AVMEM - Availability-Aware Overlays for Management Operations in Non-cooperative Distributed Systems*

Brian Cho, Ramsés Morales, and Indranil Gupta

Dept. of Computer Science
University of Illinois at Urbana-Champaign
Urbana IL 61801
{bcho2,rvmorale,indy}@cs.uiuc.edu

Abstract. Monitoring and management operations that query nodes based on their availability can be extremely useful in a variety of large-scale distributed systems containing hundreds to thousands of hosts, e.g., p2p systems, Grids, and PlanetLab. This paper presents decentralized and scalable solutions to a subset of such availability-based management tasks. Specifically, we propose AVMEM, which is the first *availability-aware overlay* to date. AVMEM is intended for generic non-cooperative scenarios where nodes may be selfish and may wish to route messages to a large set of other nodes, especially if the selfish node has low availability. Under this setting, our concrete contributions are the following: (1) AVMEM allows arbitrary classes of *application-specified predicates* to create the membership relationships in the overlay. In order to avoid selfish nodes from exploiting the system, we focus on predicates that are random and consistent. In other words, whether a given node y is a neighbor of a given node x is decided based on a consistent and probabilistic predicate, dependent solely on the identifiers and availabilities of these two nodes, but without using any external inputs. (2) AVMEM protocols discover and maintain the overlay spanned by the application-specified AVMEM predicate in a scalable and fast manner. (3) We use AVMEM to execute important availability-based management operations, focusing on range-anycast, range-multicast, threshold-anycast, and threshold-multicast. AVMEM works well in the presence of selfish nodes, scales to thousands of nodes, and executes each of the targeted operations quickly and reliably. Our evaluation is driven by real-life churn traces from the Overnet p2p system, and shows that AVMEM works well in practical settings.

Keywords: Membership protocols, availability variation, predicates, management, distributed algorithms, P2P systems.

1 Introduction

Today's large-scale distributed settings contain hundreds to thousands of hosts, and include Grids [5, 10, 27], peer-to-peer (p2p) systems, and geographically-

* This work was supported in part by NSF CAREER grant CNS-0448246 and in part by NSF ITR grant CMS-0427089.

distributed clusters such as PlanetLab [19]. Modern and emerging distributed applications running in such settings will have to address two challenges: heterogeneous availability variation of underlying hosts, and the requirement for system-wide monitoring operations. Further, these challenges have to work even under non-cooperative situations, where hosts may behave selfishly.

The availability of hosts (i.e., their fraction uptime) in any of these systems varies widely across both time and across hosts. For instance, in the Overnet p2p system 50% of hosts have a 10-day availability lower than 30% [3]. This heterogeneity across space and time is visible even in Grid applications. For instance, Grid5000 designers report that each machine reboots several tens of times *per day*, depending on the applications that are scheduled to run on it [5].

Orthogonally, in addition to this heterogeneity, several researchers and industrial companies have pointed out the dire need for monitoring and management of end-user distributed applications. Jim Gray opined that management was the most difficult problem for any distributed system [18]. The 2005 NSF report on "Grand Challenges in Distributed Computer Systems" lists among its primary concerns real-time management, automated monitoring, and dealing with heterogeneity in distributed systems [7]. Finally, end-user applications routinely form 24% to 33% of the TCO (Total Cost of Ownership) of today's clusters [24].

Finally, it is well-known that p2p and Grid systems (e.g., @Home-style applications, or spread over multiple institutions) consist of many nodes that are selfish and would like to obtain maximum benefit from the overlay, in spite of their low availability. For instance, Adar and Huberman point out in [1] that as many as 70% of nodes in Gnutella are freeloaders. Authors have looked at avoiding the effect of selfish nodes for multicast, e.g., [12], however, we believe we are the first to look at availability-based management tasks under a non-cooperative node model.

The conjunction of the above three concerns motivates the problem of designing middleware that executes *availability-based* monitoring and management tasks for such distributed applications. To stay concrete, we consider four specific types of such availability-based tasks (which we sometimes also refer to as *queries*), with significant and varied uses:

I. Threshold-multicast and Threshold-anycast: Multicast (or anycast) to all nodes with availability $> b$ (where $b \in [0, 1)$), starting from any arbitrary initiator node. This would be useful for both control and data operations. Control operations include selecting a supernode in a p2p system with a minimal threshold availability, e.g., akin to [13, 14, 16]. Data operations include a publish-subscribe or multicast application where packets are sent out to only nodes above a certain availability, e.g., [20]. Such a multicast application would *incentivize* hosts to have higher availability, in order to obtain good reliability.

II. Range-multicast and Range-anycast: Multicast (or anycast) to a node with availability in range $[b, b + \delta] \subseteq [0, 1]$, starting from any arbitrary initiator node. This operation can be used to *fingerprint* characteristics of the nodes within an availability range, e.g., one could find out the average bandwidth of

nodes below a certain availability, in order to correlate the two facts. In addition, threshold anycast would be useful for selection of replica locations for a file [4, 6], and of deployment instances for a distributed Grid application [5].

There are many other availability-based management operations not listed above that may be desired by applications. However, we find that all of the existing overlays in literature are *availability-agnostic* while selecting neighbors, thus making it inefficient to run the above classes of tasks. This observation motivates the need for an *availability-aware overlay*, that would support availability-based management operations like the ones listed above.

1.1 Design Goals, Challenges, and Principles

A decentralized solution to the availability-based management problems just described consists of two components: (I) an overlay among the nodes that helps each node maintain a set of neighbors (or a *membership list*), based on the availabilities of these nodes; and (II) operations to execute the desired management operations by leveraging this overlay.

In building the overlay (challenge (I) above), we face two challenges. The first arises because we consider a system model where nodes may be *selfish*. Under this setting, nodes (especially those with low availabilities) would like to have as many other nodes (possibly of high availability) in their own membership list, and to communicate with them. Further, these selfish nodes may wish to flood the network with copies of a genuine anycast or multicast request they received. To address this challenge, we adopt the philosophy of selecting neighbors of a given node x in a *consistent* manner based on the availabilities of nodes.

Concretely, given a node x and y, let $M(x, y)$ be a binary variable that denotes whether y is a valid entry in x's membership list or not. Consistency requires that the value of $M(x, y)$ depend only on the addresses (IP and port) of x, y, and their availabilities $av(x), av(y)$ as reported by the availability monitoring service. $M(x, y)$ should not be influenced by any external factors such as other nodes in the system, the system size, or churn in the system, etc. Notice that consistency allows both the recipient node y of any message or a third node to verify the value of $M(x, y)$, regardless of other factors in the system. This implies that any node x (selfish or otherwise) will be able to send messages only to other nodes y that are legitimately its neighbors under the consistent predicate, i.e., for which $M(x, y) = 1$.

The second challenge arises from the fact that we would like to maintain connectivity in the overlay as well as support efficient anycast and multicast operations, yet maintain only a small number of neighbors. A small number of neighbors translates to a lower bandwidth, memory, and computation overhead. In order to ensure connectivity, scalability, and efficiency, we require the neighbor selection criteria to be *flexible*, besides being consistent. Our approach addresses this challenge by coupling consistency with *randomization*.

