AVCast : New Approaches For Implementing Availability-Dependent Reliability for Multicast Receivers[*]

Thadpong Pongthawornkamol and Indranil Gupta

Department of Computer Science

University of Illinois at Urbana-Champaign

Urbana, IL 61801

{tpongth2,indy}@cs.uiuc.edu

Abstract

Today's large-scale distributed systems consist of a collection of nodes that have highly variable availability — a phenomenon sometimes called churn. *This availability variation is often a hindrance to achieving reliability and performance for distributed applications such as multicast. This paper looks into utilizing and leveraging availability information in order to provide availability-dependent message reliability for multicast receivers. An application (e.g., a publish-subscribe system) may want to scale the multicast message reliability on each receiver according to its availability — different options are that the reliability is independent of the availability, or proportional to it, or is some other arbitrary function of it. We propose several gossip-based algorithms to support several such predicates. These techniques rely on each node's availability being monitored in a distributed manner by a small group of other nodes in such a way that the monitoring load is evenly distributed in the system. Our techniques are light-weight, scalable, and are space- and time- efficient. We analyze our algorithms and evaluate*

[*]This work was supported in part by NSF CAREER grant CNS-0448246.

them experimentally by using availability traces collected from real peer-to-peer systems.

1 Introduction

Gossip-based protocols are useful information dissemination techniques for many large-scale distributed system applications such as publish-subscribe systems (e.g., RSS), multicast, peer-to-peer systems, and grid computing. They exhibit several desired properties such as simplicity, scalability, and fault-tolerance [1, 2]. Moreover, gossip-based protocols are also resistant to churn — this is the dynamism due to highly variable availability of different nodes in the system.

Several churn-resistant and scalable gossip-based multicast algorithms have been proposed in the past, e.g., [1–6]. However, to the best of our knowledge, none of these previous works provide support for availability-dependent reliability predicates, which is the capability to set the multicast reliability at multicast receivers based on the availability characteristics of the receivers. In the other words, we want to be able to specify and support a system-wide predicate that relates the reliability at each node to its availability.

There are several reasons why multicast reliability at each receiver should be related to the receiver's availability. The reasons are (1) *fairness* in reliability (high availability receivers get better reliability, but are not over-burdened), (2) *fighting freeloading* (hosts that have low availability and contribute nothing to the system, but get high multicast reliability), and (3) being an *incentive* for nodes to increase their own availability, which will result in a more reliable and resource-efficient system.

In order to address these issues, this paper presents *AVCast*, an availability-aware, gossip-based multicast protocol. AVCast currently allows a choice of two predicates that specify the availability-reliability relationship. Using AVCast, we study the effects on multicast reliability of high-variance availability distributions across hosts. Note that the terms *node*, *host*, and *receiver* have the same meaning and will be used interchangeably throughout this paper.

This paper has three main research contributions. 1) We propose a decentralized monitoring protocol for each member node to estimate the availability distribution of the system. 2) We create a generic framework to specify a range of availability-dependent reliability predicates. 3) We implement two reliability predicates : *uniform reliability*, and *availability-proportional reliability*.

Note that the conventional definition of multicast reliability is not appropriate for churned hosts that frequently switch their states between online and offline, since it is impossible for a node to receive a multicast message when that node is currently offline (this paper does not consider any repair mechanisms for each node to retrieve missing messages that were initiated during its offline period). Hence, we give a more appropriate definition of multicast reliability for a churned host as follows.

Definition – Multicast Reliability for a Churned Host: Consider a host x, and consider the number of multicast messages whose propagation time (i.e., from multicast initiation to multicast termination) completely overlaps with the available times for host x. Let f be the fraction of such messages received at host x. Then f is the multicast reliability for host x, denoted by r_x.

Notice that this definition of reliability is different and is more correct from the traditional definition, as it cleanly separates the unavailable times out of the reliability calculation.

The rest of this paper is organized as follows. Section 2 discusses some related works on churn-fairness studies and multicast protocols. Section 3 presents basic concepts and components of the AVCast system. Section 4 gives a theoretical analysis of availability-aware reliability framework in AVCast. Section 5 proposes two reliability predicates supported in AVCast. Section 6 shows the experimental results. Finally, Section 7 concludes the paper.

2 Related Work

2.1 Churn and Fairness Studies

There have been several works [7–9] addressing the characteristics of churn in large-scale distributed applications such as distributed file-sharing and multicast systems. The studies have shown that churn has effect on stability and performance of overlay networks and applications that are built on top of overlay networks. In order to solve such problems, [10, 11] have proposed techniques to build more churn-resistant overlay networks.

Besides the effect of churn on stability and performance problem, [12] also exposed the correlation between churn in global peer-to-peer applications and its effect to per-user fairness of quality of service. According to the study, most file-sharing systems consist of a significant portion of *free-riders*, the system users who exploit the benefit from the system without contributing to the system. Similarly, We quantitatively analyzes the effect of churn to stability and reliability of distributed systems, particularly in application-level, gossip-based multicast systems. Moreover, we presents a set of gossip-based multicast variation in order to ensure fairness in the term of multicast reliability.

