the table shows the percentage of nodes whose degree is L (i.e., 5). The remaining nodes' degrees are L$+1$. For all group sizes, over 90% of the nodes have degree L. The percentage of nodes with degree L does not seem related to N.

The second column presents the (smallest and largest) measured diameters for every value of N. The top curve in the graph depicts the highest measured diameter for each value of N, where the x axis is given

[7]We did not analyze the average distance and connectivity for the experiments with $N = 10,000$.

N	% Nodes degree=5	Measured diameter	Expected diameter	Avg #paths
100	98	5	4–6	5.00
500	91.8	6–7	5–8	5.01
1000	91.4	7	5–8	5.01
2000	92	7–8	6–9	5.01
4000	91.45	8	6–9	5.01
6000	90.42	8–9	7–10	5.01
8000	90.33	9	7–10	5.01
10000	90.36	9	7–10	—

(a)



(b)

Figure 5: Scalability of Araneola's overlay.

in logarithmic scale. Note that this value does not necessarily increase when we increase the group size, and hence there are plateaus in this curve. We observe that Araneola's diameter indeed grows logarithmically with N as Wormald's formula predicts; in all of our experiments, Araneola's diameter occurs (again) in the expected range, which is listed in the next column and depicted using range bars in the graph. When flooding multicast messages over the overlay's links, the diameter gives a measure for the *worst case* latency (in the absence of failures and message loss), whereas the average latency depends on the average distance between two nodes in the overlay. This average is presented in the bottom curve in the graph, and we see that it also increases logarithmically with N.

Finally, we measure the overlay's connectivity. In over 90% of our experiments, the overlay is 5-connected, i.e., there are at least 5 disjoint paths between every pair of nodes. In the few cases where the connectivity was less than 5, there were at most 4 nodes with a connectivity of 4, whereas the rest of the nodes had a connectivity of 5. The average number of node-disjoint paths between every pair of nodes is presented in the last column in the table.

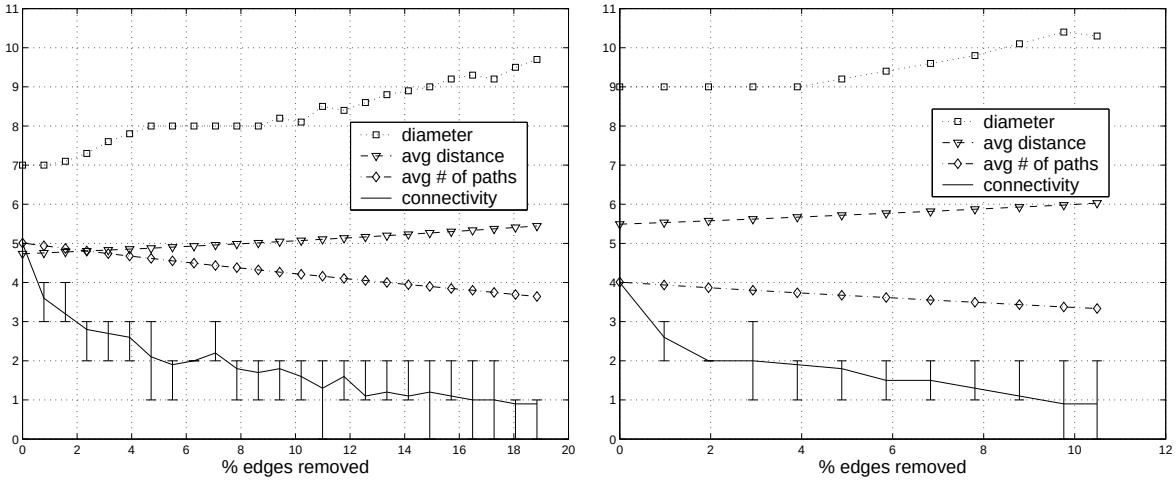
5.2 Fault-Tolerance and Graceful Degradation

We now study the fault-tolerance and robustness of the Araneola overlay. We consider two kinds of failures: communication link failures and node failures. We study the overlay's robustness with an offline analysis of the overlay snapshot obtained at the end of static experiments with 1000 and 2000 nodes. To study communication failures, we remove random subsets of edges from the overlay graph and analyze the resulting graphs. This allows us to predict Araneola's reliability and latency in the presence of message loss. Similarly, we study Araneola's resistance to node failures by removing random subsets of nodes. Note that in the analysis in this section no dynamic repairs are done, i.e., after the initial construction of the overlay no links are added as a result of a node or link failure. Such repairs would have further increased the overlay's fault-tolerance.

As in most previous studies, e.g., [21, 47, 63], we model node and edge failures as *independent and identically distributed (IID)*. For node failures, the IID assumption has no significance since the overlay structure is random. Moreover, Bhagwan et al. have found that host failures are indeed independent [8]. For

edge failures, the IID assumption fails to capture a situation in which some nodes have poorer links than others. Nevertheless, we show in Section 8.3 that even in WAN-like settings where some nodes have only poor links, Araneola exhibits similar performance as when message loss is IID. Designing an overlay that explicitly withstands correlated edge failures can be a consideration for application-specific extensions of Araneola, and it is beyond the scope of this paper.

5.2.1 Communication failures



(a) L= 5. (b) L= 4.

Figure 6: Resilience of Araneola's overlay to edge removals, 1000 nodes.

We first analyze the impact of edge removals on the overlay with L= 5 and $N = 1000$. This overlay has 2547 edges. For each percentage $p \leq 50$ of the edges, we remove 10 different random subsets consisting of $p\%$ of the edges from the overlay graph. The overlay becomes partitioned for the first time in one of the ten experiments removing 11% (280) of the edges, and then in one of the experiments removing 15% (380). In both cases, a single node became disconnected from the rest. Figure 6(a) shows how the removal of up to 19% of the edges affects the overlay's characteristics. For each p in this range, the overlay is partitioned in at most one out of ten experiments in which $p\%$ of the edges are removed. We observe that the average diameter increases from 7 to about 8 when 5–10% of the edges are removed, and to 9 when 15% of the edges are removed. The average distance increases more gradually, suggesting that message loss has a moderate effect on the average latency. The average number of disjoint paths also decreases gradually with the failure rate. The bottom curve illustrates the average connectivity. The bars around each data point show the maximum and minimum connectivity observed in experiments with this p; when the minimum goes does to 0, there was a partition in one of the 10 experiments. We next experiment with L= 4 and $N = 1000$. The overlay is less robust in this case— it partitions in more than 10% of the cases whenever $p > 11\%$. Figure 6(b) shows the overlay's degradation when up to 11% of the edges are removed.

We next examine how many of the nodes are still connected to each other, i.e., what is the size of the largest connected component in the graph. Figure 7 depicts the average size of the largest connected component after random edge removals for L= $4, 5, 6$ with $N = 1000$ and for L= 5 with $N = 2000$. We

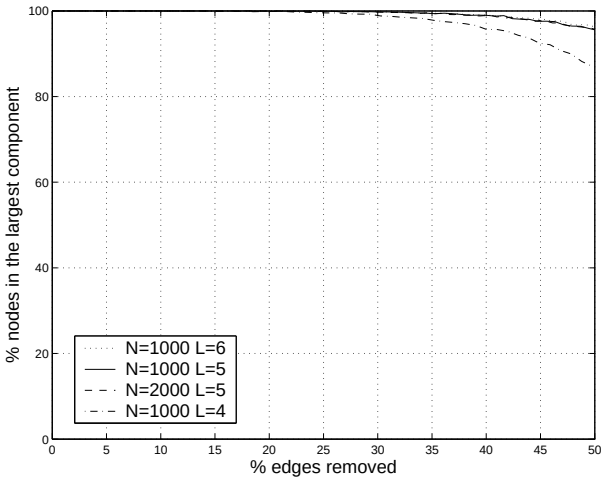
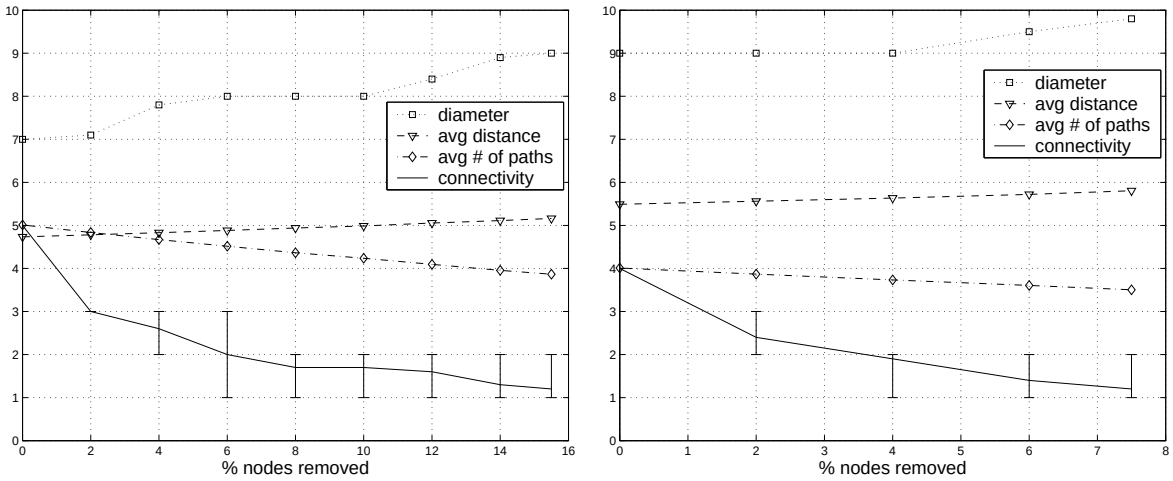


Figure 7: Graceful degradation of Araneola's overlay under edge removals, 1000 nodes.

can clearly see that the overlay's resilience to the removal of a given percentage of its edges is *completely independent of* N, as is expected in k-regular random graphs [27]: the curves for $N = 2000$ and $N = 1000$ (both with L$= 5$) are not distinguishable. As expected, the value of L does impact the overlay's robustness, but the difference between L=5 and L=6 is negligible. Remarkably, for L$= 5$, after the removal of up to 38% of the edges, 99% of the nodes are still connected to each other, and only 1% of the nodes are partitioned from the rest.