Finally, for challenge (II) above, we would like to execute anycast and multicast operations in a manner that is *fast* (i.e., has low latency), *scalable* (i.e.,

uses a small number of messages), and *reliable* (i.e., manages to complete successfully). We address this challenge by using a variety of techniques, ranging from flooding and greedy approaches, to gossip and simulated annealing.

1.2 Other Related Work and Eliminated Solutions

Using a centralized solution to execute the management tasks mentioned is prohibitive because this would limit the number of simultaneous tasks that can be addressed, especially if these tasks are for the multicast variants above. It is well-known that such central-database solutions are rather ineffective at providing real-time answers to instantaneous queries.

In the realm of decentralized solutions, one potential alternative is to leverage p2p ring-based distributed hash tables (DHTs) such as Pastry [21] or Chord [23]. Such an approach would decide DHT nodeIDs for nodes based on the node's availability, rather than a hash of its IP address. Although this allows tasks to be resolved via the DHT routing algorithm itself, this approach causes an unacceptable amount of churn in the DHTs. This churn arises since a nodeID changes with the node's availability, besides the fact that nodes are continuously going offline and coming online. In addition, when using ring-based DHT routing the latency for answering a range-multicast task is linear in the number of nodes involved, thus making it inefficient.

Another alternative could be p2p solutions that are specially built to support range searches (or range queries) such as skip trees, graphs and others [2, 9, 22, 28], or content-based publish-subscribe architectures like Sub-2-Sub [26]. In this approach, nodes would be organized and placed in the overlay based on their current availability, so that anycast and multicast tasks could be executed by doing a range search on the appropriate availability range. Once again however, there is a high degree of churn in the system; as nodes' availabilities change over time, their positions in the overlay will move around as well. Further, p2p range query structures are known to be difficult under concurrent operations.

Finally, we would like to eliminate broadcast-based solutions that flood out the multicast or anycast to all nodes, since this is inefficient, unscalable, and causes spam to nodes outside the target range.

1.3 Contributions of this Paper

In order to meet the above goals, this paper presents AVMEM which, to the best of our knowledge, is the first proposed availability-aware membership protocol. AVMEM explicitly leverages availability information of nodes in the system while selecting neighbors. AVMEM avoids the effects of selfish nodes, and allows efficient execution of our targeted availability-based management operations. Concretely, each node in AVMEM maintains two small membership lists: a *horizontal sliver* and a *vertical sliver*. The horizontal sliver at node x contains a small (random) subset of nodes with availability "close" to $av(x)$, the availability of x. In contrast the vertical sliver contains a small (random) subset of

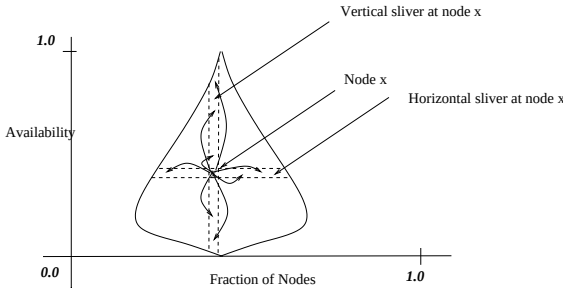

Fig. 1. AVMEM membership lists at a node x: Horizontal Sliver and Vertical Sliver.

nodes from among those with availability that is not in the vicinity of $av(x)$. This is illustrated in Figure 1.

Most importantly, AVMEM supports an arbitrary class of *membership predicates* that are random and consistent. This gives an application developer the choice of a family of AVMEM predicates in order to build the appropriate overlay for their application. The horizontal and vertical slivers at each node are selected in a *randomized and consistent fashion* by using the application-specified predicate. This maintains connectivity, reduces the effect of selfish nodes, and provides efficiency, scale and reliability for the management operations.

We discuss and analyze the family of predicates supported by AVMEM in Section 2. Then, in Section 3, we present decentralized AVMEM protocols that achieve scalable and fast discovery as well as updating of neighbors at each node. Finally, we solve: (1) anycast by using greedy and simulated-annealing approaches, and (2) multicast by using either a flooding or a gossip-based approach. We have implemented AVMEM, and we present trace-based simulations in Section 4. Specifically, we use churn traces from the Overnet p2p system [3] to evaluate and compare the effectiveness the management operations, as well as to microbenchmark the behavior of the AVMEM overlay itself. We conclude in Section 5.

2 AVMEM Membership Graph Predicates

This section presents a range of predicates for creating random and consistent membership graphs (or overlays) that are availability-aware. Section 3 will describe the discovery of membership graphs for any such given predicate.

Basics and Notation: Recall that the availability of a node x, as reported by the availability monitoring service, is denoted as $av(x)$. Further, the identifier (hash-based or IP-port) of node x is denoted as $id(x)$. Given two nodes x and y and a membership predicate, $M(x, y)$ is a binary variable that indicates whether

node x (with availability $av(x)$) should contain node y (with availability $av(y)$) in its membership list or not.

Due to our principles of randomization and consistency, we use the following framework for the AVMEM predicate in the rest of the paper:

$$M(x,y) \equiv \{H(id(x), id(y)) \leq f(av(x), av(y))\} \quad (1)$$

Here, $H(.)$ is a (consistent) normalized cryptographic hash function with range $[0,1]$ - a normalized version of SHA-1 or MD-5 could be used for this purpose. Further, f is a function that takes as input a pair of variables in the range $[0,1]$, and outputs a value that lies in $[0,1]$.

The above predicate means that for given nodes x, y, node x will include y in its membership list only if the value of $H(id(x), id(y))$ is less than the value of $f(av(x), av(y))$. This provides consistency, since the value of $M(x,y)$, as specified by equation 1 above, depends only on the identifiers and availabilities of nodes x and y, but not on anything else in the system. Further, regardless of who evaluates the condition 1 above, it will produce the same result for nodes x, y.

Since we assume that H is a fixed and well-known function, the actual AVMEM predicate is thus determined by the nature of f. For instance, if $f(.,.) = p, (p \in [0,1])$, then we derive a random overlay (like SCAMP or Cyclon), but with the additional property of consistency. In other words, for this example, given two nodes x and y, then $M(x,y) = 1$ consistently with probability p.

Section 2.1 next discusses a family of interesting AVMEM predicates specified under the framework of equation (1). Section 2.2 analyzes these predicates.