2.2 Reliable Gossip-based Multicast

Recently, reliable gossip-based multicast has become an active area of research. Over the past few years, several gossip-based multicast protocols have been proposed [3–6, 13]. We examine some of them here.

Gocast [6] implemented a proximity-aware multicast protocol on top of Resilient Overlay Network(RON) [14] in order to achieve high throughput and low message delay. However, Gocast does not address effect of churn to reliability of the multicast system. AVCast, on the other hand, focus on relating system multicast reliability to availability of the system itself. Such two works are orthogonal and thus a combination of the two approaches is possible.

DOS-Resistant Unforgeable Multicast(Drum) [3] addresses reliability of multicast protocol under malicious denial of service attacks. AVCast tries to achieve best-effort multicast reliability on systems under non-malicious churn. We consider the system model where nodes are not malicious, but can act selfishly by having low-availability.

The work that is most similar from this paper is Araneola [13]. In Araneola, each node forms a hybrid of deterministic and random overlay network and forwards messages deterministically to one of its neighbors. Araneola is capable of achieving high reliability in the presence of low-rate churn. However, each node is required to maintain a specific set of overlay network. AVCast addresses churn in more dynamic environment and hence provides weaker guarantee on message delivery. The protocol proposed in AVCast can be applied to most overlay and membership services.

3 Basic Approach

AVCast consists of two components: 1) *monitoring and membership component* and 2) *availability-aware gossip-based multicast component*. In AVCast monitoring protocol, each AVCast node acts as a *pinging node* that monitors the availability of a few other nodes – in turn, each of these pinged nodes is called a *target node*. Each target node's availability is then monitored in a distributed fashion by a small group of pinging nodes. These pinging nodes are selected randomly but consistently for each target node, so that each node only monitors a small number of other target nodes. Note that the pinging and target node relationships are inverses of one another (a node x will be a pinging node of all x's target nodes). Once the availability information has been obtained by the membership component, each pinging node then uses the multicast component to forward multicast messages to a number of target nodes. How target nodes are selected to receive forwarded messages depends on the availability-dependent reliability predicate that is to be implemented.

This section details how the monitoring component and the multicast component operate. The mathematical analysis of AVCast's availability-dependent reliability predicates will then be presented in Section 4.

3.1 Monitoring and Membership Protocol

There are several simple methods to obtain the availability information of each node in the system (i.e., to select the pinging set for a given node). We discuss some of these approaches below, along with their disadvantages.

- Each node measures its availability by itself and reports its own availability to other nodes. While this method is simple and straightforward, it allows a selfish node to cheat, i.e., to lie about its availability.

- Each target node's availability is measured by pinging nodes that are basically some of its neighbors in an application-defined *overlay network*. This method eliminates the above problem of cheating, but how pinging nodes are determined is specific to the type of overlay network. In power-law overlays, for instance, the high degree nodes would share a large pinging responsibility, thus causing load imbalance.

- Each node's availability is measured by neighbor nodes specified by randomization techniques (e.g., via a random walk). However, random walks can also make biased choices, e.g., in a star network [15], thus causing load imbalance.

- To overcome limitations of approaches mentioned above AVCast uses a *consistent randomization* to select the pinging set. AVCast uses a low-overhead, decentralized, *hash function-based* protocol to determine pinging nodes for each target node. Using a globally consistent hash function, denoted by H, each node can verify its pinging nodes and target nodes in a consistent manner across the system, thus eliminating problems of selfish nodes cheating and adversarial peers controlling the system. Further, the uniformly random nature of the function H ensures better load balance than the previous approaches above.

In the AVCast membership protocol, a node x maintains links pointing to two sets of nodes. The first set is called the *pinging set* of x (PS_x), which contains x's pinging nodes whose duties are to monitor x's availability and to probabilistically forward multicast messages to x. The second set of nodes is the target set of x (TS_x), which contains all nodes whose availability x is monitoring. Notice that $x \in PS_y$ if and only if $y \in TS_x$. A node

x treats the availability distribution obtained from TS_x as a sample of system's global availability distribution. According to [16], having TS and PS of size $O(logN)$, where N is the approximate total number of offline and online nodes in the system, is enough to provide accuracy in availability estimation and achieve good scalability in multicasting. Note that N is consistently known or approximated beforehand by using size estimation algorithms such as [17] ([17] provides an estimated number of *online* nodes, denoted by n, which is different from N, However, we can calculate N from n and average system availability $E[a]$ with the equation $N = n/E[a]$). Another way to estimate N is by setting N to the power of 2 that is closest to the scale of the system. We do not envision this to be a hindrance since most peer-to-peer systems, in spite of churn, have stable system sizes [7].