5.2.2 Node failures



(a) L$= 5$.

(b) L$= 4$.

Figure 8: Resilience of Araneola's overlay to node removals, 1000 nodes.

We now turn our attention to node failures. Figure 8(a) shows how node removals affect the properties of

an overlay with 1000 nodes and L= 5 when up to 15% of the nodes are removed. None of the experiments with up to 15% removed nodes resulted in partitions. The overlay becomes partitioned in two of the ten experiments in which 16% (160) of the nodes are removed. Experimentally, even if 15% of the nodes running Araneola fail during the brief time interval that it takes to detect and recover from failures (e.g., one minute), Araneola can continue to deliver messages reliably to surviving nodes. As with edge removals, the overlay exhibits graceful degradation: the diameter and average path length increase moderately, while the average number of disjoint paths moderately decreases. When L= 4, the overlay is half as robust to node failures as with L= 5. It becomes partitioned in two of the ten runs with 8% of the nodes removed. Figure 8(b) shows the overlay's degradation when up to 7.5% nodes are removed.

In Figure 9, we examine the size of the largest connected component that survives following node failures, for L= $4, 5, 6$ with $N = 1000$ and for L= 5 with $N = 2000$. Again, the overlay's resilience shows exactly the same trend with $N = 1000$ as it does with $N = 2000$. This suggests that Araneola's resilience to simultaneous failures of a certain percentage of its nodes is also independent of N. When L= 5, the largest component still includes 99% of the nodes following the failure of up to 38% of the nodes. When L= 4, 99% of the nodes are still connected following the failure of 28% of the nodes. When 50% of the nodes fail, the largest component with L= 5 still includes over 95% of the nodes, and with L= 4, it includes 87%. As with edge removals, increasing L from 5 to 6 achieves only slightly better robustness to node removals when there is an unrealistically high failure percentage.

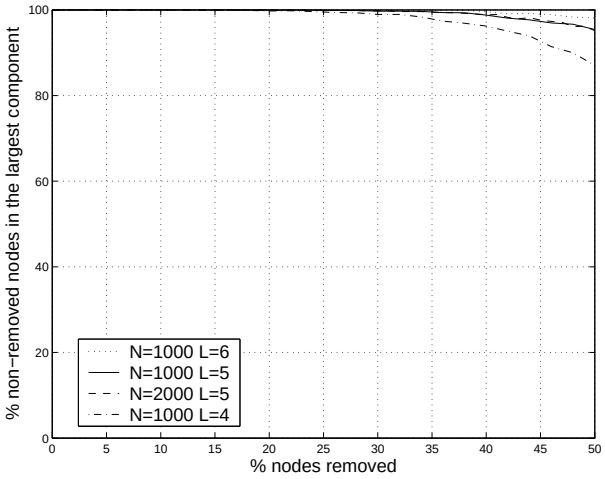


Figure 9: Graceful degradation of Araneola's overlay under node removals, 1000 nodes.

5.2.3 Setting L

In Sections 5.2.1 and 5.2.2, we showed for L equal to 4/5/6 that Araneola's overlay remains connected if the failure-rate does not exceed a certain threshold. Assuming an upper bound on the failure rate, one can choose the minimal value of L that ensures a connected overlay. We note that such a bound is not always known. However, as we show in Sections 5.2.1 and 5.2.2, beyond the failure threshold Araneola's overlay exhibits graceful degradation in the face of increasing failure rates, and therefore inaccurate setting of L has a moderate effect on the overlay's connectivity. Moreover, in Sections 5.3 and 8.3, we show that with L equal to 5, Araneola's overlay remains connected despite churn rates exceeding the ones measured over

the Internet and over the Mbone and in a WAN-like setting, respectively. In this paper, we do not deal with dynamically adapting L according to changing churn and failure rates. Note that this approach is also used in other studies of scalable multicast, e.g., both the number of trees/stripes (k) in SplitSream [13] and the number of nodes with which each node exchanges digests in Bullet [45] depend in the expected failure rates.

5.3 Dynamic Evaluation

5.3.1 Methodology

Our model for this evaluation is based on studies of user behavior in multicast groups on the MBone [3], and in file sharing applications [58]. These studies model the join and leave rates of most of the nodes using an exponential distribution. Moreover, both studies observe that a small portion of the nodes have substantially longer life times than others. However, these studies greatly differ in the mean life times they measure: the mean life time measured on the MBone is generally short, e.g., 7 minutes in a typical multicast session, whereas the average measured life time in a file sharing application is roughly one hour.

Saroiu et al. [58] found that only 20% of the nodes in a peer-to-peer lookup system have an uptime of 93% or more. Motivated by this study, we designate a small subset (roughly 7%) of the nodes as *perseverant*. Perseverant nodes are created at the beginning of the experiment and remain active throughout the experiment. Subsequently, every minute, 50 additional (non-perseverant) nodes are awaken, until all nodes (1000 or 2000) are up. Each non-perseverant awaken node joins the multicast group (becomes *active*) with probability 0.5. Otherwise, the node remains *inactive*. This gradual joining is modeled after the Berkeley session in [3]. Throughout the experiment, each non-perseverant node once a minute flips a coin with probability λ in order to decide whether to change its state from active to inactive and vice versa. We experiment with values of λ ranging from 0.01 (yielding a mean life time of 100 minutes) to 0.15 (giving a mean life time of 6.7 minutes). As a baseline, we also experiment with $\lambda = 0$, in which case nodes do not change their states. There are roughly 1000 nodes alive at the end of each experiment with $N = 2000$, (and respectively, 500 when $N = 1000$), regardless of λ, since the join rate is equal to the leave rate. In all the dynamic experiments, we set L to 5 and H to 10.

5.3.2 Join/leave overhead

We now examine the cost of constructing and maintaining the overlay. This overhead is composed of control messages and membership overhead. The membership protocol piggybacks a small and constant (and hence independent of the churn rate) number of bytes on messages sent to the node's neighbors, as calculated in Section 4.3.1. In this section, we measure the join/leave overhead. The size of control messages is fixed, and consists of less than ten bytes. Therefore, we measure the cost of constructing and maintaining the overlay in terms of the number of control messages. We count the total number of control messages received by all the nodes throughout the experiment, and divide this number by the number of joins and leaves occurring in that experiment. We do not separately measure the overhead for join and leave since we cannot fully distinguish between the two. E.g., when a node receives a CONNECT message, we do not know whether to attribute this message to a prior LEAVE event that reduced a node's degree, or to a new node trying to join the overlay. There are roughly 1000 more join events than leave events in experiments with $N = 2000$ (respectively 500 in experiments with $N = 1000$). Table 2 shows the exact number of join and leave events for experiments with 2000 nodes.

λ	# of join events	# of leave events
0.01	1411	387
0.025	2005	955
0.05	2908	1872
0.075	3825	2768
0.1	4690	3650
0.125	5480	4495
0.15	7965	7029

Table 2: The number of join and leave events in experiments with 2000 nodes.

Figure 10 shows the overhead measured for different values of λ with $N = 1000$ and $N = 2000$. Remarkably, the overhead *decreases* as the rate of such events increases, the only exception occurring when λ increases from 0 to 0.01. This rise is due to the facts that (i) when $\lambda = 0$, no leave events occur, and (ii) the overhead associated with a leave operation is bigger than the overhead associated with a join operation (see Section 4.3). But in general, the overhead associated with a join or leave operation decreases as the churn rate rises because when many join and leave events occur concurrently, their costs can be amortized. For example, assume that node n has a degree of L at time t, and at this time node n receives a LEAVE message from one of its neighbors. If the churn rate is low, then the probability that a joining node will send node n a CONNECT request shortly after time t is small. Hence, in this case, node n is required send one or more CONNECT requests in order to increase its degree back to L. However, if the churn rate is high, then there is a higher probability that a joining node will send node n a CONNECT request shortly after the leave event. This eliminates the need for correcting the overlay, and hence reduces the maintenance overhead. Furthermore, we observe that the overhead does not increase with N. This is especially impressive given that the overhead for handling joins in structured overlays based on DHTs increases logarithmically with the number of nodes.

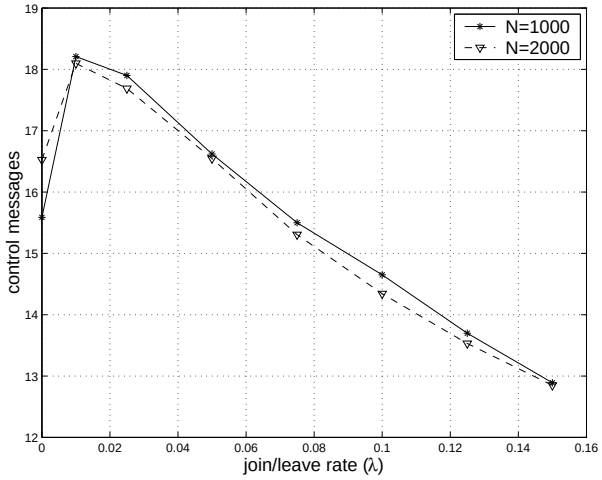


Figure 10: Average cost per join/leave with increasing churn rates for different group sizes, L=5.

