Latency and Bandwidth-Minimizing Failure Detectors

Kelvin C. W. So   Emin Gün Sirer
Dept. of Computer Science
Cornell University
Ithaca, NY 14853
{kelvinso, egs}@cs.cornell.edu

ABSTRACT

Failure detectors are fundamental building blocks in distributed systems. *Multi-node failure detectors*, where the detector is tasked with monitoring N other nodes, play a critical role in overlay networks and peer-to-peer systems. In such networks, failures need to be detected quickly and with low overhead. Achieving these properties simultaneously poses a difficult tradeoff between detection latency and resource consumption.

In this paper, we examine this central tradeoff, formalize it as an optimization problem and analytically derive the optimal closed form formulas for multi-node failure detectors. We provide two variants of the optimal solution for optimality metrics appropriate for two different deployment scenarios. $\sqrt{s}$*-LM* is a *latency-minimizing* optimal failure detector that achieves the lowest average failure detection latency given a fixed bandwidth constraint for system maintenance. $\sqrt{s}$*-BM* is a *bandwidth-minimizing* optimal failure detector that meets a desired detection latency target with the least amount of bandwidth consumed. We evaluate our optimal results with node lifetimes chosen from bimodal and Pareto distributions, as well as real-world trace data from PlanetLab hosts, web sites and Microsoft PCs. Compared to standard failure detectors in wide use, $\sqrt{s}$ failure detectors reduce failure detection latencies by 40% on average for the same bandwidth consumption, or conversely, reduce the amount of bandwidth consumed by 30% for the same failure detection latency.

Categories and Subject Descriptors

C.2.3 [**Network Operations**]: Network monitoring

General Terms

Algorithms, Design, Performance

Keywords

Failure detection, overlays, wide-area networks

1. INTRODUCTION

Detecting failed nodes is an essential task in many distributed systems. In particular, *multi-node failure detectors* are ubiquitous in many settings and form a fundamental part of higher-level failure recovery operations in many systems. Specifically, a multi-node failure detector is tasked with monitoring a set of nodes (a *fail-detect set*) and needs to detect failures within the set with low latency while consuming little network bandwidth. Multi-node failure detectors are fundamental components in overlays, content distribution networks, and group communication services, as well as many other distributed systems. For instance, peers in a distributed hash table need to monitor nodes in their routing tables to detect when they should replace or take over the duties of failed neighbors. A content distribution network needs to monitor the status of servers in order to direct clients to the most suitable live replica. Group communication systems [3, 17, 27, 23] use failure detectors to determine group membership. Much past work on distributed failure detectors [9, 11, 25, 1, 24, 2, 6, 36, 15, 31, 21, 41, 14] assumes and builds on top of such a failure detector module at each node.

The challenge in failure detection stems from the conflicting goals of low detection latency and low bandwidth consumption. In the limit, low latency can be achieved by using all available bandwidth to constantly poll the nodes in the fail-detect set. Similarly, it is possible to build a very low bandwidth (and infinite latency) failure detector by not probing any nodes at all, though such a detector is useless in practice. Between these two extremes, there exists an optimal point where a node checks members of its fail-detect set at just the right frequency such that the total bandwidth consumption is limited to a targeted value and failure detection latency achieves a desired level of performance. This paper presents a technique for determining this optimal point.

The standard practice in distributed systems is to use ad hoc measures to pick a single period τ, and to probe every node in the fail-detect set every τ seconds [33, 32, 18, 20]. The selection of a single global τ would make sense in a setting where all nodes are homogeneous and exhibit the same session duration. However, past studies have demonstrated that node lifetimes are highly skewed [34], which in turn renders a single probe frequency far from optimal. It is possible to posit many heuristics for picking a node specific τ_i, but heuristics are unlikely to achieve a good tradeoff between bandwidth and latency, and the conditions under which they might work well are hard to characterize. In this paper, we derive the optimal strategy for resource-optimal failure detection from first principles.