The availability membership protocol is described in Figure 1 and Figure 2. Besides its own id number, each node x maintains two node-specific parameters used to determine sizes of pinging set and target set, K_{in}^{x} and K_{out}^{x}. A node x will add a node y to its target set TS_x (and y will add x to PS_y) if and only if

$$H(x, y) < \sqrt{K_{out}^{x}.K_{in}^{y}}/N \tag{1}$$

where H is a globally consistent hash function known to every node. H could be a SHA-1 or a MD5 hash function, but with the result normalized to the range $[0, 1]$. Initially, each node x sets its K_{in}^{x} and K_{out}^{x} values to a value of K (a known priori of all nodes set to a value that is $O(logN)$). Note that K is the expected size of PS and TS. However, if a node x finds its pinging set or target set smaller or larger than K, it can adjust its K_{in} and K_{out} parameters to balance the size of its TS and PS respectively. Such procedure is called *rebalancing operation*. The rebalancing operation will be discussed later in this section.

```
procedure join()
1: if $x$ joins for the first time then
2:     $y \leftarrow$ random node
3:     send $\langle$REQ$, x, K^x_{in}, K^x_{out}\rangle$ to $y$
4: else
5:     $PS_x \leftarrow$ persistent storage
6:     $TS_x \leftarrow$ persistent storage
7:     $K^x_{in} \leftarrow$ persistent storage
8:     $K^x_{out} \leftarrow$ persistent storage
9: end if
procedure update($y, K^y_{in}, K^y_{out}$)
1: if $H(x,y) < \sqrt{K^x_{out}.K^y_{in}}/N$ then
2:     add $y$ into $TS_x$
3: end if
4: if $H(y,x) < \sqrt{K^y_{out}.K^x_{in}}/N$ then
5:     add $y$ into $PS_x$
6: end if
```

Figure 1. Availability Membership Protocol

```
receive $\langle$REQ$, y, K^y_{in}, K^y_{out}\rangle$
1: multicast_send($\langle$ADV$, y, K^y_{in}, K^y_{out}\rangle$)
receive $\langle$ADV$, y, K^y_{in}, K^y_{out}\rangle$
1: update($y, K^y_{in}, K^y_{out}$)
2: if $y \in TS_x$ or $id' \in PS_x$ then
3:     send $\langle$REP$, x, K^x_{in}, K^x_{out}\rangle$ to $y$
4: end if
receive $\langle$REP$, y, K^y_{in}, K^y_{out}\rangle$
1: update($y, K^y_{in}, K^y_{out}$)
```

Figure 2. Membership Message Handlers

```
every period $T_1$
1: if $|PS_x| < (1-\alpha)K$ then
2:     $K^x_{in} \leftarrow (1+\beta)K^x_{in}$
3: end if
4: if $|PS_x| > (1+\alpha)K$ then
5:     $K^x_{in} \leftarrow (1-\beta)K^x_{in}$
6: end if
7: if $|TS_x| < (1-\alpha)K$ then
8:     $K^x_{out} \leftarrow (1+\beta)K^x_{out}$
9: end if
10: if $|TS_x| > (1+\alpha)K$ then
11:     $K^x_{out} \leftarrow (1-\beta)K^x_{out}$
12: end if
13: if $K^x_{in}$ or $K^x_{out}$ is changed then
14:     multicast_send($\langle$ADV$, x, K^x_{in}, K^x_{out}\rangle$)
15: end if
```

Figure 3. Rebalance Operation

```
every period $T_2$
1: for each node $y \in TS_x$ do
2:     send $\langle$PING$, x\rangle$ to $y$
3:     if receive $\langle$PONG$\rangle$ back from $y$ before timeout
       $T_0$ then
4:         mark $y$ as available
5:     else
6:         mark $y$ as unavialable
7:     end if
8:     recalculate $y$'s availability value $a_y$
9: end for
```

Figure 4. Membership Maintenance Procedure

```
procedure multicast_send($message$)
1: for each node $y \in TS_x$ do
2:     send $message$ to $n$
3: end for
receive multicast message from $y$
1: if $y \in PS_x$ then
2:     for $i = 1$ to $C$ do
3:         for each currently online node $z \in TS_x$) do
4:             with probability $= p(a_z)$, forward message to $z$
5:         end for
6:     end for
7: end if
```

Figure 5. Availability-aware No-wait Gossiping protocol at node x

Now, we focus on the action during the monitoring process. When a node x joins the system for the first time, it sends a REQ (request) message to an arbitrary node z in the system (that node z then becomes x's contact node). The REQ message contains x's node id, and K_{in}^{x}, and K_{out}^{x} values. The contact node z then uses an ADV (advertise) message to forward x's request to all other nodes via a multicast. Any node y that receives the ADV message then evaluates the equation (1). If the condition is true, it will add x into its pinging set PS_y and sends a REP (reply) message back to x. Similarly, node y evaluates (using the same equation (1)) whether node x should belong to TS_y. Upon receiving a REP message from y, x can verify the equation (1) and add y into its target set TS_x (or PS_x respectively). Note that the PS_x and TS_x lists are stored in x's persistent storage so that if x goes offline and joins the system again, it can retrieve the information without needing a contact node.