Theory versus practice. In the Section 4.3, we analyzed the expected number of control messages incurred by a single join or leave operation occurring after the system has stabilized. We now compare this analyzed overhead to the above measured join/leave overhead.

We found out that the expected join and leave overhead during a stable period is: $L(4p + 10q - 2p^{L+1})$ and $(L+q) * [p(1+4p+10q-2p^{L+1})+q]$, respectively, where p is the percentage of nodes with degree L and $q = 1 - p$. Empirically, when the system is stable and L is set to 5, roughly 92% of nodes have a degree of L (see Figure 5). Substituting 5 for L, 0.92 for p, and 0.08 for q, we get that the expected join and leave overhead is 16.3 and 20.4 messages, respectively.

When $\lambda = 0$, no leave events occur. The measured average cost per join operation in this case is 15.6, which is close to the expected overhead (16.3). The difference between the expected overhead and the measured one stems from the fact that in a static experiment (when $\lambda = 0$), a node with a degree lower than L may receive a CONNECT request from some other node, reducing the number of CONNECT requests it needs to issue itself.

When $\lambda = 0.01$, the system is similar to a stable system, as the rate of join and leave operations is low. In an experiment with $N = 2000$ and $\lambda = 0.01$ there were 1411 join events and 387 leave events. Thus, the expected overhead for a join/leave operation in this experiment is: $(1411 * 16.3 + 387 * 20.4)/1798 \approx 17.2$. Indeed, the measured overhead in this case, 18.2, is close to the expected one.

5.4 The Effect of the Membership Service on the Overlay

In order to evaluate the effect of the membership service on the overlay, we run an experiment with 1000 nodes using the setting of Section 5.1 with the exception that in this experiment the initial distribution of the membership views is skewed as follows: at bootstrap, each node's (including a node that is inactive at bootstrap) membership view contains ten node identities chosen uniformly at random out of a set that including only 10% of the nodes, that is, only 10% of the nodes appear in the initial views.

We run this experiment five times in a static setting and five times in a dynamic setting according to the methodology described in Section 5.3. We compare these overlays to 1000-node overlays obtained using the setting of Section 5.1, where each node's initial view including ten node identities chosen uniformly at random among all 1000 nodes. In all of these experiments, L and H are set to 5 and 10, respectively. We summarize our results in Table 3.

Initial distribution of membership views	Static/Dynamic	Diameter	Connectivity
skewed	static	7–8	4–5
uniform	static	7	5
skewed	dynamic	7	5
uniform	dynamic	7	5

Table 3: The effect of an initially skewed distribution of membership views on the overlay.

As the table shows, in static experiments, the initially skewed distribution of the membership views has a small effect on the overlay: whereas all the overlays obtained using an initially uniform distribution are 5-connected and have a diameter of 7, the overlays obtained using a skewed initial distribution have a connectivity of 4 or 5 and a diameter of 7 or 8. In dynamic settings, the initial distribution of membership views has no effect on the properties of the overlay. Below, we explain these results.

In an experiment with an initially skewed distribution, all the nodes' initial L connect requests are sent to 10% of the nodes. However, each of the nodes in this set can maintain only up to H connections. Upon refusing to accept a connection (due to a high degree), the target node n sends its membership view to the node $\hat{n}$ that issued the connect request, and also adds $\hat{n}$ to its membership view. Assume now that another node n' sends a connect request to n. n rejects this request (due to its high degree), and sends its membership view to n'. Now, n' can send a connect request to $\hat{n}$, and $\hat{n}$ will accept this request. Hence, by limiting each node's degree and by sending the membership view upon a rejection of a connect request, our construction protocol overcomes an initially skewed distribution of the membership views.

In a dynamic setting, an initially skewed distribution of the membership views affects only on the first join operation of each node. Since i) empirically, the views' distribution becomes uniform over time; and ii) prior to leaving the overlay each node saves its membership view to a log file, subsequent join operations will create random links. In addition, leave operations lead to the destruction of non-random links, which are replaced by random links created by subsequent join operations. Therefore, in a dynamic setting, join and leave operations "heal" the overlay, and hence the initial distribution of the membership views has no effect on the overlay's properties.

5.5 Comparison with k-Regular Random Graphs

We have observed that Araneola's basic overlay achieves the important mathematical properties of k-regular random graphs, namely logarithmic diameter and k-connectivity (and hence high robustness). In this section, we compare these properties of Araneola overlays to those measured in centrally constructed k-regular random graphs. Specifically, we compare Araneola overlays to L-regular random graphs created by the algorithm of [61, 43] (as described in Section 2 above), for $N = 1000$ and L = 4, 5, and 6. We summarize our results in Table 4.

Note that an Araneola overlay contains slightly more edges than the corresponding L-regular random graph, since in Araneola roughly 90% of the nodes have a degree of L, and the rest have a degree of L+1. E.g., when L= 4, 5, and 6, an Araneola overlay with 1000 nodes contains on average 49, 43, and 38 (respectively) more edges than an L-regular random graph with 1000 vertices.

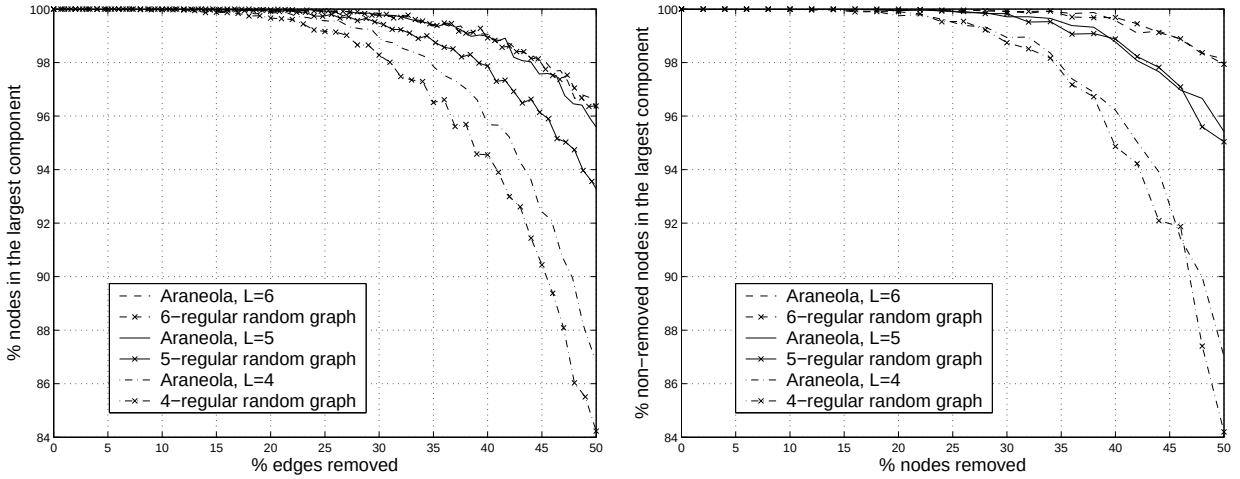
Overlay	Highest diameter	Avg distance	Avg # of disjoint paths
4-regular random graph	11	5.63	4
Araneola, L=4	**11**	**5.49**	**4.01**
5-regular random graph	7	4.71	5
Araneola, L=5	**7**	**4.69**	**5.01**
6-regular random graph	6	4.18	6
Araneola, L=6	**6**	**4.16**	**6.01**

Table 4: Araneola versus a centralized construction of L-regular random graphs, 1000 nodes.

The first column in Table 4 shows the highest diameter measured for each type of overlay. In all of our experiments, the diameter of the Araneola overlay is identical to the corresponding L-regular random graph. The next column presents the average distance between two nodes in the overlay. In all of our experiments, this distance is slightly smaller in Araneola than in the L-regular graph. The average distance between two nodes in the overlay determines the average latency in which multicast messages are received, and hence this parameter is important. An even more important parameter is the average number of disjoint paths between

two nodes in the overlay, presented in the last column of the table. This number determines the robustness of the overlay/graph. In all of our experiments, Araneola contains on average slightly more disjoint paths than the L-regular random graph, again, this is due to the slightly larger number of Araneola edges.

In order to further compare the robustness of Araneola to that of L-regular random graphs, we remove random subsets of edges/nodes from the different Araneola overlays and L-regular random graphs and analyze the resulting graphs. We present our results in Figure 11. As the figure shows, in all of our experiments, Araneola achieves the same robustness as the L-regular random graph or slightly better.



(a) Removing edges, largest component.

(b) Removing nodes, largest component.

Figure 11: Robustness of Araneola versus centralized construction of L-regular random graphs, 1000 nodes.

6 Example of Application-Specific Extension: Exploiting Network Proximity and Bandwidth Heterogeneity

Araneola's basic overlay, like many peer-to-peer systems, treats all nodes and all communication links equally: all nodes have almost the same degree, and all links have an equal likelihood of being used. In reality, however, node capabilities and communication channels are diverse. A wide-area network is typically structured as a collection of LANs, where communication in each LAN is orders of magnitude faster and cheaper than inter-LAN communication.