Overall, this paper makes three contributions. First, it formalizes the multi-node failure detection problem, formulates it in a manner amenable to mathematical optimization and identifies two variants of the optimal solution encountered in practice. The *latency-minimizing* variant minimizes failure detection latency while constraining the amount of bandwidth used for probing nodes. The *bandwidth-minimizing* variant achieves a targeted failure detection latency while minimizing the amount of bandwidth consumed. The former is applicable when the amount of bandwidth dedicated to system overhead is known a priori as it can achieve the optimal latency within a strict resource limit, while the latter is best suited for settings where a particular performance level is sought from the failure detector module as it can achieve the performance goals using the minimal resources required. Second, this paper analytically derives closed-form formulas for the optimal rate at which failure checks need to be performed to solve the latency- and bandwidth-minimizing failure detection problems. The closed form formulas depend solely on parameters readily available in the problem statement or measurable via simple mechanisms, such as the distribution of node lifetimes, loss rate, targeted level of accuracy, the bandwidth constraint and the latency target. Finally, we evaluate the proposed solutions using three realistic traces from PlanetLab, web servers and Microsoft Desktop PCs, compare them to standard failure detectors commonly found in distributed systems, and show that they significantly improve the performance metrics. Compared to the the failure detectors found in FreePastry and Bamboo implementations, $\sqrt{s}$-LM improves failure detection latency by 40% while consuming the same amount of bandwidth as the standard solution, while $\sqrt{s}$-BM can reduce bandwidth requirements by 30% for typical failure detection latency targets.

The next section provides a brief overview of related work in failure detection. Section 3 describes our approach and derives the formulas for an optimal failure detector. Section 4 describes our implementation and discusses some of the problems encountered in failure detection in practice. Section 5 evaluates our approach using detailed, long-term data from three realistic traces. Section 6 summarizes our contributions and describes future directions.

2. RELATED WORK

Past work on failure detectors has focused on the *single-node* and *distributed* failure detection problems, which differ substantially from the multi-node failure detection problem we examine. Our work involves different tradeoffs from the former and is complementary to the latter. It arose in the context of peer-to-peer systems, where failure detection is critical, and existing approaches are typically ad hoc.

Chen [10] provides the most comprehensive analysis to date of the quality of service (QoS) guarantees for single-node failure detectors. Given QoS requirements, Chen et al. [11] show how to compute the parameters, in particular the time between heartbeats and estimated network delays, for failure detectors under probabilistic message losses and network delays. Bertier et al. [5] propose an adaptive failure detector that computes the heartbeat period by combining the TCP estimation function with Chen's, while [16] automatically adjusts the heartbeat period based on network characteristics. These approaches examine a single node at a time; they treat a set of nodes being monitored as collection of independent single-node detectors and do not examine the resource tradeoffs between different hosts. In contrast, we specifically focus on multi-node failure detectors and show how to optimally dedicate bandwidth among multiple nodes in the fail-detect set.

Some past work has examined how to incorporate application specific requirements into failure detectors. In [12], the authors use two different timeouts to generate two levels of suspicions. Instead of using a suspect list, φ and Φ failure detectors [22, 13] compute an estimate for each node that captures the likelihood that the node has crashed. This approach provides a more expressive interface from the failure detector to applications. In contrast, we provide a richer interface to the failure detector by which applications can specify their performance goals and resource constraints, and achieve those goals within the constraints.

Recent work has examined how to adaptively calculate a homogeneous probing period for peers in an overlay network [8]. The authors derive an analytical model to compute the probing rate given packet loss rates and node lifetimes. The protocol is able to self-tune the probing rate in response to changes in these two parameters. In contrast, we derive an analytical model to compute a node-specific polling rate for each node as opposed to a single, homogeneous probing rate for all nodes, and achieve analytical, closed form solutions for the optimal strategy.

Distributed failure detectors combine suspicions from multiple nodes into an accurate estimate on process status. Chandra and Toueg [9] introduce the concept of unreliable distributed failure detectors and characterize them in terms of completeness and accuracy. They use the failure detector abstraction to solve the consensus and atomic broadcast problems in an asynchronous network model. This approach has formed the foundation of much subsequent work on reducing message complexity and network load [25, 1, 24, 2].

Subsequent work in distributed failure detectors has focused on how to detect failures in a set of servers in a scalable manner as the size of the set grows. Hierarchical failure detectors [6, 36] arrange nodes into a multi-level hierarchy which partitions the monitoring and reporting tasks along a tree to improve scalability. Gulfstream [15] reduces probing traffic by arranging nodes in a circular virtual identifier space and probing only adjacent nodes. Gossip-based failure detectors [31] can improve scalability and reduce failure detection time via random and periodic communication among the monitoring nodes. Recent work by Gupta et al. [21] presents a randomized distributed failure detector that achieves low failure detection latency. A recent study on failure detection in overlay networks [41] empirically examines the performance of five distributed failure detection algorithms, including the basic periodic failure detector that we use for our baseline. FUSE [14] is a distributed failure detector that focuses on lightweight and scalable failure notification. These systems assume the presence of a multi-node failure detector on each node, and focus on the communication among the monitoring nodes. Our approach is complementary to these systems.