Rebalancing operation : In practice, the distribution of the hash space may not be uniform, resulting in the sizes of PS and TS at each node being different from K. To reduce the load variance across nodes, AVCast uses a rebalance procedure shown in Figure 3. This procedure adjusts K_{in} and K_{out} of individual nodes in order to keep the size of PS and TS as close to the expected size K as possible. The rebalance procedure defines two system-wide constants: α and β. α and β are preconfigured values ranging from 0 to 1. α can be considered as the level of tolerance of link degree invariance, while β defines how reactive the rebalance procedure is. A node x whose target set contains more than $(1+\alpha)K$ members decreases its K_{out}^{x} value by the scale of $(1-\beta)$. Similarly, if x's target set contains less than $(1-\alpha)K$ members, x increases its K_{out}^{x} value by the scale of $(1+\beta)$. Every time either K_{in}^{x} or K_{out}^{x} is recomputed, x re-advertises its new parameters. Each node x repeats the rebalancing procedure until $|TS_x|$ and $|PS_x|$ are within range $[(1-\alpha)K, (1+\alpha)K]$. The smaller α is, the less load invariance the system has and the more control message overhead incurred for the rebalancing procedure.

The monitoring component operates in asynchronous protocol rounds (typically 5 to 10 seconds long) without synchronization between nodes. During each round, at a node x, it periodically sends PING messages to all target nodes in TS_x and waits for reply messages from them. Any target nodes that fail to send back reply messages before the next round will be considered unavailable during that round. Each node then stores its target nodes' raw availability traces in its persistent storage. To prevent excessive overhead, each pinging node uses the availability trace from T most recent rounds to calculate target nodes' availability value, where T is a globally defined constant for the system. The availability value of a target node y, denoted by a_y, measured at one of its pinging nodes x is calculated as the fraction of T most recent rounds that y responded to x. The availability traces older than T rounds can be either discarded or aggregated into a coarser-scale archive, depending on the implementation of the system.

3.2 Availability-aware Gossip-based Multicast Protocol

AVCast adopts an existing gossip-based multicast protocol called *no-wait gossiping*. This is not a new protocol, but was proposed in [6]. Figure 5 shows the availability-aware version of no-wait gossiping protocol used in AVCast. In AVCast, the sender node x initiates a multicast by sending a multicast message to each *online* node in its target set TS_x. Upon receiving a multicast message, a node immediately forwards C copies of the message, each one to C selected *online* target nodes, where C is a globally defined constant. The way the online target nodes are picked up is not uniform — instead, a node y forwards the message to its online target node z with probability $p(a_z)$, where $p(.)$ is a global probability function of availability, and a_z denotes z's availability as observed by y. $p(.)$ is chosen depending on the global predicate that is to be satisfied (Section 5). The effect of choosing C and $p(.)$ will also be analyzed quantitatively in Section 4.

The no-wait protocol is completely stateless in the sense that each node forwards a message to some of its target nodes *only once* and immediately after it receives the message. We believe that this stateless property makes the protocol appropriate to use in dynamic systems where nodes frequently go offline and online.

4 Analysis of Gossip-based Multicast Protocol

In this section, we analyze several characteristics of AVCast availability-aware gossip-based multicast protocol introduced in Section 3.2. We show the effect of two global system parameters: the target selection probability function $p(.)$ and the number of copies each node forwards per message C. We also present the analysis of how different $p(.)$ and C affect the reliability predicate of the system in this section.

We model a multicast as a synchronous process operating in multiple protocol rounds (The term *round* here is different from one in Section 3.1). The first round starts when the multicast is initiated and the last round ends when the multicast dies out. For a given multicast message, at any time, each *online* node falls into one of three categories: *virgin* nodes which have not yet received the message, *active* nodes which have just received the message but have not yet forwarded the message, and *inactive* nodes which have already received and forwarded the message. In the analysis, we assume that any node that was online when a multicast was initiated will stay online during the entire period that the multicast is active. Since typical average multicast latencies are in the scale of a few hundred milliseconds to a few seconds while typical online durations of nodes observed in [7] are in the scale of several minutes, this assumption is reasonable. With this assumption, we can discard offline nodes and focus on only online nodes in the analysis.

Recall that, in each round, each active node gossips C copies of a message, one each to C selected online target nodes and then becomes inactive. Virgin nodes that receive the gossip message then become new active nodes in the next round. On the other hand, inactive nodes do nothing if they receive duplicates of the same message. The multicast process stops when there is no active node left in the system.

Consider a system of N nodes with an availability mass function f (i.e. the fraction of overall system nodes that have availability equal to a is $f(a)$). Hence, the average number of online nodes in the system n at any time is

$$n = \sum_{\{a:f(a)\neq 0\}} (af(a)N) = E[a]N,$$

where $E[a]$ is the mean availability of all nodes in the system. We also define the *online* availability mass function $g(a)$ as the fraction of online nodes that have availability a. We can calculate $g(a)$ from $f(a)$ by the following equation:

$$g(a) = \frac{af(a)}{E[a]} \tag{2}$$

Now, for a given multicast, let x_t, y_t, and z_t be the number of online nodes (with respect to n) that are active, inactive, and virgin in the system at round t respectively ($x_t + y_t + z_t = n$ at any round t). Also, let $g_t(a)$ be the fraction of virgin nodes that have availability equal to a at protocol round t. Initially, a sender node initiates a multicast by broadcasting the message to all K_{on} online nodes in its target set (K_{on} is the number of online target nodes, which is equal to $KE[a]$ on average). In each of subsequent rounds, each active node forwards a constant number of copies C to its online target nodes using target probability function $p(a)$. Hence the number of messages forwarded in each round t is equal to Cx_t. Thus, the protocol model at round $t+1$ can be described by the following set of equations.