This section presets an extension to Araneola's basic overlay that exploits network proximity and bandwidth heterogeneity by incorporating additional links between nearby nodes. This extension runs in parallel with and independently of the basic overlay construction and maintenance code presented in Section 4. The extension code has two components: (i) a mechanism for locating nearby nodes; and (ii) a connect_nearby task. The first component discovers nearby nodes and stores them in a set named $nearby\_cand$. The second component uses this set.

Generally speaking, Araneola can use a variety of mechanisms for locating nearby nodes. Our implementation does this as follows: at bootstrap time, each node n measures the network-level hop-count distances to the nodes in its local view using the UNIX tracepath utility, and inserts them to the $nearby\_cand$

set in an ascending order of their network-level hop-count distances from n.

The connect_nearby task closely resembles the connect task presented in Section 4, except that no reduction rules are applied and no REDIRECT messages are sent. Specifically, there are three control messages: CONNECT_NEARBY, CONNECT_OK_NEARBY, and LEAVE_NEARBY, which correspond to CONNECT, CONNECT_OK, and LEAVE. In addition, both L and H are replaced by the parameter NB, which is the maximum number of nearby neighbors the node is willing to be connected to, and the $neighbors$ set is replaced by the $nearby\_neighbors$ set, which holds the node's current nearby neighbors. Note that every node can set its own NB parameter to reflect its available bandwidth. Each CONNECT_NEARBY request is issued to the closest node in $nearby\_cand$, rather than to a random node from the local view.

We evaluate this mechanism over the Internet, running 500 nodes over 25 Planet Lab [54] physical machines, with no two machines at the same site. Out of the 25 Planet Lab physical machines, 10 are located in North America, 10 are located in Europe, and 5 are located in Asia. In all the experiments presented in this section, all the nodes are created simultaneously, and remain up throughout the experiment. Although in principal, each node can choose its own NB parameter, in our experiments, we use the same value of NB for all nodes. We denote an experiment in which each node chooses L random neighbors and NB nearby neighbors as $\langle$L,NB$\rangle$.

It is known that in order to achieve the good properties of k-regular graphs, each node should choose at least three random neighbor [65]. Thus, we run experiments in which each node chooses three random neighbors and three nearby neighbors ($\langle$3,3$\rangle$). We contrast these experiments against experiments in which each node chooses six random neighbors ($\langle$6,0$\rangle$), and against experiments in which each node chooses six nearby neighbors ($\langle$0,6$\rangle$). In addition, we run experiments in which the each node's degree is roughly eight ($\langle$3,5$\rangle$, and $\langle$5,3$\rangle$). Note that all the overlays we experiment with have a low degree, of either 6 or 8, compared to those used in previous systems, e.g., in SplitStream [13], Bullet [45], and Saxsons [59], the maximal node's degree is 16, 10, and 16, respectively. For each selection of $\langle$L,NB$\rangle$, we run three experiments. In all our experiments, more than 97% of the nodes end up with NB nearby neighbors, and more than 90% of the nodes have exactly L random neighbors; the overall average node degrees in experiments with $\langle$3,3$\rangle$, $\langle$6,0$\rangle$, and $\langle$0,6$\rangle$ are almost identical as are those of experiments with $\langle$3,5$\rangle$ and $\langle$5,3$\rangle$.

We quantify the effectiveness of our approach by measuring the average number of physical hops that links in the extended overlay traverse. As we explain below, this metric is significant because a smaller hop-count distance implies reduced communication latencies as well as less stress on physical links. The results are summarized in Table 5. The first column shows the percentage of links between two nodes running on the same machine. The second column shows the percentage of short links with a hop-count distance of 3. These are Internet2 links between machines deployed at different sites belonging to the same enterprise. Finally, the third column shows the average hop-count in the overlay. Clearly, as NB is increased at the expense of L, there are more local and short links and the average number of physical hops that each link traverses is reduced.

Having verified that this mechanism achieves its goal, we next check its impact on the overlay's robustness. We repeat the experiments of Section 5.2, i.e., we remove random subsets of edges and nodes from the overlay graphs and measure the sizes of the largest remaining components. The top two curves in Figure 12 and Figure 13 are for experiments with $\langle$5,3$\rangle$ and $\langle$3,5$\rangle$. These curves are indistinguishable. Slightly below these are the curves for experiments with $\langle$6,0$\rangle$ and $\langle$3,3$\rangle$, which are also conjoined. The bottom curve in both figures is for experiments with $\langle$0,6$\rangle$. Remarkably, the robustness of an overlay with $\langle$5,3$\rangle$ is almost identical to that with $\langle$3,5$\rangle$, and the robustness of an overlay with $\langle$6,0$\rangle$ is virtually identical to that with $\langle$3,3$\rangle$. We believe that this stems from the fact that there is sufficient randomness in the choice of links since: (i)

⟨L,NB⟩	% of links on the same machine	% of short links	Avg hop count
⟨**3,3**⟩	**34.43**	**15.27**	**5.21**
⟨6,0⟩	4.97	6.93	8.69
⟨0,6⟩	74.23	3.4	1.88
⟨**3,5**⟩	**51.18**	**12.25**	**3.82**
⟨5,3⟩	35.6	10.46	5.54

Table 5: Hop-count statistics with different selections of ⟨L,NB⟩.

the nodes in $nearb\_cand$ are chosen from the randomized local view; and (ii) each node is connected to at least 3 random neighbors.

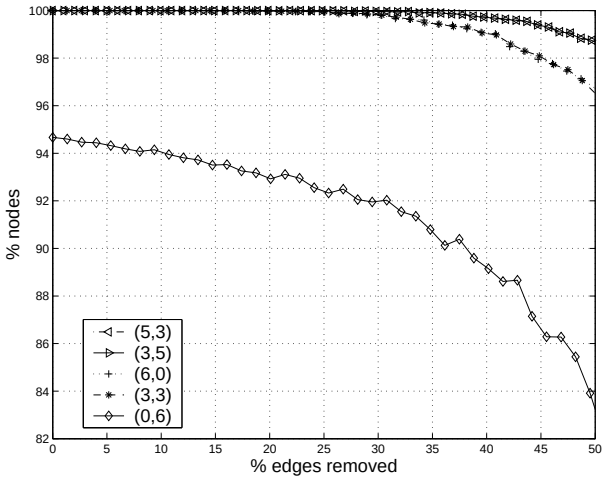


Figure 12: Removing edges, largest component, 500 nodes.

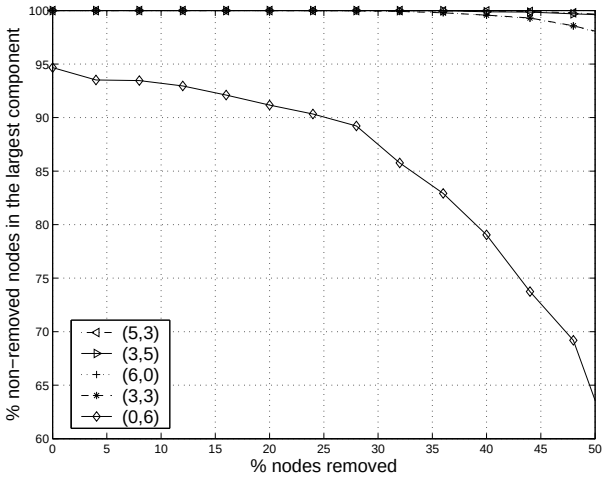


Figure 13: Removing nodes, largest component, 500 nodes.

The curves for experiments with ⟨0,6⟩ show why it is important to choose random nodes as neighbors: in all these experiments, the overlay is partitioned even before we remove any edge or node. Moreover, as the percentage of removed edges or nodes increases, the robustness of the overlay deteriorates much quicker than when random edges are used.

Finally, we measure the average distance between a pair of nodes in an overlay in terms of overlay hop count (i.e., the number of overlay hops in the shortest path between a pair of nodes). We call this distance the average overlay hop count. Similarly to the previous results, the average overlay hop count in an overlay with ⟨5,3⟩ is almost identical to that with ⟨3,5⟩, and the average overlay hop count in an overlay with ⟨6,0⟩ is also almost identical to that with ⟨3,3⟩. As we explained above, this is since there is sufficient randomness in the choice of links. We note that these results confirm that the proximity-driven extension actually reduces the total physical link traversals. This is since this extension substantially reduces the average physical hop count (see Table 5) without increasing the average overlay hop count.

We conclude from the experiments in this section that it is preferable for each node to have three random neighbors, and to allocate the rest of its available bandwidth for communication with nearby nodes or other nodes chosen according to application-specific needs.

7 Multicasting over Araneola

Given Araneola's overlay, it is possible to disseminate data messages by flooding the overlay, as done, for example, in [47]. Using this approach, messages propagate quickly to all the nodes. The price of using this approach is high bandwidth consumption due to the large number of duplicates sent: Each message is sent at least once on each link in the overlay, i.e., at least $NL/2$ times. Some of the messages may be sent simultaneously by both of the nodes that share the link, but only in case this is not the first time each node receives the message. Thus, when the average degree in the overlay is bounded by $L + 0.5$ (as is guaranteed at static times), a flooded message is sent at most $N(L - \frac{1}{2}) + 1$ times. This approach is appropriate for use in low-degree overlays when bandwidth is not a concern (i.e., there is much more available bandwidth than the application needs), and where low latency and reliability are of essence. For example, it is suitable for instant messaging and chat applications, in which payload messages are typically small.