2.1 A Family of Availability-Aware AVMEM Predicates

We consider a family of interesting predicates that leverage the known probability distribution function (PDF) of the availability variation in a given system. Notice that such information can be collected and analyzed offline by either a crawler or a central server. This information can then be communicated to all nodes at pre-run-time and used consistently. Suppose the PDF of the availability distribution of the system is specified as $p : [0,1] \rightarrow [0,1]$, i.e., $p(a) \cdot da$ is the *fraction* of nodes with availability between a and $(a - da)$, when $da \rightarrow 0$. Then, our canonical AVMEM predicate is specified as:

$$f(av(x), av(y)) = \begin{cases} hs(av(x), av(y), p(.)) & \text{if } |av(x) - av(y)| < \epsilon. \text{ [Hor. Sliver]} \\ vs(av(x), av(y), p(.)) & \text{otherwise. [Vertical Sliver]} \end{cases}$$

Recall that a horizontal sliver at node x is defined as a partial list of nodes (called horizontal sliver neighbors of x in the overlay) with "similar" availability as node x. According to the above framework, we use an availability range of $(\pm\epsilon)$ around $av(x)$ as candidate nodes for the horizontal sliver at node x. The value of ϵ is fixed globally, and *does not depend on the target ranges of multicast or anycast operations (or vice-versa)*. Our experiments find that using $\epsilon = 0.1$ suffices to give good scalability and reliability for management tasks.

To understand the horizontal sliver concept intuitively, the reader may realize that the horizontal sliver is somewhat like similar notions in DHTs, i.e., like the "leaf table entries" in Pastry [21], and the "successors/predecessors" in Chord [23]. However, our setting is different since those systems deal with hash nodeIDs, while we are dealing with availability space instead. The horizontal sliver helps to maintain a connected overlay among nodes with availability around $av(x)$. Notice that if there are M such nodes, the number of neighbors has to be $O(log(M))$, selected uniformly at random, for connectivity to hold w.h.p. [8].

On the other hand, a vertical sliver at node x is defined as a random sample of nodes with availabilities ranging all the way from 0 to 1. The goal of a vertical sliver is to maintain connectivity throughout the system via a sufficient number of "long-distance" links (in availability space) among nodes. This is most akin to the routing table entries in Pastry or Chord DHTs [21, 23]. However, once again, we are dealing with the availability space rather than hashed nodeIDs, thus our problem setting is quite different.

Below we describe and analyze several AVMEM predicates. Some of these predicates will assume knowledge of the expected system size (i.e., number of online nodes) as a parameter N^*. Just like the availability PDF, the value of N^* can be calculated offline by crawlers, and communicated to all nodes consistently. N^* would not be changed even if the actual number of online nodes changes. Thus, N^* need not be accurate - our algorithms and analysis hold even when the actual system size is off by a constant factor from the value of N^*.

Below, we first discuss several options for selecting the vertical sliver (i.e., different vertical sub-predicates) and then for selecting the horizontal sliver (i.e., different horizontal sub-predicates).

I. Vertical Sub-Predicate Possibilities: There are several ways of specifying the vertical sliver sub-predicate, i.e., $vs()$. We discuss three options below, in increasing order of complexity. We are most interested in the second option and analyze it in detail in Section 2.2. The first option we discuss is availability-independent:

$vs(av(x), av(y), p(.)) = d_1, d_1 = O(log(N^*))$ [I.A: Constant Vertical Sliver]

This predicate works best in a system where any node is equi-probable of having any given availability. That is, the availability PDF distribution is a uniform one.

However, distributed systems rarely have homogeneous availability PDFs. This motivates us to consider other predicates that are more expressive. We derive a very generic vertical sliver sub-predicate:

$vs(av(x), av(y), p(.)) = min(\frac{c_1 \cdot log(N^*)}{N^* \cdot p(av(y))}, 1.0)$ [I.B: Logarithmic Vertical Sliver]

Here, c_1 is a constant. Section 2.2 proves that this predicate ensures a *uniformity of coverage of the availability space* (Theorem 1). In other words, for any availability range $[b, b + \epsilon]$ (non-overlapping with $[av(x) - \epsilon, av(x) + \epsilon]$), a node

x will have the same expected number of vertical sliver neighbors in this range, *regardless of the value of b*.

Finally, one may desire that the density of vertical sliver neighbors in an infinitesimal interval around a value b becomes smaller and smaller as the absolute value of $|b - av(x)|$ becomes larger and larger. This would provide an overlay somewhat akin to Pastry routing table entries and Chord finger table entries, where neighbors are chosen with exponentially increasing distance as one moves away (there, in the hashed id space). This is realized by the following predicate, as proved in Corollary 1.1 of Section 2.2:

$$vs(av(x), av(y), p(.)) = min(\frac{c_1 \cdot log(N^*)}{N^* \cdot p(av(y)) \cdot |av(y) - av(x)|}, 1.0)$$

[I.C: Logarithmic-Decreasing Vertical Sliver]

II. Horizontal Sub-Predicate Possibilities: Just like for vertical slivers, there are several possible horizontal sliver sub-predicates. We enumerate two of them below. The second of these predicates is more interesting, and is analyzed in Section 2.2.

The first option is to select a constant fraction of the nodes that lie in the availability range $[av(x) - \epsilon, av(x) + \epsilon]$. The predicate is:

$$hs(av(x), av(y), p(.)) = d_2, d_2 = O(log(N^*))$$ [II.A: Constant Horizontal Sliver]

Although this ensures connectivity w.h.p. among the nodes in this availability range, it involves too many nodes. Specifically, it is possible that the range $[av(x) - \epsilon, av(x) + \epsilon]$ contains much fewer nodes than N^*. This raises the possibility that the size of the horizontal sliver at a node x can be reduced. This leads us to the following predicate:

$$hs(av(x), av(y), p(.)) = min(\frac{c_2 \cdot log(N^*_{av(x)})}{N^{*min}_{av(x)}}, 1.0)$$

[II.B: Logarithmic-Constant Horizontal Sliver]

Here, c_2 is a constant. This formulation involves two new parameters - $N^*_{av(x)}$ and $N^{*min}_{av(x)}$. First, $N^*_{av(x)}$ is the expected number of online nodes in the availability range $[av(x) - \epsilon, av(x) + \epsilon]$. Mathematically, this can be derived from the PDF of the availability distribution. That is, $N^*_{av(x)} = N^* \times \int_{av(x)-\epsilon}^{av(x)+\epsilon} p(a)da$, where N^* = the stable system size. Second, $N^{*min}_{av(x)}$ is the *minimum* number of expected online nodes present in any availability interval of width ϵ that lies wholly within $[av(x) - \epsilon, av(x) + \epsilon]$. This can also be calculated from the PDF of the availability distribution as follows: $N^{*min}_{av(x)} = N^* \times (min\{\int_v^{v+\epsilon} p(a)da, [v, v+\epsilon] \subseteq [av(x) - \epsilon, av(x) + \epsilon]\})$.

Note that these values can be easily calculated from a discretized PDF distribution of the system created from a small sample set of nodes. $\int_v^{v+\epsilon} p(a)da$ is merely the number of nodes that have availability lying in this interval, divided by the total number of entries in the discretized PDF.

Section 2.2 shows, via Theorems 2 and 3, that the logarithmic constant vertical sliver sub-predicate maintains connectivity w.h.p. among all nodes lying in the range $[av(x) - \epsilon, av(x) + \epsilon]$.