$$\begin{aligned}
x_{t+1} &= \text{Expected number of virgin nodes that have received the message in round } t \\
&= \sum_{\{a:g_t(a)\neq 0\}} (ng_t(a)\text{P[node receives the message]}) \\
&= \sum_{\{a:g_t(a)\neq 0\}} \Big(ng_t(a)(1-(1-p(a))^{\frac{CK_{on}x_t}{n}})\Big) \\
&= \sum_{\{a:g_t(a)\neq 0\}} \Big(ng_t(a)(1-(1-p(a))^{\frac{CKE[a]x_t}{n}})\Big) \\
y_{t+1} &= y_t + x_t \\
z_{t+1} &= n - x_{t+1} - y_{t+1} \\
g_{t+1}(a) &= g_t(a)((1-p(a))^{\frac{CKE[a]x_t}{n}})
\end{aligned}$$

with the initial conditions as

$$x_0 = K_{on} = KE[a], y_0 = 0, z_0 = n - x_0$$

and

$$g_0(a) = g(a)(1-\frac{K_{on}}{n}) = g(a)(1-\frac{K}{N})$$

where $E[a] = \sum_{\{a:f(a)\neq 0\}}(af(a))$ = mean availability of the system

Note that $g_t(a)$, the fraction of virgin nodes with availability a, keeps changing in each round t. The intuition behind this is that virgin nodes with different availability will be picked up to receive the message with different probability.

Given C, the number of copies forwarded per message, the availability distribution function $f(a)$, the total number of nodes in the system N, and the average size of pinging set and target set K as inputs, we can use the model mentioned above to analyze several characteristics of the multicast as described below.

4.1 System-wide Multicast Reliability and Message Propagation Delay

The system-wide multicast reliability is defined as the fraction of online nodes that receive at least one copy of multicast message. Thus, the system-wide multicast reliability R is

$$R = \frac{y_v}{n} = \frac{y_v}{NE[a]}$$

, where $v =$ the minimum value of t such that $x_t = 0$.

Hence, given the average size of the sender's target set K (usually, $K = O(\log n)$), the system availability mass function $f(a)$, and the target probability function $p(a)$, we can calculate the appropriate number of copies C in order to achieve desired system-wide multicast reliability. Calculating equations above can be locally performed at each multicast node using the availability distribution from its target set TS. Note that each node x in the system can estimate the availability mass function f as $f(a_y) = \frac{1}{|TS_x|}$ for each node $y \in TS_x$.

The message propagation delay is the number of protocol rounds in which the system contains a least one active node. Hence, the message propagation delay d can be defined as

$$d = v$$

where $v =$ the smallest t such that $x_t = 0$

4.2 Node-level Multicast Reliability

The node-level multicast reliability is defined as the probability that a node eventually receives at least one copy of multicast message from its pinging set (from the definition in Section 1, this probability is given that the node is available throughout the multicast period. According to the equations, the multicast reliability $r(a)$ at a node whose availability equal a can be estimated as follows.

$$\begin{aligned}
r(a) &= P[\text{node receives at least one copy}] \\
&= 1 - P[\text{node receives no copies}] \\
&= 1 - \prod_{t=1}^{\infty}\left[(1-p(a))^{\frac{CKE[a]x_t}{n}}\right] \\
&= 1 - (1-p(a))^{\frac{CKE[a]y_v}{n}} \\
&= 1 - (1-p(a))^{CKE[a]R} \qquad (3)
\end{aligned}$$

where

$v =$ the smallest t such that $x_t = 0$, and $R =$ system-wide multicast reliability

The analysis shows that the system-wide multicast reliability has an effect on node-level multicast reliability, no matter what node-level availability a or gossip target probability $p(a)$ are chosen. Also, the relation between expected global system-wide reliability R and local per-node reliability r can be described as

$$R = \frac{E[ra]}{E[a]}$$

where

$$E[ra] = \sum_{\{a:f(a)\neq 0\}} (af(a)r(a))$$

5 Availability-dependent Reliability Predicates

Currently, AVCast supports two target probability functions, leading to two availability-reliability predicates. The first predicate is *uniform reliability,* which all nodes receive roughly the same level of multicast reliability. The second predicate is *availability-proportional reliability,* where the reliability each node receives is equal to the availability value of the node itself.

5.1 Uniform Per-Node Reliability ($r =$Constant)

According to the model presented in the previous section, specifying target selection probability function $p(a)$ = $\frac{1}{K_{on}}$, where K_{on} is the number of available nodes in the target set, will result in a naive uniform gossip-based multicast scheme where every available node in the pinging set is equally likely to be picked up as a message receiver.

The node-level multicast reliability $r(a)$ of a node with availability a in uniform gossip-based multicast can be expressed as the following equation.

$$\begin{aligned} r(a) &= 1-(1-p(a))^{CKRE[a]} \\ &= 1-(1-\frac{1}{K_{on}})^{CK_{on}R} \\ &\geq 1-e^{-RC} \end{aligned}$$

It can be seen that per-node reliability $r(a)$ value does not depend on the per-node availability a value. Thus, the quality of service each node obtains is equal to system-wide reliability. Moreover, the system-wide reliability can be expressed as $R = \frac{E[ra]}{E[a]} = \frac{E[r]E[a]}{E[a]} \geq 1-e^{-RC}$ for this predicate.