If the multicast payload consists of large messages, it is possible to flood message identifiers on the overlay in lieu of actual messages, and have nodes request missing messages. Although this approach increases the number of *messages* sent, it can dramatically reduce the bandwidth consumption when payload messages are large. The penalty for using this approach instead of flooding is increasing the message latency by a factor of three. Such a penalty is acceptable for numerous non-real-time applications, for example, file sharing applications like BitTorrent [18], software update dissemination, and video streaming. Many scalable ALM systems are geared towards such applications: virtually all overlay-based and gossip-based ALMs have non-optimal message delays since messages traverse a number of hops that do not necessarily bring the packet (network-wise) closer to its destination. Furthermore, a number of ALMs, like Bullet [45] and Overcast [37], exploit the freedom to delay messages in order to achieve a more bandwidth-efficient system design (using large caches at intermediate nodes in Overcast, and obtaining packets on-demand from nodes that have them in Bullet).

If payload messages are large and are not sent frequently, then flooding message identifiers can be effective. However, if many payload messages are sent, then flooding each identifier in a separate message can induce a high load. In order to overcome such situations, one can bundle a number of messages identifiers together, and periodically send this bundle in a *gossip message*. This is a generalization of the identifier-flooding approach, where the system designer can control the tradeoff between delay and overhead according to specific application needs and traffic characteristics: by sending gossip messages more frequently, one reduces the delay, and by sending them less frequently, one reduces the overhead. For example, by setting gossip rounds to 2 seconds as in [21], in a large group of $10,000$ nodes in an overlay with L$= 5$, we get an average (worst-case, respectively) latency of roughly 7 (18, respectively) seconds. We now describe this gossip-based approach in more detail.

7.1 Gossip-Based Multicast

Each node locally divides its time into *gossip rounds*. A gossip round consists of two phases: in the first phase, each node gossips about recent message identifiers and requests missing messages from its neighbors (the *gossip task*), and in the second, the corresponding missing messages are sent.

The gossip-based implementation is presented in Figure 14. *messages* is a FIFO queue of recently received messages. The set *missing_msg* holds identifiers of messages that the node heard of but did not receive. A function *heard_from* maps each identifier in this set to nodes from which it was heard. *recent_mids* holds identifiers of messages received in the latest gossip round along with the identities of the nodes from

which they were received.

Every gossip_round_timeout, a node sends a gossip message to each of its neighbors. A gossip message m sent by a node a to its neighbor n is identified by a message identifier, $m.id$, which includes a's identifier (e.g., IP address and port) and a one byte serial number (cyclic counter). The field $m.degree$ holds a's current degree (line 5). The set $m.ids$ includes recent message identifiers that a has received in the last gossip round, and has not heard about from n (line 6). In addition, Araneola piggybacks message requests on gossip messages instead of sending them in separate request messages. Therefore, m includes a set $m.reqs$ of message identifiers that a is requesting from n. These are messages that a is missing, and has heard their identifiers first from n (line 7). Note that a sends a different gossip message to each of its neighbors. After sending the gossip messages, the first element in each *heard_from* list is moved to the end of that list (line 9) in order to vary the node from which the message is requested, and *recent_mids* is reset (line 10) so as not to gossip about the same identifiers again.

When node a receives a gossip message m from neighbor n, for each identifier id in $m.ids$ such that a message with this identifier is not in the *messages* buffer, id is inserted into *missing_msgs* (line 14) and n is appended to *heard_from(id)* (line 15). In addition, a sends to n all the messages requested in $m.reqs$. When a data message arrives, it is enqueued in *messages*, removed from *missing_msgs*, and its identifier and sender are inserted into *recent_mids* (lines 19–21).

Periodically, old messages are purged from *messages* and *missing_msgs*. This garbage collection mechanism is straightforward, and is omitted from the pseudo code.

7.1.1 Load balancing for single-source multicast

Although Araneola is intended for multi-point to multi-point communication, it can also be used for point to multi-point multicast. When the multicast has multiple uniformly distributed sources, the load on Araneola nodes is naturally balanced: each node sends the same number of gossip messages per round, and each nodes handles roughly the same number of message requests on average. However, if the multicast would be initiated at a single source, then the message requests would most often be sent on a spanning tree of the overlay rooted at the source. This can result in a higher load on inner nodes of the spanning tree.

We propose the following simple solution to this difficulty: Let node n be the single-source in a multicast session. Whenever a new data message is created at n, n sends the message to a random set of nodes (from its partial randomized membership view) instead of sending it to its neighbors. A different set of nodes is chosen each time. This simulates a situation in which messages are created by multiple uniformly distributed sources, and the message requests follow many different spanning trees.

7.1.2 The multicast and management overhead

In this section, we assume that each node is connected to L nodes, there is no packet loss, and the load on Araneola nodes is balanced as described in the previous section. We denote the multicast rate as p data packets per *gossip_round_timeout*.

The multicast overhead. In Araneola, as opposed to other multicast protocols, e.g., Lpbcast [22] and Bullet [45], no duplicate data packets are sent. Whereas in Lpbcast, on average, each node receives $\log N$ copies of each data packet, and in Bullet [45], roughly 10% of received data packets are duplicates, in Araneola, each node receives one copy of each data packet. Since the load on Araneola nodes is balanced,

Data structures:
messages – queue of messages tagged with $m.id$, initially empty.
missing_msg – set of messages identifiers, initially $\emptyset$.
heard_from:missing_msgs $\longrightarrow$ list of nodes.
recent_mids – set of pairs $\langle id, from \rangle$, initially $\emptyset$.
Parameters:
Timeout: gossip_round_timeout.

Gossip task:
1. loop forever
2. sleep (gossip_round_timeout)
/ Send gossip messages to neighbors */*
3. **foreach** $n \in neighbors$
4. create new gossip message m, with new $m.id$
5. $m.degree \leftarrow |neighbors|$
6. $m.ids \leftarrow \{i.id : i \in recent\_mids \wedge i.from \neq n\}$
7. $m.reqs \leftarrow \{i \in$ *missing_msgs*: *heard_from(i).first*$= n\}$
8. send $\langle$GOSSIP,$m\rangle$ to n
/ Update data structures */*
9. move 1st element of each *heard_from(mid)* list to end
10. *recent_mids* $\leftarrow \emptyset$

Event handlers:
11. upon receive $\langle$GOSSIP,$m\rangle$ from n **do**
12. $neighbor(n).degree \leftarrow m.degree$
13. **foreach** $id \in m.ids \wedge id \notin messages$
14. *missing_msgs* $\leftarrow$ *missing_msgs* $\cup\{id\}$
15. append n to *heard_from(id)*
/ Send requested messages to n */*
16. **foreach** $r \in reqs$
17. send $\langle$DATA,message with identifier = $r.id\rangle$ to n

18. upon receive $\langle$DATA,$m\rangle$ from n **do**
19. *messages.enqueue(m)*
20. *missing_msgs.remove(m.id)*
21. *recent_mids* $\leftarrow$ *recent_mids* $\cup\{\langle m.id, n\rangle\}$

Figure 14: Gossip-based multicast.

on average, each node forwards each data packet to one of its neighbors. Hence, the per-node multicast load is p data packets per *gossip_round_timeout*, which is the multicast rate.

The management overhead. In addition to data packets, every *gossip_round_timeout*, each node sends a gossip message to each of its neighbors. Assuming each node is connected to L nodes, the per-node management overhead is L gossip messages per *gossip_round_timeout*. A gossip message sent from a node n to one of its neighbors $\hat{n}$ contains a one-byte serial number, n's identifier (6-bytes), n's degree (one byte), identifiers of recent messages that n has received in the last gossip round whose source is not $\hat{n}$, and n's message requests from $\hat{n}$. Both a message identifier and a message request are represented by eight bytes. Below, we calculate the average size of a gossip message.

Each node sends each message identifier to $L-1$ nodes, and, in the absence of message loss, requests each message from one of its neighbors. Therefore, on average, each gossip message contains $p\frac{L-1}{L}$ message identifiers and $\frac{p}{L}$ message requests (recall that the multicast rate is p data packets per *gossip_round_timeout*). Hence, the average size of a gossip message is $1 + 6 + 1 + 8(p\frac{L-1}{L} + \frac{p}{L}) = 8(1 + p)$ bytes. Therefore, the per-node management overhead is $\frac{1}{gossip\_round\_timeout} \cdot 8(1 + p) = \frac{8(1+p)}{gossip\_round\_timeout}$ bytes per-second. For example, if the multicast rate is 10 data packets per-second and the *gossip_round_timeout* is 5 seconds, then the per-node management overhead is less than 18 bytes per-second.

8 Evaluation of Gossiping over Araneola

We implement the gossip-based multicast module on top of the code for constructing and maintaining the basic Araneola overlay, described in Section 4. In order to run large scale simulations, we run most of our experiments on a LAN [64]. In Section 8.3, we also run WAN-like simulations of Araneola.

We use the standard UDP protocol as the multicast module's transport protocol. With this approach, no retransmissions are sent, and therefore we do not increase the network load at times of congestion, i.e., when there is high message loss. Even without retransmissions, as we show in this section, Araneola achieves full reliability of data delivery despite high churn and message loss rates by disseminating message identifiers on multiple disjoint paths in Araneola's overlay. We use the standard UDP protocol at the available bandwidth rate of each machine. In particular, we never over saturate the network. Designing a flow control mechanism to adjust this rate is orthogonal to our study. For example, the TFRC transport protocol [23] adjusts its transmission rate on a per-connection basis based on prevailing network conditions [45].