Accordion [26] has examined how to efficiently manage bandwidth usage in a distributed hash table. It focuses in dynamically adjusting DHT parameters to achieve low lookup latencies subject to a bandwidth constraint. Our approach to failure detection is orthogonal to optimizations for DHTs and is more broadly applicable.

3. APPROACH

In this section, we first describe the general system model we use as our foundation, then analytically express the tradeoffs involved in failure detection, and finally derive the equations for two variants of an optimal failure detector.

3.1 System Model

The key metrics of interest in a multi-node failure detector are failure detection latency, bandwidth overhead and accuracy. We assume that each node j has a fail-detect set σ_j, $|\sigma_j| = N$, which it needs to monitor for failures. We build on an end-to-end model, in which the preferred way to test if a node has failed is to send an application-specific request and receive a response. Typically, such a request consists of a no-op remote procedure call handled at the application level, though a simpler substitute, such as an ICMP ping packet, may be used in cases where the application shares its fate with the entity responding to the request. We use the term *ping* for a single such packet sent to detect node failure; a *probe* is a series of up to r pings that are separated by a Δ timeout value, sent in sequence to guard against losses in the network.

The current state of the art in failure detection is to pick a fixed period τ and to probe every node $i, i \in \sigma_j$ every τ seconds. If consumed bandwidth is not a concern, low failure detection latencies can be achieved simply by setting $\tau = 0$ and continuously probing every node. This strategy is clearly a terrible choice in practice, as all available bandwidth would be dedicated to failure detection, leaving no surplus bandwidth for useful work. We therefore limit the maximum per-node bandwidth the system can dedicate to failure detection to T_B[1].

A node j probes a node i in its fail-detect set, $\sigma_j, i \in \sigma_j$ with period τ_i, $\tau_i > r\Delta$, where Δ is the ping timeout period. If any ping elicits a response, the probe is successful, the series of pings are terminated, and the node is deemed alive. If more than r consecutive pings to a given node elicit no response, the destination node is marked as having failed.

Table 1 summarizes all the variables used in the following analysis. A key variable in the analysis is the estimated lifetime for a given node, which is denoted by l_i. Given these key variables, the next section derives the relevant formulas for optimal failure detection.

[1]We assume a uniform T_B for all nodes in the system for simplicity of discussion; it is straightforward to extend the analysis to accommodate a node-specific bandwidth cap.

Parameter	Description
τ_i	Probing period for node i in the fail-detect set
N	Number of nodes in the fail-detect set
L	Failure detection latency
T_L	Targeted failure detection latency
B	Bandwidth consumed for failure detection
T_B	Targeted bandwidth consumption
l_i	Estimated lifetime for node i
d_i	Mean time to recovery for node i
α	Desired failure detection accuracy
p	Round-trip packet loss probability
Δ	Timeout for a ping response
r	Maximum number of ping packets in a probe
q	Expected number of ping packets in a probe
s	Ping packet size

Table 1: Notation. Key variables used in the analysis.

3.2 $\sqrt{s}$–LM: Latency-Minimizing Optimal Failure Detector

First, we examine a failure detector designed to achieve the lowest possible failure detection latency while remaining within a given bandwidth budget.

Failure detection latency is the time between the time a given target node failed, either due to network or node failure, and the time the detector decides that the node failed. If the probing period is τ_i and a node is considered to be down after a probe is unsuccessful, then the average failure detection latency is $\frac{\tau_i}{2}$.

Consider the behavior of the system over a long time period κ. For each neighbor i, a node will use bandwidth $\frac{s}{\tau_i}$ to send out a probe every probing interval, where s is the size of the ping packet. Therefore, the total amount of network bandwidth consumed at each node for monitoring the fail-detect set over the period κ is

$$B = \sum_{i=1}^{N} \frac{sq}{\tau_i} \qquad (1)$$

In the equation above, q is the expected number of ping packets in a probe before the probe either successfully elicits a response or is unsuccessful. The precise definition of q depends on the operating environment, and is derived in Section 3.6. For now, note that it is a constant between 1 and r; 1 if node i is continuously up and there are no network losses, and r if the node is permanently down.