5.2 Availability-Proportional Reliability ($r = a$)

Although defining the target selection policy function as a constant results in equality of node-level multicast reliability at each node, such a policy does not provide the fairness property because high-availability nodes achieve the same level of multicast reliability as low-availability nodes. Since the fairness property is an important incentive for users in many large-scale peer-to-peer applications, one might want to construct a multicast infrastructure where multicast reliability at each node is *linearly proportional* to the availability of the node itself.

According to node-level reliability analysis (i.e., equation (3)),

$$r(a) = 1 - (1 - p(a))^{CKRE[a]}$$

However, we want $r(a)$ to be equal to a to satisfy the predicate. Replacing $r(a)$ with a in the above equation, the new equation is

$$a = 1 - (1 - p(a))^{CKRE[a]}$$

Also, since $r(a) = a$, global system-wide reliability R can be expressed as

$$R = \frac{E[ra]}{E[a]} = \frac{E[a^2]}{E[a]}$$

By replacing R and rearranging the equation, the target probability function can be expressed as a function of availability as follows.

$$p(a) = 1 - (1 - a)^{\frac{1}{CKE[a^2]}}$$

Notice that $E[a^2]$ at node x can be calculated based on TS_x's availabilities. With the formula above, each node x can adjust its C value locally so that $\sum_{z \in TS_x} (p(a_z)) \geq 1.0$. The rest of the protocol is the same as the main protocol framework described in Section 3.2.

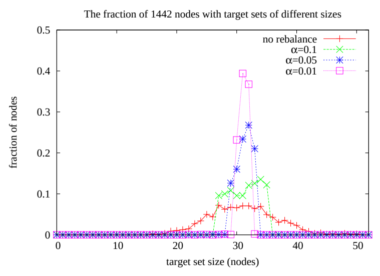
6 Experimental Results

We have evaluated the AVCast protocol via simulation. Our implementation of AVCast contains almost 3,000 lines of C++ code including the membership and the gossiping protocols. The availability distribution of nodes in the experiment is obtained from Overnet file-sharing system trace [7], which has average availability roughly equal to 0.3. In the simulation, the system consisting of 1442 nodes runs the multicast protocol for 6,000 protocol rounds (each round lasts around 5 seconds in practical). For simplicity, round are synchronized throughout the system in the simulation. At the beginning of each round, each node randomly tosses a number between 0 and 1 to decide whether it is online or offline throughout that round (each node stays online if the number is less than its availability value). During the first 3,000 rounds, each peer runs the availability monitoring and view rebalance operations. During the last 3,000 rounds, a randomly selected online node initiates one multicast message to the system per round. Each multicast message's propagation is assumed to die out within a single round because nodes in the no-wait gossiping scheme forward a message all at once, resulting in a very quick multicast process. Hence, we can also assume that a node is either fully offline or fully online for a given message. Setting round duration to 5 seconds is reasonable since a multicast typically completes within 5 seconds while the monitoring process can achieves high accuracy. The average system reliability is the average fraction of online nodes that receives message in each of 3,000 rounds. The average node reliability of each node is measured by the number of rounds the node receives messages, divided by the number of rounds the node is online.

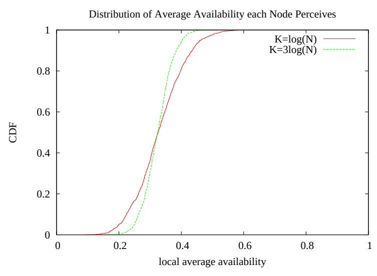
We first discuss the effectiveness of the membership protocol under different α (balance sensitivity) and K (expected size of TS and PS parameters). The effectiveness will be measured in terms of (1) how well the load is balanced throughout all nodes, (2) how accurately the availability distribution each node perceives from its target set, and (3) the number of control message overhead incurred from the rebalancing operations. Then we evaluate the availability-aware gossiping protocol in both predicates: uniform node reliability and availability-proportional node reliability. Our evaluation is also based on how well the node reliability distribution implements the predicates. We test each predicates with different number of copies C and the average target set size K parameters. Unless explicitly stated, each experiment is done with the following default parameter values: rebalance sensitivity $\alpha = 0.05$, balance aggressiveness $\beta = 0.1$, total number of nodes $N = 1,442$.

6.1 Availability Membership

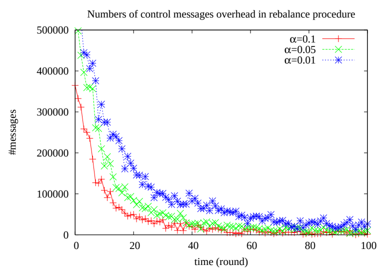
At time $t = 0$ in the simulation, all of 1,442 nodes were brought up online and thus each node knows all of its target set. During each t between $t = 0$ and $t = 3,000$, each node runs the view rebalance operation in order to keep its target size to $3logN = 3log(1,442) = 31$. In the rebalance operation, we vary α, the sensitivity factor, from 0.1 to 0.01 and fix β, the aggressiveness factor, at 0.1. At $t = 3,000$, we observe the size of target set at each node and the availability distribution each node has recorded from its target set.