We run multiple Araneola nodes per machine, and therefore need to space the gossip rounds sufficiently so as to allow all the nodes running on the same machine to complete their gossip operation during a round. Thus, we chose a fairly large round duration of 5 seconds. When there is only one node per machine, the round duration can be an order of magnitude smaller. In order to construct and maintain Araneola's overlay we used the code described in Section 4.2 with the parameters and timeouts described in Section 5. Throughput this section, we measure the rate at which messages propagate on the overlay in terms of an overlay-level hop count— each message is tagged with a counter, and every node that receives the message increases the counter. We use this approach in order to allow a fair comparison between Araneola and a standard gossip-based multicast protocol, in which the latency is measured in terms of gossip rounds (see Section 8.1.3). The actual propagation rate is the propagation rate in terms of an overlay-level hop count multiplied by the round duration.

In Section 8.1 we evaluate the performance of the multicast layer in static settings, and in Section 8.2,

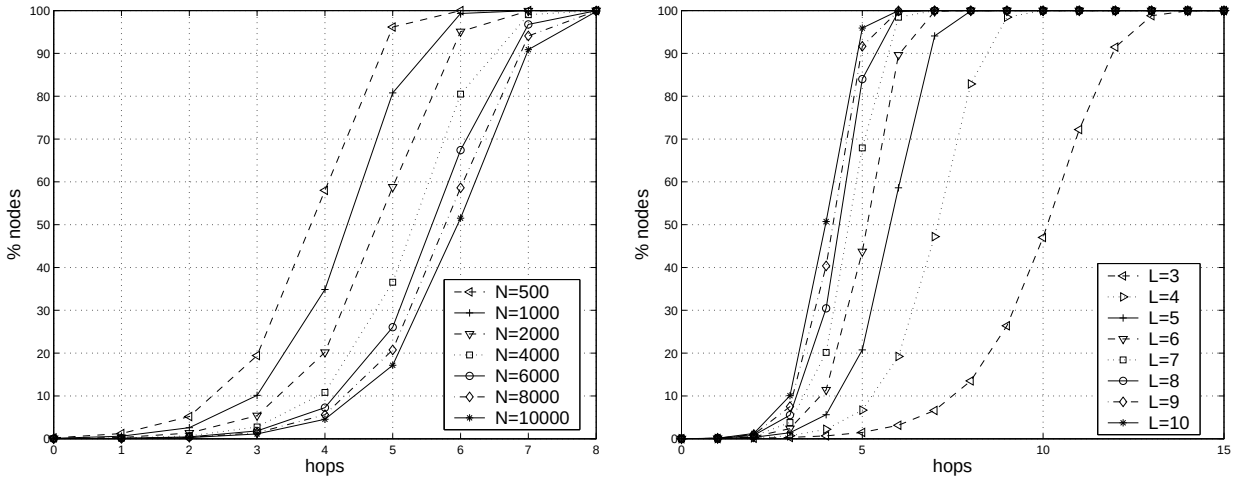
we consider high churn. Finally, in Section 8.3, we evaluate the performance of the multicast layer in a WAN-like setting.

8.1 Static Evaluation

In our static evaluation, all the nodes are created simultaneously, and remain up throughout the experiment. In each round, a single data message is injected into the system (by the application), each time from a different machine. At least 200 data messages are sent in each experiment, and each experiment (with a given number of nodes and choice of parameter settings) is repeated at least twice.

8.1.1 Scalability

We first examine the impact of number of nodes N on the rate at which messages propagate on the overlay. Figure 15(a) depicts the message propagation rates measured for values of N ranging from 500 to $10,000$. L and H are set to 5 and 10, respectively. For each number of hops x, the curves depict the average percentage of nodes that receive a message within x hops. As N increases, messages take longer to propagate, but the slow-down is gradual. The average latency in each of our experiments was close to the average distance between two nodes in the overlay presented in Figure 5. For each N, an average of over 99% of the nodes receive a message within a number of hops equal to the overlay's diameter. On rare occasions, messages were propagated in more hops than the graph's diameter if they reached their destination on a "longer" path before reaching it on the shortest path between the source and destination.



(a) L= 5.

(b) 8000 nodes.

Figure 15: Message propagation rates for different degree Araneola overlays.

8.1.2 The impact of L

Araneola's parameter L affects the overlay's diameter, and hence affects the latency of message delivery. We study the impact of this parameter in runs with 8000 nodes (on 100 Netbed machines) and values of L ranging from 3 to 10. In each experiment, H was set to be L$+5$. Increasing the value of L increases the load,

since a node with degree k sends k gossip messages in each gossip round, and sends each message identifier $k-1$ times (once to each downstream neighbor). However, such increase also reduces the message latency, as shown in Figure 15(b). Each curve in the figure depicts the message propagation rate for a given value of L (in experiments with 8000 nodes). The figure shows that the latency decreases as L increases. When L$=3$, messages reach 99.94% of the nodes within 14 hops, and 100% of the nodes within 15 hops; when L$=4$, messages reach 99.97% of the nodes within 10 hops, and 100% of the nodes within 11; while when L$=5$, messages reach 99.3% of the nodes in 8 hops and all the nodes in 9. The improvement becomes less dramatic as L increases beyond 5.

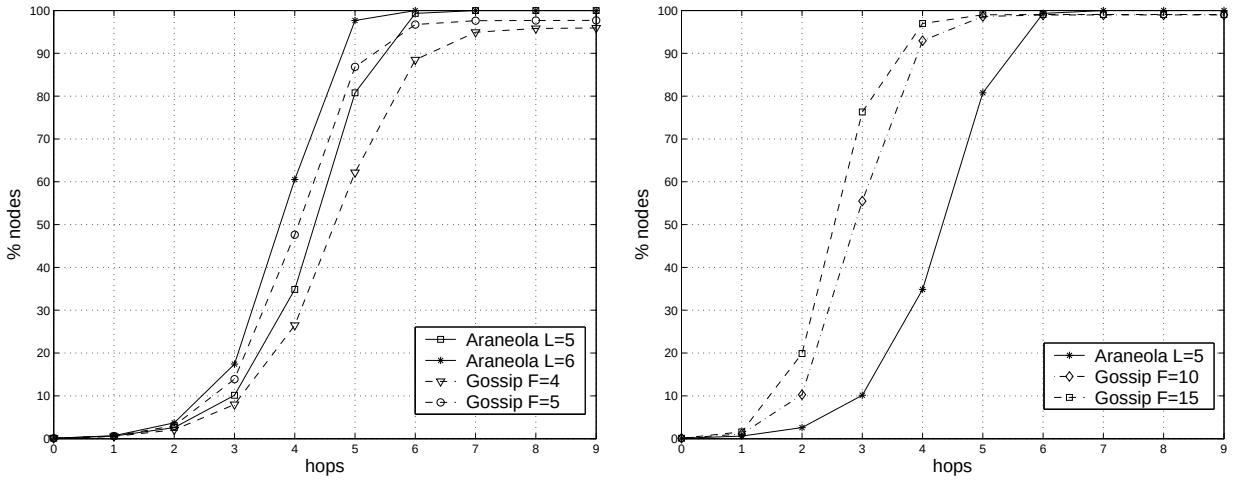
8.1.3 Comparison with gossip protocol

We now compare Araneola to the gossip-based multicast protocol described in [47]. Similarly to Araneola, such a protocol supports dynamic user behavior: the reliability of a gossip-based protocol gracefully degrades with the churn rate, and each join or leave operation incurs a small overhead. In contrast, as explained in Section 2, tree-based overlays like SplitStream [13] and Bullet [45], which are designed for content streaming, do not strive to achieve full reliability under high churn rates and induce higher join/leave overhead than the join/leave overhead incurred by Araneola and a gossip-based multicast protocol.

We have implemented the gossip protocol based on Araneola's gossip-based multicast module. The gossip protocol takes a parameter F, which is its fan-out. Where an Araneola node sends gossip messages to its neighbors, the gossip protocol sends gossip messages to F randomly selected nodes from its membership view. Whereas Araneola sends each message identifier downstream only, the gossip protocol sends all its *recent_mids* to all the chosen targets. Thus, the gossip protocol instantiated with a fan-out of F sends information as many times as Araneola with L$=F+1$. With the gossip protocol, message requests are sent immediately upon receipt of a gossip message, and re-sent periodically in case the requested message is not recovered.

We experiment with 1000 nodes on 20 Netbed machines. In each experiment, 400 messages are sent. Figure 16(a) compares the average message propagation rates of Araneola with L$=5$ and 6 to those of the gossip protocol with the corresponding fan-outs – $F=4$ and 5. Evidently, Araneola propagates information much more effectively than the gossip protocol. Initially, the propagation rates are similar, but after about 6 hops, Araneola continues to effectively propagate the message, while the gossip protocol tapers off. Araneola succeeds in disseminating all the messages to 100% of the nodes in 7 hops with L$=5$, and in 6 hops with L$=6$. In contrast, the gossip protocol only reaches 95.91% of the nodes on average with $F=4$, and 97.69% with $F=5$. Indeed a fan-out of 5 does not suffice for the gossip protocol with 1000 nodes. According to previous studies [42], a fan-out of 14 is required. This is due to the fact that with the gossip protocol only the out-degree (fan-out) is balanced, while the in-degree (fan-in) may be unbalanced. In contrast, Araneola's in-degrees and out-degrees are balanced as all links in the overlay are bi-directional. As more nodes have a given message, the gossip protocol is more likely than Araneola to "waste" its gossip on nodes that already have the message, and therefore is less effective at spreading the information to additional nodes.