2.2 Analysis of AVMEM Predicates

In this section, we show that the logarithmic vertical sliver ensures uniformity of coverage in the availability space (Theorem 1), the logarithmic-constant horizontal sliver ensures connectivity among online nodes whose availabilities lie close to each other (Theorem 2), and that the above two sliver rules together ensure a small, scalable set of online neighbors for each node in the system (Theorem 3).

Theorem 1: The logarithmic vertical sliver sub-predicate (equation I.B) ensures that, given a node x, for any $a \in [av(x)-\epsilon, av(x)+\epsilon]$, the expected *number* of online nodes with availability in an (infinitesimally small) interval around a, that are vertical sliver neighbors of node x, does not depend on the value of a.
Proof: The expected number of online nodes, in the vertical sliver of node x, that have their availabilities lying in an interval of size da around a, is given as $=p(av(y))da \cdot N^* \times \frac{c_1 \cdot log(N^*)}{N^* \cdot p(av(y))} = c_1 \cdot log(N^*)da$. This is independent of a. □

Corollary 1.1: The logarithmic-decreasing vertical sub-predicate (equation I.C) selects online neighbors that are exponentially increasing distances from node x, where distances are measured in the availability space $av(.)$. (The proof follows along similar lines as Theorem 1.)

Theorem 2: The logarithmic-constant horizontal sliver (equation II.B) sub-predicate ensures that for a given node x, the sub-overlay consisting of all online nodes with availabilities in the interval $[av(x) - \epsilon, av(x) + \epsilon]$ is connected w.h.p.
Proof: For the given node x, define X^+ as the set of all online nodes (other than x itself) that have availability $\in [av(x), av(x)+\epsilon]$. Similarly, define X^- as the set of all online nodes (other than x) that have availability $\in [av(x) - \epsilon, av(x))$. We will show the proof in three parts: (i) the sub-overlay graph of nodes in X^+ is connected w.h.p., (ii) the sub-overlay graph of nodes in X^- is connected w.h.p., and (iii) x knows at least one node in X^+ and at least one node in X^- w.h.p.

For any node u, define $N^{*+}_{av(u)}$ and $N^{*-}_{av(u)}$ as the expected number of online nodes lying respectively in the upper half and lower half of the interval $[av(u) - \epsilon, av(u) + \epsilon]$. That is, $N^{*+}_{av(u)} = N^* \times \int_{av(u)}^{av(u)+\epsilon} p(a)da$, and $N^{*-}_{av(u)} = N^* \times \int_{av(x)-\epsilon}^{av(u)} p(a)da$.

We first prove (i), and the proof of (ii) follows analogously. For any node $y \in X^+$, notice first that the interval $[av(y) - \epsilon, av(y) + \epsilon]$ wholly contains the interval $[av(x), av(x) + \epsilon]$. We use a well-know result from [8] that in a graph of M nodes, if each node has $\Omega(log(M))$ neighbors that are selected at random, then the graph is connected w.h.p.

Firstly, from the definition of the logarithmic-constant horizontal sliver rule, notice for each node u that belongs to X^+, the probability of y picking u as

neighbor is independent of where $av(u)$ lies. Thus, neighbors are picked uniformly at random. Secondly, we need to show that if there are $M = N^{*+}_{av(x)}$ nodes in the interval X^+, each node in that interval has an expected $\Omega(log(M))$ online neighbors lying in X^+. From the horizontal sliver rule at node y, the expected number of online nodes from the interval X^+ that y has as neighbors is:

$$
\begin{aligned}
&= \int_{av(x)}^{av(x)+\epsilon} (c_2 \cdot \frac{log(N^*_{av(y)})}{N^{*min}_{av(y)}} \times (N^* \cdot p(a)))da \\
&= \frac{c_2 \cdot log(N^*_{av(y)})}{N^{*min}_{av(y)}} \cdot N^{*+}_{av(x)} \\
&\geq c_2 \cdot log(N^*_{av(y)}), \text{ (since } N^{*+}_{av(x)} \geq N^{*min}_{av(y)}) \\
&\geq c_2 \cdot log(N^{*+}_{av(x)}), \text{ (since } N^*_{av(y)} \geq N^{*+}_{av(x)})
\end{aligned}
$$

This completes the proof of (i), and thus (ii). Finally, to prove (iii), notice that we can derive, based on the same reasoning as above, the probability of x knowing at least one node in the set X^+, and at least one node in X^-, as:

$$
\begin{aligned}
&\geq (1 - (1 - \frac{c_2 \cdot log(N^*)}{N^{*+}_{av(x)}})^{N^{*+}_{av(x)}}) \times (1 - (1 - \frac{c_2 \cdot log(N^*)}{N^{*-}_{av(x)}})^{N^{*-}_{av(x)}}) \\
&\geq (1 - e^{-c_2 \cdot log(N^*)}) \cdot (1 - e^{-c_2 \cdot log(N^*)}) \\
&\geq (1 - \frac{2}{(N^*)^{c_2}})
\end{aligned}
$$

□

Theorem 3: The logarithmic-constant horizontal sub-predicate (equation II.B) and the logarithmic vertical sub-predicate (equation I.B) , together, ensure that the total expected number of online neighbors (vertical sliver + horizontal sliver) at a given node x: (1) is at most $(N^*_{av(x)} - 1 + c_1 \cdot log(N^*))$; and (ii) $O(log(N^*))$ if $N^{*min}_{av(x)} = \theta(N^*)$.

Proof: Consider a node x. From the discussion of Theorem 1's proof, the expected number of online vertical sliver neighbors at x is:

$= \int_0^{av(x)-\epsilon} c_1 \cdot log(N^*)da + \int_{av(x)+\epsilon}^1 c_1 \cdot log(N^*)da \leq c_1 \cdot log(N^*)$

Since the horizontal sliver at node x can contain at most $(N^*_{av(x)} - 1)$ nodes, this proves the part (i) of the theorem.

To show (ii), we use a similar derivation as in the discussion of Theorem 2's proof. We can show that the expected number of online horizontal sliver neighbors of node x is:

$\leq \int_{av(x)-\epsilon}^{av(x)+\epsilon} (c_2 \cdot \frac{log(N^*_{av(x)})}{N^{*min}_{av(y)} \cdot \epsilon} \times (N^* \cdot p(a)))da = c_2 \cdot \frac{log(N^*_{av(x)})}{N^{*min}_{av(y)}} \times N^*$

Since $N^{*min}_{av(x)} = \theta(N^*)$ and $N^*_{av(x)} \leq N^*$, this is $O(log(N^*))$. □

3 AVMEM Maintenance and Management Operations

Here, we first discuss in Section 3.1 how nodes discover their AVMEM neighbors according to any application-specified predicate. Then, Section 3.2 describes how the anycast and multicast operations are executed atop the AVMEM overlay.

3.1 AVMEM Membership Maintenance

In this subsection, we first describe the techniques used by AVMEM to *discover and maintain* neighbors, i.e., horizontal sliver (HS) and vertical sliver (VS) neighbors, in conformity with the application-specified AVMEM predicate. We then analyze the optimality of this protocol, and check whether the memory, bandwidth, and discovery time scale to medium-scale systems.