(a) The size each node's target set



(b) The distribution of average local availability



(c) Control message overhead during rebalance period

Figure 6. The monitoring component

6.1.1 Local view balance

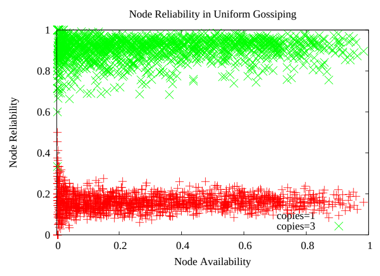
Figure 6(a) shows the distribution of sizes of the target set at each node at $t = 3,000$. Note that the results are also similar to the distribution of sizes of the pinging set. The smaller α is, the more consistent the sizes of the target set are. As the result also applies to the pinging set of each node, smaller α leads to a more balanced load across all nodes. However, setting α too small may cause oscillations and frequent view changes.

6.1.2 Accuracy

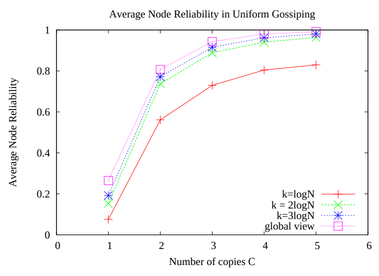
We use the average system availability each node observes from its target set as a measurement of how accurately each node perceives the availability condition of the system. Figure 6(b) shows the result of the system with different target set sizes. Notice that as the bigger the view size is, the more accurate system availability each node perceives. However, it seems not to have too much of a difference between $K = logN$ and $K = 3logN$.

6.1.3 Control Message Overhead

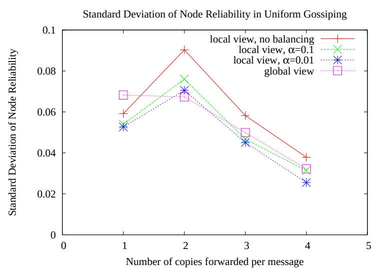
Figure 6(c) shows the control message overhead used in the rebalancing protocol during the first 100 rounds of the rebalancing operation. Message overhead was high at first since all nodes were adjusting its K_{out} and K_{in} values. After few rounds, traffic load dropped drastically since most nodes were satisfied with their settings. According to the result, the higher α is, the higher control message overhead incurred in the system. This is because more nodes will need to rebalance their views due to the more strict constraint. .



(a) Node-level reliability distribution for $K = 2logn$ and $\alpha = 0.05$

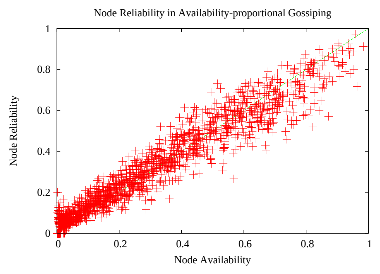


(b) Average node-level reliability

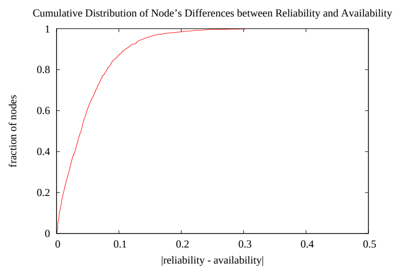


(c) Standard deviation of Node-level reliability with different α

Figure 7. Uniform reliability predicate under churn (Overnet trace)



(a) Node-level reliability distribution



(b) CDF of nodes by differences between availability and reliability

Figure 8. Availability-proportional reliability predicate under churn (Overnet trace)

6.2 Availability-aware Gossiping

This section presents the result of the availability-aware gossiping between $t = 3,000$ and $t = 6,000$ in the simulation. The experiment is done with different view sizes K and different α parameters. The results are compared to the system where each node has global membership knowledge.

6.2.1 Uniform Reliability

The results of the uniform gossip-based multicast simulation are shown in Figure 7. The overall conclusions are as follows. It can be seen that each node merely obtains the same node-level reliability, regardless of its availability value. As the number of copies forwarded per message increases, the multicast reliability also increases. Also, the equations derived in Section 4 predict the system-wide reliability accurately when the average target size is more than $2logN$. Figure 7(a) displays the availability-reliability scatter plot where each point represents each node. There are three sets of plots in the graph, representing three experiments with three different C, the number of copies forwarded per message. All three experiments used the average target size $= 2logN$ and $\alpha = 0.05$.

Figure 7(b) demonstrates the average node reliability of system with different view sizes K and different C parameters without the rebalance procedure (simulations with rebalance procedure yield the similar result as ones without rebalance procedure). The figure shows the effectiveness of system in the sense that setting target view size more than or equal to $2logN$ suffices to have the same performance as setting each node to have the global membership knowledge. The performance difference between systems with different view sizes converges when C is increased. The figure concludes that for the system sizes considered, setting $K = 3logN$ and $C = 4$ results in good performance while incurring reasonable space and network overhead.