The second plot in Figure 16 shows the propagation rate of gossip with fan-outs of 10, and 15 as compared to Araneola with L$=5$. The gossip protocol's propagation rate with the large fan-outs is initially much more rapid than that of Araneola with L$=5$, but after about 6 hops, Araneola already succeeds to reach more nodes than the gossip protocol. While Araneola succeeds in delivering all the messages to all the nodes, the gossip protocol with $F=15$ fails to do so; it reaches only 99.12% of the nodes on average.



(a) Comprable fan-outs for Araneola and gossip.

(b) Larger fan-outs for gossip.

Figure 16: Araneola versus gossip, 1000 nodes.

With $F = 10$, it reaches 98.97% of the nodes on average.

Our measurements of the gossip protocol are close to those reported in [42] although not identical to them. Whereas [42] reports of 100% reliability with $F = 14$, we measure 99.12% reliability with $F = 15$. We believe that this slight discrepancy stems from differences in the evaluation methodology. First, the evaluation in [42] uses simulations; it sends a single data message at a time; and it assumes that nodes are never over-loaded and no messages are lost. In contrast, we run multiple nodes on each machine, the nodes communicate over a real network, and multiple data messages diffuse through the system concurrently. Therefore, we do experience some scheduling delays and message loss, although not often. Second, the system of [42] uses servers in order to have the membership views perfectly uniformly distributed. Since our evaluation does not do so, our membership views are less perfectly "random". Since Araneola is evaluated using exactly the same methodology, our comparison is fair.

8.2 Dynamic Evaluation

One of Araneola's major design goals is to provide an undisrupted multicast service to nodes that are up despite node and link failures and high churn rates. As long as the overlay contains only one connected component, Araneola's multicast module achieves full reliability of message delivery as it floods message identifiers over the overlay's links. In Section 5.2, we studied the fault-tolerance and robustness of the Araneola overlay in static settings and saw that Araneola's overlay remains connected following massive random node and link failures. In this section, we study the performance of the multicast layer under high churn rates using the dynamic simulation scenarios used in Section 5.3. An application message was injected into the system by one of the machines in each round. Between 433 and 476 messages were sent in each experiment. The parameters L and H were set to 5 and 10 respectively.

In each dynamic experiment, for each message m, we define *nodes that are up during m's transmission* to be nodes that have joined Araneola's overlay at least 12 rounds before m's transmission, and did not leave at least 12 rounds after the transmission. We chose 12 as a gross over-estimate. In fact, as we show below,

nodes can normally begin to receive messages reliably immediately upon requesting to join. However, since we run 50 Araneola nodes on each physical machine, which due to contention at the network interface may cause a joining node's messages to be delayed for several rounds, we have chosen to wait additional rounds before considering the node to be up.

In all of our dynamic experiments, each message was received by 100% of the nodes that were up during its transmission. Moreover, messages were delivered with *the same latency as in static runs*. We illustrate this in Figure 17 for $N = 1000$; similar results were obtained with $N = 2000$. The bottom curve depicts the average latency with which messages are delivered for different values of λ. It shows that the latency is unaffected by the join/leave rate. The middle curve shows that, for all values of λ, the average number of hops it takes a given multicast message to reach at least 99% of the live nodes is 6. We did, however, observe a small difference in the message propagation rate: for values of λ up to 0.1, messages reach over 99.9% of the nodes within 6 hops, whereas with $\lambda = 0.125$ and $\lambda = 0.15$, they reach only 99.5% of the nodes within 6 hops. The top curve shows that regardless of λ, it takes 7 hops for messages to reach all the nodes. All the latencies are roughly the same as in static runs with 500 nodes, which is the average number of live nodes in a dynamic experiment with $N = 1000$.

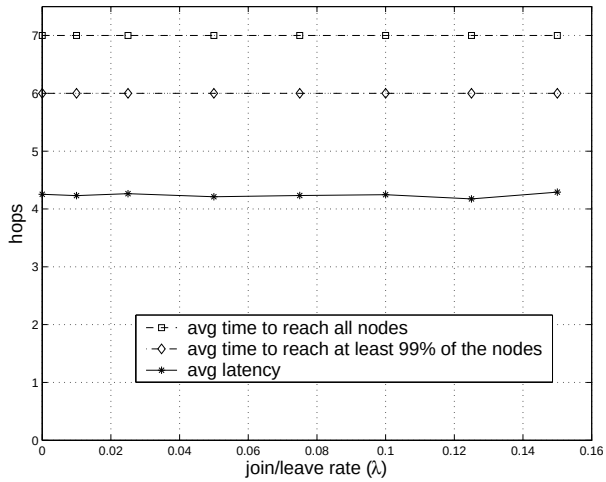


Figure 17: Average latencies for different churn rates, 1000 nodes, L$= 5$.

Comparison with gossip protocol. Churn has little effect on the performance of a gossip-based multicast protocol [4]. In this section, we showed that churn has virtually no effect on the performance of Araneola. Moreover, as the simulations in Sections 8.1.3 and 8.2 show, under churn, Araneola achieves higher reliability than the reliability achieved by a classic gossip protocol in a static failure-free setting, while incurring smaller delay and overhead.

Fast join. The final aspect of the multicast layer we evaluate is how fast it allows joining nodes to begin to receive messages reliably. In order to avoid scheduling race conditions and contention at the network interface, we ran 100 nodes on a single machine, with L$= 5$ and H$= 10$. Our measurements show that a joining node not only receives all the messages sent after its creation, but actually receives 100% of the messages sent up to 6 rounds before its join. This occurs because it usually takes 5 hops for a message to propagate to all of the new node's neighbors. If any of the new node's neighbors receives the message in

5 hops, then the new node will receive this message in the next round, as the 6th hop. We conclude that Araneola incorporates joining nodes into the multicast group without delay.

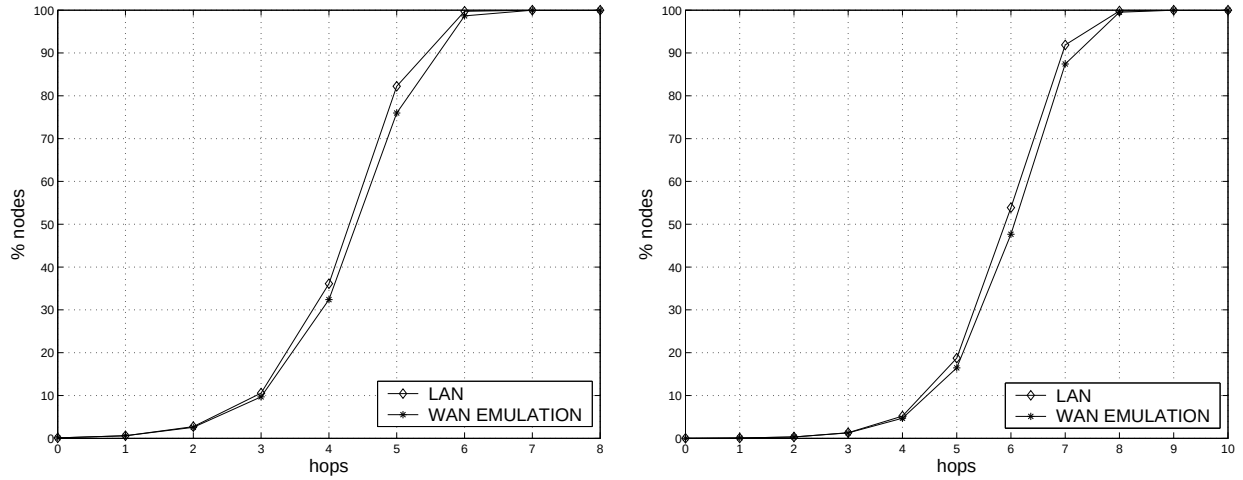
8.3 WAN Emulation

In this section, we report about simulations of Araneola's gossip-based multicast module in a WAN-like setting. We run these simulations in order to evaluate the performance of Araneola in a wide-area setting, in which the message loss rates and delays are much higher than in a LAN setting. Our WAN-like setting is motivated by measurements of upload bandwidth of peer-to-peer clients [58] and measurements of loss rates and RTTs (round trip time) of Internet links [34]. For simplicity, in our WAN-like simulation, we use 5 types of links (see Table 6). In order to measure the worst case reliability in such a setting, a given node's links are all of the same type; this simulates the node's worst link.

Link Type	Loss Rates	RTTs	% of Nodes
Excellent	$< 0.1\%$	0	0.1%
Good	0.1%–1%	$< 62.5ms$	4.9%
Acceptable	1%–2.5%	$62.5ms$–$125ms$	30%
Poor	2.5%–5%	$125ms$–$250ms$	45%
Very Poor	5%–12%	$250ms$–$500ms$	20%

Table 6: Links loss rate and RTT.

We use the setting of Section 8.1 for two group size: 1000 and 8000 nodes. L and H are set to 5 and 10, respectively. In Fig. 18, we compare the message propagation in WAN-like simulations with the message propagation in LAN simulations. As the figure shows, the differences between a WAN-like setting and a LAN setting are small. Below, we explain these results.



(a) 1000 nodes.

(b) 8000 nodes.

Figure 18: Message propagation rates for WAN-like and LAN simulations.