For discovery and maintenance, we leverage two types of existing services in a black-box manner. These services are:

1. an *availability monitoring service*, e.g., centralized, or distributed such as AVMON [17]; and
2. a *decentralized shuffling partial membership service*, e.g., SCAMP [8], CYCLON [25], T-MAN [11], LOCKSS [15].

An availability monitoring service is defined as one that can be queried for the long-term availability (e.g., raw, or aged) of any given node. It returns an answer that is reasonably accurate, and that is reasonably consistent over time. The level of accuracy and consistency of course depends on the actual availability monitoring protocol itself. The more accurate and consistent it is, the better our AVMEM discovery will perform. For our practical implementation, we leverage our own availability monitoring service called AVMON [17]; our experiments show that this gives good results.

A decentralized shuffling membership service has a node maintain a random list of some of the nodes in the system (irrespective of any predicate). This is a *weakly consistent list* that is incomplete, and may even contain stale entries. Further, this list is shuffled, i.e., its contents are continuously changed by the underlying shuffling protocol, so that given a node y and node x that stay long enough in the system, the entry for node y will *eventually* appear in the shuffled list at node x. For our practical implementation, we could have chosen any one of existing systems such as SCAMP [8], CYCLON [25], T-MAN [11], LOCKSS [15], etc.. However, we chose to use our AVMON implementation's underlying *coarse view* mechanism [17], which fulfills the requirements of shuffling membership. This simplifies the overall design of our system, and Section 4 shows this approach performs well.

Given the above two services, the core AVMEM maintenance protocol consists of two sub-protocols: (I) a Discovery sub-protocol, and (II) a Refresh sub-protocol. The discovery protocol enables nodes to discover new AVMEM relationships and thus HS and VS neighbors. On the other hand, the refresh protocol checks whether existing HS and VS neighbors still satisfy the predicate, and eliminates them if they do not. Each sub-protocol is elaborated below.

I. Discovery Sub-Protocol: At any given node x, the discovery protocol runs periodically, i.e., once every *protocol period* time units (typically 1 minute). It iterates through the entries in the coarse view (i.e., the shuffled membership list). For each entry node y that is not already in $HS(x) \cup VS(x)$, it queries the availability monitoring service for the availability of y, and checks the AVMEM predicate to see if y is a valid HS or VS neighbor of x. If one of these sub-predicates evaluates to true, then y is included in $HS(x)$ or $VS(x)$, as appropriate. We will soon analyze the discovery time of this protocol.

II. Refresh Sub-protocol: The refresh sub-protocol iterates through the entries of the $HS(x)$ and $VS(x)$ lists. For each node y in these lists, the sub-protocol queries the availability monitoring service for y's current availability, and evaluates the appropriate AVMEM predicate to see if $M(x, y) = 1$ or not. If $M(x, y)$ has become 0, then y's entry is deleted from the appropriate list. It is easy to see that once $M(x, y)$ becomes false, node x will delete y from its AVMEM membership list within a worst case time of 1 protocol period. In our implementation, we found that using a refresh period of 20 minutes suffices to maintain reasonable correctness of AVMEM predicates.

Discovery Protocol - Optimality and Reality Check: The underlying shuffling membership protocols we are considering (SCAMP, T-MAN, CYCLON, LOCKSS, AVMON's coarse view) all maintain a view of size v at each node, where the entries in this view are randomly selected as well as continuously shuffled[2]. For AVMEM, we are concerned about the memory, computation, and bandwidth spent by a node on the one hand, and the discovery time for neighbors on the other hand. The former three scale linearly with v - memory is of course v, computation comes from evaluating the predicate periodically for each entry in the view (thus v), and bandwidth from fetching the availability information for these entries ($O(v)$).

Discovery time is defined as follows: given a pair of nodes x and y for which $M(x, y) = 1$, this is the time until x actually includes y in its $HS(x)$ or $VS(x)$, as appropriate. The discovery time depends on the operation of the underlying shuffling protocol, but fortunately, the fact that there is constant shuffling tells us that the expected time for a given node y to appear in x's view is $O(\frac{N}{v})$.

In order to optimize the above concerns, we thus wish to minimize $f(v) = v + \frac{N}{v}$. Differentiating this with v, gives $\frac{df(v)}{dv} = 1 - \frac{N}{v^2} = 0$, or $v = O(\sqrt{N})$. This is a reasonably small number for medium-scale systems. Even for $N = 100,000$, $v = \sqrt{N} \simeq 320$. With 20 B per entry and a 1 minute protocol period, the memory is 6.3 KB, and the bandwidth is 105 Bps. Finally, if the average discovery time is $\frac{N}{v}$ protocol periods, this turns out to be around 5 hours. This is a reasonable amount given that large-scale Grid computations run for several days, users survive in p2p systems for months, and PlanetLab nodes are up for years.

[2] Since we are using AVMON [17], this v would be the same as *cvs* in [17], i.e., AVMON's "coarse view size".

3.2 Management Operations over AVMEM

In this section, we describe algorithms for executing the four operations laid out in Section 1, namely: threshold-multicast, threshold-anycast, range-multicast, and range-anycast. For ease of exposition, we first discuss the two anycast operations, and then the multicast operations.

I. {Threshold,Range} Anycast: We discuss how to route an anycast message intended for range R - a threshold anycast follows a similar approach, where the range R stretches from the threshold to 1.0. A node x receiving an anycast message checks to see if it itself lies within range R - if yes, then the anycast is successful and we are done. Each anycast has a TTL (time-to-live) that is decremented by 1 at each virtual hop. If this TTL value is 0 the message is not forwarded. In any other case the message is forwarded to another node. We discuss three approaches for forwarding of an anycast below.

• Greedy Forwarding: Node x forwards the multicast to an AVMEM neighbor that lies inside R. If there is no such neighbor, x selects as the next hop the neighbor whose availability is closest to r.

• Retried Greedy Forwarding: To increase the reliability for anycasts, we allow nodes to retry a prospective next-hop if the previous candidate was not responsive (i.e., was found to be offline). To implement this, we introduce an integer parameter $retry$, initialized to k at the initiator. Each forwarded message carries the value of $retry = k$. This parameter determines the number of nodes tried using the greedy metric, before dropping the message. Specifically, each next-hop node is required to acknowledge receipt of the anycast message - failing this, the previous hop node will decrement the value of $retry$ by 1, and retry its next-best neighbor, according to the greedy metric (i.e., distance to range target R). The retrying stops when either $retry$ reaches 0, or there are no more next-best nodes left in the AVMEM neighbor list of node x.

• Simulated Annealing: An alternative approach is to follow simulated annealing, where the probability of choosing a random next-hop is high initially (in the first few hops) but decreases as the anycast proceeds. Specifically, we choose $p = e^{-\Delta/ttl}$, where ttl = remaining time to live, and Δ = the Euclidean distance between the edge of R and the availability of the current next-hop under consideration. At each hop, a random next-hop can be selected (from among the AVMEM neighbors) with probability p, as the list of neighbors is traversed, otherwise the greedy approach is used (with probability $(1 - p)$).