Another perspective to evaluate the constant reliability predicate is the consistency in the quality of service each node receives from the system. Figure 7(c) shows the standard deviation of node reliability in systems for $k = 3logN$ and different α values. The result is consistent with the result from figure 7(a) that the standard deviation increases as C increases from 1 to 2, but the standard deviation decreases as C increases beyond 2. Also, the smaller α is, the smaller the standard deviation in the system. Generally, the standard deviations of node reliability in local-view systems are comparable to the one in the global-view system.

In conclusion, the simulation shows that the system's behavior in uniform gossiping follows the constant reliability predicate very well. In addition, setting $K = 3logN$, $C = 3$, and $\alpha = 0.1$ is an appropriate configuration.

6.2.2 Availability-proportional Reliability

This section presents the simulation result of the availability-proportional gossiping protocol. The experiment is done with view size $K = 3logN$ and $\alpha = 0.05$. Figure 8(a) shows the scatter plot between the availability and the reliability at each node. As shown in the figure, most nodes have multicast reliability at roughly the same level as its availability, which is consistent with the availability-proportional predicate.

Figure 8(b) shows the cumulative distribution of nodes whose reliability differs from their availability in different scales. As seen from the picture, around 60% of nodes in the system obtain multicast reliability that differs from their availability within a value of 0.05 or less. Around 80% of nodes in the system obtain multicast reliability that differs from their availability in the scale of 0.1. Only 2% of all nodes obtain the multicast reliability that differs from their availability more than 0.2.

In conclusion, the availability-proportional gossiping protocol performs well in the sense that each node receives the service with quality proportional to its contribution to the system. Only a few nodes receive a service with quality significantly different from their behavior.

7 Conclusions

This paper presented AVCast, an availability-aware membership management and multicast framework. AVCast provides an availability-monitoring service and an availability-aware gossip-based multicast service for each node in the system in a decentralized manner. The paper also presented a generic framework that allows an application to adjust AVCast parameters in order to implement a multicast system with the desired availability-dependent reliability predicate. Finally, the paper analyzed two reliability predicates that lead to system fairness — uniform per-node reliability and availability-proportional per-node reliability. The experimental results validated the correctness and applicability of the proposed schemes.

References

[1] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal Multicast," *ACM Transactions On Computer Systems*, vol. 17, no. 2, pp. 41–88, May 1999.

[2] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking, "Randomized Rumor Spreading," in *Proc. FOCS '00*. Washington, DC, USA: IEEE Computer Society, 2000, p. 565.

[3] G. Badishi, I. Keidar, and A. Sasson, "Exposing and Eliminating Vulnerabilities to Denial of Service Attacks in Secure Gossip-Based Multicast," in *Proc. DSN '04*. IEEE Computer Society, 2004, pp. 223–232.

[4] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, "Lightweight Probabilistic Broadcast," in *Proc. DSN '01*. IEEE Computer Society, 2001, pp. 443–452.

[5] L. Rodrigues, S. B. Handurukande, J. O. Pereira, R. Guerraoui, and A.-M. Kermarrec, "Adaptive Gossip-Based Broadcast," in *Proc. DSN '03*. IEEE Computer Society, 2003, pp. 47–56.

[6] C. Tang and C. Ward, "GoCast: Gossip-Enhanced Overlay Multicast for Fast and Dependable Group Communication," in *Proc. DSN '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 140–149.

[7] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding Availability," in *Proc. IPTPS '03*, vol. 2735. Springer, 2003, pp. 256–267.

[8] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Proc. MMCN '02*, San Jose, CA, USA, January 2002. [Online]. Available: citeseer.ist.psu.edu/article/saroiu02measurement.html

[9] K. C. Almeroth and M. H. Ammar, "Collecting and Modeling the Join/Leave Behavior of Multicast Group Members in the MBone," in *Proc. HPDC '96*. Washington, DC, USA: IEEE Computer Society, 1996, p. 209.

[10] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for Low Latency and High Throughput," in *Proc. NSDI '04*. USENIX, 2004, pp. 85–98.

[11] S. C. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *Proc. USENIX Annual Technical Conference, General Track*. USENIX, 2004, pp. 127–140.

[12] E. Adar and B. A. Huberman, "Free Riding on Gnutella," *First Monday*, vol. 5, no. 10, 2000.

[13] R. Melamed and I. Keidar, "Araneola: A Scalable Reliable Multicast System for Dynamic Environments," in *Proc. NCA '04*. IEEE Computer Society, 2004, pp. 5–14.

[14] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *Proc. SOSP '01*. New York, NY, USA: ACM Press, 2001, pp. 131–145.

[15] A. Singh, T.-W. Ngan, P. Druschel, and D. S. Wallach, "Eclipse Attacks on Overlay Networks: Threats and Defenses," in *Proc. InfoCom '06*, Barcelona, April 2006.

[16] A. J. Ganesh, A.-M. Kermarrec, and L. Massouli, "SCAMP: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication," in *Proc. NGC '01*. London, UK: Springer-Verlag, 2001, pp. 44–55.

[17] D. Kostoulas, D. Psaltoulis, I. Gupta, K. Birman, and A. Demers, "Decentralized Schemes for Size Estimation in Large and Dynamic Groups," in *Proc. NCA '05*. IEEE Computer Society, 2005, pp. 41–48.