From the lifetime estimate of a node, we can calculate the expected number failures of that node over time κ, which is $\frac{\kappa}{l_i}$. Average latency of detecting failed neighbor i is $\frac{\tau_i}{2} + r\Delta$. Consequently, the average latency of detecting failed neighbors over time κ is given by:

$$L = \frac{\sum_{i=1}^{N}(\frac{\tau_i}{2} + r\Delta) \times \frac{\kappa}{l_i}}{\sum_{i=1}^{N} \frac{\kappa}{l_i}} \qquad (2)$$

We can now cast the latency-minimizing failure detection problem as a mathematical optimization problem: minimize average latency of detecting failed neighbors (Eq. 2) subject to bandwidth constraint $B \leq T_B$ (Eq. 1). We introduce Lagrange multiplier λ, and use the independence of the terms under the summation to yield the following equation.

$$\frac{1}{2l_i} = -\lambda \frac{sq}{\tau_i^2}, \forall i, 1 \leq i \leq N \tag{3}$$

Given this equation, we can solve for τ_i using the constraint $B \leq T_B$ to reach the following solution:

$$\tau_i^* = \frac{sq}{T_B}\sqrt{l_i}\left(\sum_{j=1}^{N}\frac{1}{\sqrt{l_j}}\right), \forall i, 1 \leq i \leq N \tag{4}$$

A simple example illustrates the potential benefits of this approach. Suppose that a system consists of 40 nodes whose lifetimes are drawn from a bimodal distribution. Twenty short-lived hosts have lifetimes of 1 hour, while the other 20 long-lived hosts have lifetimes of 225 hours. Suppose that the amount of bandwidth devoted to failure detection is 1 kB/sec, and that the size of a ping packet is 100 bytes. The traditional approach to failure detection would ping both the long and short-lived node every 4 seconds, consuming 1kB/sec and yielding an average failure detection latency of 2 seconds. Our latency-minimizing optimal failure detector ($\sqrt{s}$-LM) algorithm instead pings long- and short-lived nodes every 32 and 2.13 seconds respectively, consumes 1 kB/sec, and achieves an average failure detection latency of 1.2 seconds.

3.3 $\sqrt{s}-BM$: Bandwidth-Minimizing Optimal Failure Detector

In some deployment scenarios, the amount of bandwidth dedicated to system maintenance is not capped, and the critical performance metric for the failure detector is the average detection latency. Next, we examine a failure detector designed to achieve a targeted failure detection latency, while minimizing the bandwidth consumed to achieve the desired level of performance.

The derivation of the parameters for a bandwidth-minimizing optimal failure detector is similar to the latency-minimizing optimal failure detector, except that the mathematical optimization problem is slightly different. Namely, $\sqrt{s}$-BM will minimize bandwidth (Eq. 1) given a target failure detection latency (Eq. 2) of T_L. We introduce Lagrange multipliers to find the minima and solve for the optimal τ_i subject to the constraint $L \leq T_L$, similar to the approach in the previous subsection. This yields the following equation.

$$-\frac{sq}{\tau_i^2} = \lambda\left(\frac{1}{2l_i\sum_{j=1}^{N}\frac{1}{l_j}}\right) \tag{5}$$

Now, we can solve for τ_i using Equation 2, the constraint $L \leq T_L$ and Equation 5 to reach the following solution.

$$\tau_i^* = \frac{2(T_L - r\Delta)(\sum_{j=1}^{N}\frac{1}{l_j})\sqrt{l_i}}{\sum_{j=1}^{N}\frac{1}{\sqrt{l_j}}}, \forall i, 1 \leq i \leq N \tag{6}$$

We can illustrate the potential benefits of this approach with a simple example. Assuming the same sample setup as in the preceding section, a traditional, periodic failure detector that targets 2 second average failure detection latency would ping nodes every 4 seconds, and consume 1 kB/sec as a consequence. Our $\sqrt{s}$-BM result instead pings long- and short-lived nodes every 56.5 and 3.77 seconds respectively, achieves 2 second failure detection latency on average, and consumes only 0.56 kB/sec.

3.4 Worst-case guarantees

Our derivation so far targeted a minimal or desired detection latency in the average case. The approach described above does not (yet) provide a bound on worst-case detection latency. Such a bound can be accommodated by specifying the worst-case detection latency, Γ, solving for τ_i^* using the $\sqrt{s}$-LM or the $\sqrt{s}$-BM equations above, and setting τ_i to Γ in cases where $\tau_i^* \geq \Gamma$. Subtracting the resulting bandwidth consumption from B and re-solving for the remaining τ_j will yield the optimal solution for the average-case behavior subject to the worst-case bound.