We first note that link latency neither reduces the reliability of message delivery nor increases the latency of message propagation. This is since we measure the message propagation in rounds, and a round duration is an order of magnitude longer than a link latency. The message loss does reduce the reliability. However, as opposed to tree-based multicast systems, in Araneola, as long as the message loss does not exceed a certain threshold (see Section 5.2), failures do not reduce Araneola's reliability. This is since there are L disjoint paths between each pair of nodes in the overlay, and hence each message can be retrieved from L different neighbors. In our WAN-like setting, the failure rates do not exceed the above threshold, and therefore Araneola achieves full reliability in this setting. The message loss, however, does slightly increase the latency, since several messages are received not through the shortest path. However, as the figure shows, this increase is small, since the average link loss is small.

9 Conclusions

We have presented Araneola, a scalable reliable multi-point to multi-point application-level multicast system for dynamic environments. We have evaluated Araneola over both a LAN and a WAN, and have shown that Araneola is scalable. The only aspect of Araneola that varies with the number of nodes is message latency, which increases logarithmically with the group size, whereas Araneola's load, reliability, resilience to message loss, resilience to simultaneous node failures, and overhead for handling join and leave events are all independent of the group size. Araneola can deliver messages with high reliability and predictable latency in the presence of sizable message loss rates, simultaneous failures of a certain percentage of the nodes, and high churn. The failure rates that Araneola can withstand depend on a tunable parameter, L. As the failure rate increases beyond its expectation, Araneola's reliability degrades gracefully. We have also shown how to extend Araneola to exploit available bandwidth for communication with nearby nodes. Such an approach substantially reduces the communication costs and message latency without hurting the overlay's robustness to random failures.

Acknowledgments

We thank Ophir Ovadia for his assistance with implementing part of the code. We thank Dahlia Malkhi and Ziv Bar-Yossef for helpful comments. We are grateful to the Flux research group at the University of Utah, and especially Leigh Stoller and Jay Lepreau, for allowing us to use their network emulation testbed [64] and assisting us with our experiments.

References

[1] Microsoft Combat Flight Simulator 3. http://www.microsoft.com/games/combatfs3/.

[2] I. Abraham, D. Malkhi, and O. Dobzinski. Land: Stretch (1+epsilon) locality aware networks for dhts. In *To appear in the ACM-SIAM Symposium on Discrete Algorithms (SODA04)*, 2004.

[3] K. C. Almeroth and M. H. Ammar. Collecting and modeling the join/leave behavior of multicast group members in the mbone. In *High Performance Distributed Computing (HPDC)*, August 1996.

[4] G. Badishi, I. Keidar, and A. Sasson. Exposing and eliminating vulnerabilities to denial of service attacks in secure gossip-based multicast. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 3:1, March 2006.

[5] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *ACM Sigcomm*, 2002.

[6] S. BANERJEE, S. LEE, B. BHATTACHARJEE, and A. SRINIVASAN. Resilient multicast using overlays. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 2003.

[7] E. Bender and R. Canfield. The asymptotic number of labeled graphs with given degree sequences, *J. Combinatorial Theory Ser.* A 24 (1978), no. 3, 296-307.

[8] R. Bhagwan, S. Savagen, and G. Voelker. Understanding availability. In *2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[9] K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):36–53, December 1993.

[10] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.

[11] B. Bollobas. A probabilistic proofof an asymptotic formula for the number of labelled regular graphs, *European J. Combin.* 1 (1980), no. 4, 311316.

[12] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[13] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, October 2003.

[14] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: a large-scale and decentralized application-level multicast infrastructure. *IEEE J. Selected Areas in Comm. (JSAC)*, 2002.

[15] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evalution of scalable application-level multicast built using peer-to-peer overlays. In *IEEE INFOCOM*, April 2003.

[16] Y. Chawathe. Scattercast: An adaptable broadcast distribution framework. *Special issue of the ACM Multimedia Systems Journal on Multimedia Distribution*, 2002.

[17] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *ACM SIGCOMM*, August 2003.

[18] B. Cohen. Incentives build robustness in BitTorrent. In *1st Workshop on the Economics of Peer-to-Peer Systems*, 2003.

[19] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, Shenker, Stuygis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, 1987.

[20] D. S. E. Multicast routing in a datagram internetwork. PhD thesis, Stanford University, December 1991.

[21] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A. M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *The International Conference on Dependable Systems and Networks (DSN)*, 2001. Full version to appear in ACM Trans. Comput. Syst.

[22] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A. M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, November 2003.

[23] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM 2000*, pages 43–56, Stockholm, Sweden, August 2000.

[24] P. Francis. Yoid: Extending the internet multicast architecture. http://www.aciri.org/yoid/.

[25] J. Friedman. On the second eigenvalue and random walks in random d-regular graphs. Combinatorica, vol. 11, pp. 331-362, 1991.

[26] J. Gemmell, J. Leibeherr, and D. Bassett. In search of an api for scalable reliable multicast. TR MSR-TR-97-17, Jun 1997.

[27] A. Goerdt. The giant component threshold for random regular graphs with edge faults. *Theoretical Comput. Sci.*, 259(1-2):307–321, 2001.

[28] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *21st IEEE International Symposium on Reliable Distributed Systems (SRDS)*, pages 180–189, October 2002.

[29] Y. h. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *ACM SIGCOMM*, August 2001.

[30] Y. h. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast*, 20(8), 2002.

[31] F. Harary. The maximum connectivity of a graph. *The National Academy of Science*, 48:1142–1146, 1962.

[32] D. Helder and S. Jamin. End-host multicast communication using switchtrees protocols, 2002.

[33] M. Hofmann, T. Braun, and G. Carle. Multicast communication in large scale networks. In *Proceedings of Third IEEE Workshop on High Performance Communication Subsystems (HPCS), Mystic, Connecticut*, Aug. 1995.

[34] ICFA-SCIC Monitoring WG. January 2003 Report of the ICFA-SCIC Monitoring Working Group. http://www.slac.stanford.edu/xorg/icfa/icfa-net-paper-dec02/.

[35] Instiute for Simulation and Training. Standard for distributed interactive simulation - application protocols. TR IST-CR-94-50, University of Central Florida, Orlando, 1994.

[36] V. Jacobson and S. McCanne. Using the LBL network whiteboard. Lawrence Berkeley Laboratory, University of California, Berkeley, 1994.

[37] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and H. W. O. Jr. Overcast: reliable multicasting with an overlay network. In *Symp. Operating Systems Design and Implementation (OSDI)*, 2000.

[38] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *ACM/IFIP/USENIX 5th International Middleware Conference (Middleware 2004)*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer-Verlag.

[39] F. Kaashoek and D. Karger. Koorde: A simple degree-optimal hash table. In *2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, 2003.

[40] R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *IEEE Symposium on Foundations of Computer Science*, 2000.

[41] D. Kempe, J. Kleinberg, and A. Demers. Spatial gossip and resource location protocols. In *33rd ACM Symp. on Theory of Computing (STOC)*, 2001.

[42] A.-M. Kermarrec, L. Massouli, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, March 2003.

[43] J. H. Kim and V. H. Vu. Generating random regular graphs. In *Proceedings of the thirty-fifth ACM symposium on Theory of computing*, pages 213–222. ACM Press, 2003.

[44] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *32nd ACM Symp. on Theory of Computing (STOC)*, pages 163–170, 2000.

[45] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, October 2003.

[46] C. Law and K. Siu. Distributed construction of random expander networks. In *IEEE Infocom*, 2003.

[47] M. J. Lin, K. Marzullo, and S. Masini. Gossip versus deterministically constrained flooding on small networks. In *14th International Symposium on DIStributed Computing (DISC)*, pages 253–267, 2000.

[48] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience. In *In Proceedings of the ACM SIGCOMM '03 Conference*, 2003.

[49] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *ACM Symposium on Principles of Distributed Computing (PODC)*, July 2002.

[50] L. Massoulie, A.-M. Kermarrec, and A. J. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *22nd IEEE International Symposium on Reliable Distributed Systems (SRDS)*, pages 47–55, October 2003.

[51] B. D. McKay and N. C. Wormald. Uniform generation of random regular graphs of moderate degree. *Journal of Algorithms*, 11:52–67, 1990.

[52] V. Paxson. End-to-end Internet packet dynamics. In *ACM SIGCOMM*, September 1997.

[53] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 49–60, 2001.

[54] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In Proceedings of ACM HotNets-I, October 2002.

[55] B. Quinn and K. Almeroth. IP Multicast Applications: Challenges and Solutions. RFC 3170, September 2001. Network Working Group.

[56] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *3rd International Workshop on Networked Group Communication (NGC)*, November 2001.

[57] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[58] S. Saroiu, K. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking*, January 2002.

[59] K. Shen. Structure management for scalable overlay service construction. In *the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'04), San Francisco CA*, March 2004.

[60] A. C. Snoeren, K. Conley, and D. K. Gifford. Mesh based content routing using XML. In *18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 160–173, 2001.

[61] A. Steger and N. Wormald. Generating random regular graphs quickly. *Combinatorics, Probab. and Comput*, 8:377–396, 1999.

[62] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM*, pages 19–23, August 2002.

[63] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *IEEE Transactions on Networking*, 11, February 2003.

[64] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Symp. Operating Systems Design and Implementation (OSDI)*, pages 255–270, Boston, MA, Dec. 2002.

[65] N. Wormald. Models of random regular graphs. *Surveys in Combinatorics*, 276:239–298, 1999.

[66] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications*, 2003. Special Issue on Service Overlay Networks.

[67] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault tolerant wide-area data dissemination. In *11th International Workshop Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, June 2001.