A few notes about the above approaches. Each of the above three variants naturally has three flavors, depending on whether only the horizontal sliver neighbors of x are used (HS-only), only the vertical sliver is used (VS-only), or whether both are used (HS+VS). To be generic, we referred to the considered set of sliver neighbors as merely "AVMEM neighbors" above. Thus, we have a total of nine algorithms. Section 4 presents data on the most promising variants.

Further, when node x is considering potential next-hops for an anycast, it uses *cached* values of availabilities for its neighbors. Typically, these cached values were fetched the last time the refresh operation was done at node x - this eschews

querying the availability service for each forwarded message. Section 4 evaluates how much caching allows flooding attacks by selfish nodes.

II. {Threshold,Range} Multicast: For these operations, we once again consider only the range R; the threshold-based variant follows similarly. The multicast operation follows a two-stage process: an anycast into the range R, followed by a multicast within the range. The anycast follows the techniques listed above. Hence, we now discuss multicast only when the initiator is within the range R. Once a node x has received a multicast message M for a range R (where $av(x) \in R$), it can use one of two approaches for forwarding it:

• Flooding: Node x forwards the multicast to *all* its AVMEM neighbors that lie in range R. Any duplicate copies of the multicast are ignored, and the forwarding is done only once. This is a highly reliable approach, but is wasteful since each node will receive multiple copies of the multicast - in the worst case, it may receive one copy from each of its in-neighbors.

• Gossip: To avoid the above overhead, we use a gossip-based approach. Here, node x (after receiving the multicast) *gossips* the multicast M. It does so periodically - once every *protocol period* seconds, it selects up to $fanout$ of its AVMEM neighbors: (1) whose availabilities lie within the range R, and (2) to whom x has not already forwarded M. These neighbors could be selected randomly, but for our implementation we use a deterministic iteration through the list in order to select gossip targets. The node repeats the above process for N_g protocol periods after it first receives the multicast. Any duplicate copies of the multicast it receives are eliminated. We select N_g and fanout so that $(N_g \times fanout) = log(N^*)$, thus ensuring dissemination w.h.p. via gossip [8].

Just as for anycast, there are three variants for each of the above two approaches - HS-only, VS-only, and HS+VS, depending on which set of AVMEM neighbors are used for the operations. This gives us a total of six algorithms. We implemented all these options, and Section 4 presents data from the best ones.

4 Experiments

We implemented AVMEM in C/C++, and present evaluation results from a discrete event simulation. In order to be realistic, we inject churn (availability variation) traces from the Overnet p2p system [3] into our system. These traces were originally collected over a 7 day period, at 20 minute intervals, for a fixed population of 1442 hosts, and are injected as such. By default, we build and use AVMEM overlays using the two sub-predicates of Logarithmic Vertical Sliver (equation I.B) and Logarithmic-Constant Horizontal Sliver (equation II.B), from Section 2. We evaluate both the AVMEM overlay (Section 4.1) as well as the management operations atop it (Section 4.2).

4.1 Microbenchmarks: AVMEM Overlay Properties

Overlay Properties: We evaluate whether the number of horizontal and vertical sliver neighbors in our implementation follow theoretical predictions. The

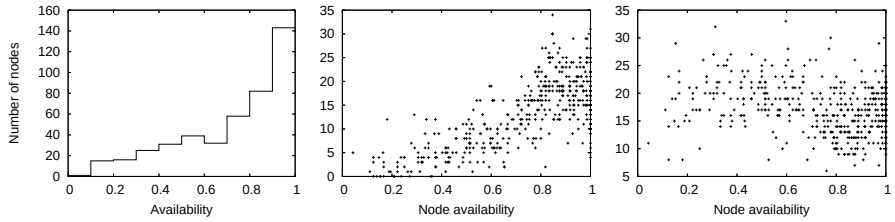

Fig. 2. System Snapshot of Online Nodes showing: (a) the distribution of online nodes (b) the size of horizontal slivers and (c) the size of vertical slivers with respect to availability (each dot in the plot stands for a node). There are 442 online nodes.

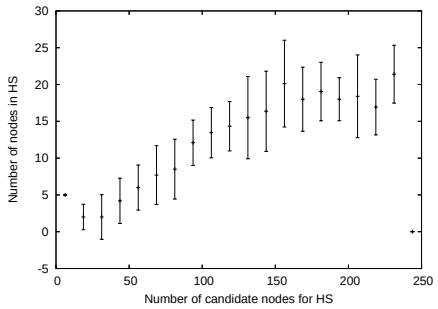

Fig. 3. Horizontal Sliver Scaling: Size of horizontal sliver at a node grows sub-linearly with total number of nodes within ϵ availability of the node).

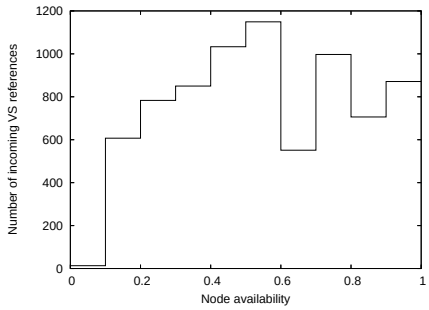

Fig. 4. Vertical Sliver Link Distribution: Number of incoming vertical sliver links to an availability range is uniform ([0,0.1] skewed as it has one node).

system was allowed to warm up for 24 hours, and a snapshot was taken of online nodes. Figure 2(a) shows that the availability distribution of online nodes in this snapshot is highly skewed, making this trace set a good test for our algorithms.

Figures 2(b,c) respectively show the distributions of horizontal sliver size and vertical sliver size at all these online nodes. From Figure 2(c), it is clear that the median values of the vertical sliver sizes are uncorrelated to the availability, as expected. Figure 2(b) shows an increasing median value of the horizontal sliver size with node availability. Yet, Figure 3 demonstrates that this increase is only sublinear - the horizontal sliver size grows sublinearly with the total number of nodes present within $\pm\epsilon$ availability. Finally, Figure 4 counts the total number of *incoming* vertical sliver links to nodes in different availability ranges. We observe that this number is largely uncorrelated to the distribution of nodes (seen in Figure 2(a)). Thus, we conclude that the AVMEM slice sizes follow theoretical analysis, even under realistic churn models.

Attack Analysis: We first evaluate the effect of a *flooding attack*, where a selfish (or malicious) node wishes to send out a message to all nodes that are not part of its AVMEM neighbor list(s). Although each node checks each incoming message to verify if its sender is a valid in-neighbor (according to the AVMEM predicate), and reject it if not, this is open to attacks due to several reasons: (1) nodes may use cached and stale availability information to do this check, and (2)

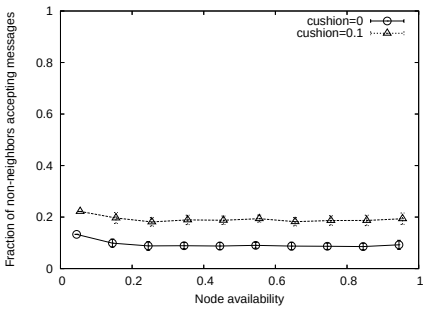

Fig. 5. Flooding Attack: Fraction of peers that are not currently neighbors that would accept communications. Measurement averaged across 0.1-wide availability ranges.

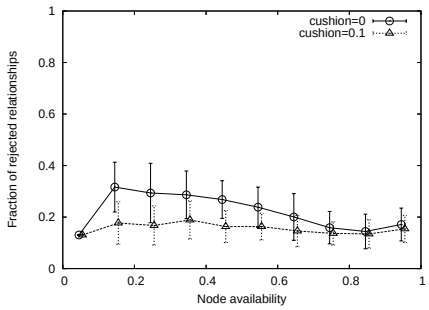

Fig. 6. Legitimate Rejection Rate: Fraction of nodes that will reject communications from an AVMEM in-neighbor. Measurement averaged across 0.1-wide availability ranges.

availability information reported by our underlying AVMON service could give inconsistent or inaccurate answers. Figure 5 (line for cushion=0) depicts that regardless of the availability of the selfish node, fewer than 10% of nodes outside of its AVMEM neighbor list accept its flooding message. This is reasonable - it means that to receive an audience from one additional peer, a selfish node must obtain information about 10 additional peers.

Second, we evaluate how the above inaccuracies and cached information affect the rejection of valid messages sent to AVMEM neighbors. Figure 6 shows that this number is below 30% regardless of the sending node's availability. To reduce this effect further, we add a constant *cushion* to the right hand side of equation (1) in Section 2, i.e., to function f. This reduces the rejection rate to below 20% (see also Figure 5). This is reasonable - it means that a node attempting to forward a message will have to try only an expected $\frac{1}{0.8} = 1.25$ neighbors before succeeding. From these two attacks, we conclude that AVMEM guarantees uniform attack resilience and acceptance rate for legitimate messages, *independent of the sending node's availability.*

4.2 Management Operations over AVMEM

In order to explore anycasts and multicasts systematically, we select the initiator node in one of three ways, and the target range in one of three ways, thus effectively giving us nine combinations for each management operation. Although we evaluated all the nine combinations, for brevity, we show data for only the most interesting ones below. Specifically, the initiator is chosen as either (1) `LOW` $\in [0, 0.3333)$, or (2) `MID` $\in [0.3333, 0.6666)$, or (3) `HIGH` $\in [0.6666, 1.0)$. For threshold operations (anycasts or multicasts), the target availability range was either 0.25, or 0.49 or 0.90. For range operations, the target availability range was either one of $[0.2, 0.3]$, or $[0.44, 0.54]$, $[0.85, 0.95]$. Each point on any plot is the average of 5 different protocol runs, each with 50 messages.

Basic Anycast Operations: We first evaluate anycast based on greedy forwarding using VS-only, HS-only, and HS+VS, as well as simulated annealing with HS+VS (see Section 3.2). The retried-greedy variation will be discussed soon. All anycasts are sent with $TTL = 6$. Among the nine options discussed above, the following four settings were the most interesting. First, Figure 7 shows the results for a range-anycast experiment with initiator in the `MID` and target $[0.85, 0.95]$. All variants gave a 100% success rate for messages, with all except HS-only finishing w.h.p. within 1 hop. This makes intuitive sense as messages will not travel far in availability space by using HS-only.

Second, Figure 8 shows the number of delivered range anycasts out of 50 sent, from nodes in availability range `HIGH` to three different target availability ranges: $[0.85, 0.95]$, $[0.44, 0.54]$, and $[0.2, 0.3]$. The third of these is the most harsh scenario, since it is very likely that either (1) there are no nodes online in the low availability ranges, or (2) the anycast takes a longer path via low-availability nodes, and thus has a high probability of being dropped inside the overlay, as its TTL expires. Of the multiple options, HS+VS comes out the best.

Retried-Greedy Anycast: Figure 9 shows the reliability and latency of retried-greedy forwarding, for different values of $retry$, under the harshest possible scenario of the initiator in `HIGH` and target range $[0.2, 0.3]$. The latency on each virtual hop here was selected uniformly at random from the interval $[20ms, 80ms]$. Notice that even under such harsh scenarios, $retry = 8$ gives as good a performance as the plateau - 60% delivery with an average latency of 739ms.

Benefit of AVMEM Predicate: In order to compare the usefulness of the horizontal (logarithmic-constant) and vertical (logarithmic) sub-predicates used in the above AVMEM overlay, we ran exactly the same range-anycast operation from Figure 9, but *over a random overlay graph* similar to those created by alternative membership protocols like SCAMP [8], CYCLON [25], T-MAN [25], etc. Figure 10 shows the data for this, and should be compared against Figure 9. A look at these figures tells us that for range-anycasts: (1) for management operations, overlays based on AVMEM predicates give a higher success rate than random graphs, while (2) both achieve similar latencies.

Multicast Operations: Figure 11 shows the latency performance of range- and threshold-multicast, using both flooding (default) and lower-cost gossip ($fanout = 5$, $N_g = 2$, gossip period=1 s). The latency for each multicast is the worst case, i.e., it is the time of the *last receiving* node obtaining the multicast. The CDF shows that this stays below 300ms for flooding, and 5.5s for gossiping. Figure 12 shows that the spam factor for multicasts is low, i.e., the fraction of multicasts overflowing the target range, and reaching a node outside is below 8% for most cases, except the topmost case where data is skewed by the small number of nodes in the target range. Finally, Figure 13 shows that flooding gets a reliability above 90%, while gossip reaches 70%. Bandwidth savings due to gossip may thus be worthwhile to applications less concerned about reliability.

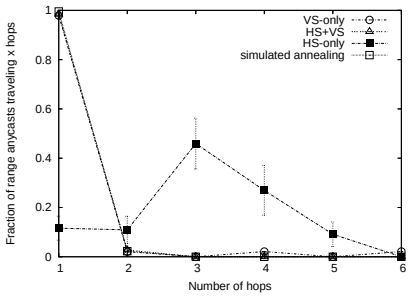

Fig. 7. Range anycast: Hops required to delivery when sending from `MID` to range $[0.85, 0.95]$.

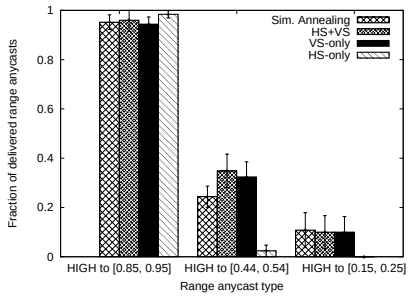

Fig. 8. Range anycast under increasingly harsh scenarios: Lower target availability ranges have lower success rate.

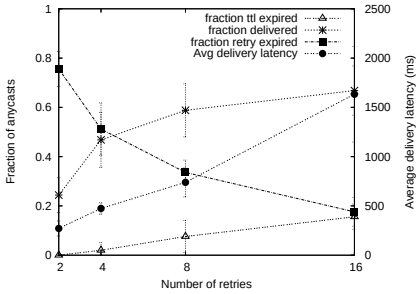

Fig. 9. Retried Greedy Anycast in harsh environment: Anycasts sent to target availability range $[0.15, 0.25]$ from nodes in `HIGH`.

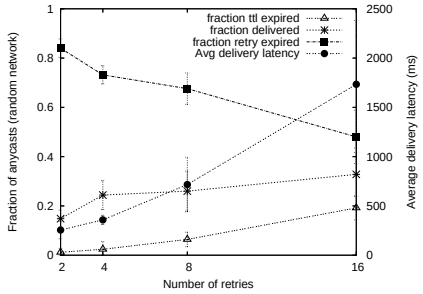

Fig. 10. Retried Greedy Anycast (Random Overlay, instead of AVMEM): Anycasts sent to target availability range $[0.15, 0.25]$ from nodes in `HIGH`.

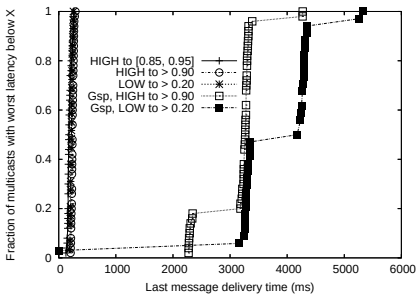

Fig. 11. Multicast Latency CDF: Latency of last message delivered for each multicast.

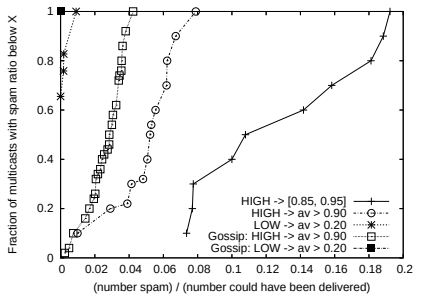

Fig. 12. Multicast Spam Ration CDF: Ratio of number of multicasts received by a node outside target range, to number of valid nodes in range.

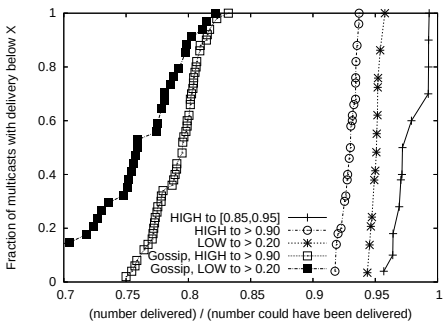

Fig. 13. Multicast Reliability CDF: Fraction of nodes inside target range, that received a multicast.

5 Conclusions

We have presented the design and evaluation of AVMEM, an availability-aware overlay. We showed that AVMEM overlay construction is scalable and that a set of availability-based management operations can be run efficiently and reliably on this overlay.

Our experimental evaluation using realistic overlay traces shows that the theoretical properties hold. Selfish nodes are implicitly kept under control and good overlay connectivity is achieved by the proposed AVMEM predicates. This allows anycasting and multicasting to availability ranges to be performed reliably.

References

1. E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), 2000.
2. J. Aspnes and G. Shah. Skip graphs. In *Proc. ACM-SIAM SODA*, pages 384–393, 2003.
3. R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proc. IPTPS*, pages 135–140, Feb. 2003.
4. R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker. Total Recall: System support for automated availability management. In *Proc. Usenix NSDI*, 2004.
5. F. Cappello and et al. Grid'5000: A large scale, reconfigurable, controlable and monitorable Grid platform. In *Proc. GRID*, 2005.
6. B.-G. Chun and et. al. Efficient replica maintenance for distributed storage systems. In *Proc. Usenix NSDI*, pages 45–58, 2006.
7. F. K. et al. Report of the NSF Workshop on Research Challenges in Distributed Computer Systems. http://www.nsf.gov/cise/cns/geni/workshop_report.pdf.
8. A. Ganesh, A.-M. Kermarrec, and L. Massoulie. Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2):139–149, Feb. 2003.

9. N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proc. USITS*, 2003.
10. IBM. The Oceano Project. http://www.research.ibm.com/oceanoproject/.
11. M. Jelasity and O. Babaoglu. T-Man: Gossip-based overlay toplogy management. *Self-Organising Systems: ESOA*, LNCS 3910:1–15, July 2005.
12. H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *Proc. Usenix OSDI*, 2006.
13. J. Liang, R. Kumar, and K. W. Ross. The fasttrack overlay: A measurement study. *Computer Networks*, 50(6):842–858, Apr. 2006.
14. V. Lo, D. Zhou, Y. Liu, C. Gauthier-Dickey, and J. Li. Scalable supernode selection in peer-to-peer overlay networks. In *Proc. IEEE Hot-P2P*, pages 18–27, 2005.
15. P. Maniatis, M. Roussopoulos, T. J. Giuli, D. S. H. Rosenthal, M. Baker, and Y. Muliadi. Preserving peer replicas by rate-limited sampled voting. In *Proc. ACM Symposium on Operating Systems Principles (SOSP)*, pages 44–59, 2003.
16. S.-H. Min, J. Holliday, and D.-S. Cho. Optimal super-peer selection for large-scale p2p system. In *Proc. ICHIT*, pages 588–593, 2006.
17. R. Morales and I. Gupta. AVMON: Optimal and scalable discovery of consistent availability monitoring overlays for distributed systems. In *Proc. ICDCS*, 2007 (to appear).
18. D. Patterson. A conversation with Jim Gray. *ACM Queue*, 1(4), Jun. 2003.
19. L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. In *Proceedings of the First ACM Workshop on Hot Topics in Networking (HotNets)*, October 2002.
20. T. Pongthawornkamol and I. Gupta. AVCast : New approaches for implementing availability-dependent reliability for multicast receivers. In *Proc. IEEE SRDS*, 2006.
21. A. Rowstron and P. Druschel. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware*, 2001.
22. Y. Shu, B. Ooi, K.-L. Tan, and A. Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Proc. P2P*, pages 173–180, 2005.
23. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM*, pages 149–160, 2001.
24. TechWise Research Inc. Are some RISC-based clusters easier to manage than others? http://h71000.www7.hp.com/openvms/whitepapers/sm_whitepaper.pdf, 2004.
25. S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.
26. S. Voulgaris, E. Riviere, A.-M. Kermarrec, and M. van Steen. Sub2Sub: self-organizing conten-based publish-subscribe for dynamic large scale collaborative networks. In *Proc. IPTPS*, 2003.
27. T. Weiss. Grid computing gets push from Sun, IBM and Compaq. *Computer World*, Nov. 2001.
28. C. Zheng, G. Shen, S. Li, and S. Shenker. Distributed Segment Tree: Support of range query and cover query over DHT. In *Proc. IPTPS*, 2006.