3.5 Probe length

Accuracy of failure detection depends significantly on transient losses encountered in the network. Since network failures are indistinguishable from host failures, losses in the network can lead to false positives where a node erroneously concludes that another node is down because all pings or ping responses were lost in the network. We consider a ping lost if a response is not received within a timeout period Δ. Previous work on temporal dependence of packet losses [40] indicates that for choices of $\Delta > 1s$, the losses are likely to be independent, yielding a probability of p^r for losing r consecutive ping packets, where p is the packet loss rate. In order to achieve a false positive rate at α, $0 \leq \alpha \leq 1$, we can pick $r = \log_p(1-\alpha)$, and send r such consecutive pings after timeouts before declaring a node down.

3.6 Expected number of pings

Having derived analytical formulas for the optimal probing strategy, we are now in a position to compute q, the number of times a node expects to send ping packets to a monitored node before it successfully receives a response or concludes that the node is down. If the node is down, the expected number of pings is simply $q_i^{down} = r$. If the host is up, and the probability of loss of the ping-response pair is p, the expected number of ping packets is the sum of the number of pings for successfully receiving a response and declaring the node up (the summation term), and the number of packets for the remaining case where all packets are lost in the network and the node is erroneously declared down:

$$q_i^{up} = \left(\sum_{t=1}^{r} t p^{t-1}(1-p) \right) + r p^r = \frac{1-p^r}{1-p} \tag{7}$$

The precise value for q depends on the manner in which the failure detector is used. Failure detectors can be classified into two categories based on how they manage their fail-detect sets. In *static* fail-detect sets, the set of nodes being monitored does not change in response to failures. Static fail-detect sets are encountered in practice in settings where the system is closed and set members are known a-priori; for instance, replicated DNS servers running BIND monitor the server set without replacement through failures. In *dynamic* fail-detect sets, failed nodes in the fail-detect set are removed at run-time. They may potentially be replaced with alternative nodes, but nevertheless the invariant is that the fail-detect set consists of all live nodes at all times. Dynamic fail-detect sets are encountered in settings where system membership can change at run time and newly arriving nodes can be tasked with explicitly announcing their arrival. For instance, the Pastry overlay replaces failed nodes in a node's vicinity (leaf-set) by picking the closest alive nodes in the circular identifier space. In effect, failure detectors are generic event detectors; static fail-detect sets correspond to event detectors that monitor both failure and recovery events, whereas dynamic fail-detect sets correspond to event detectors that monitor only failures.

In a dynamic fail-detect set, failed nodes are probed at most once following a failure. Consequently, $q = \text{avg}(q_i^{up})$. In a static fail-detect set, nodes are probed even when they are down. We need consider not only the lifetime, but also d_i, the mean time to recovery, in order to model q accurately. Taking into account the percentage of time a node is up or down, and the relative costs of probing alive and failed nodes leads to the following formula:

$$q = \sum_{i=1}^{N} \left(\frac{d_i}{l_i + d_i} q_i^{down} + \frac{l_i}{l_i + d_i} q_i^{up} \right) \tag{8}$$

This completes the full derivation of all parameters required for the latency- and bandwidth-minimizing optimal failure detectors from first principles. The critical inputs to the formulas are the estimates for node lifetimes, whose accuracy greatly determines the performance of the failure detector. The next section examines the lifetime estimates as as well as other practical considerations in implementing $\sqrt{s}$ failure detectors.

4. IMPLEMENTATION

In order to actually build an optimal detector, the parameters used in the analysis need to be accurately and efficiently estimated so the formula can be computed. The key parameter a practical implementation needs to estimate is l_i, a node's lifetime.

The choice of an estimation technique depends on the distribution of node lifetimes (also known as session lengths) as well as their variation over time. Note that for the defunct case where node lifetimes are homogeneous, i.e. identical for all nodes, our optimal solution yields the same result as traditional, periodic failure detectors. In practice, studies have shown that node lifetimes vary substantially, which provides the opportunity for our approach to significantly outperform simple periodic polling.

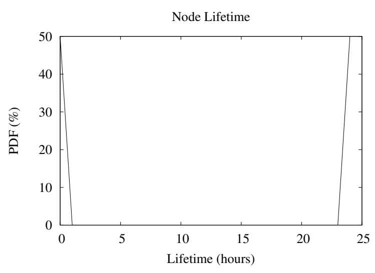
We examine three simple techniques for estimating node lifetimes, a moving average, an exponential moving average, and a hybrid approach. A node monitoring other nodes' failures has ready access to the node's last session length. In Moving Average (MovAvg), the monitoring node will calculate the average lifetime of last k sessions, and use it for its estimated value of l_i. In Exponential Moving Average (ExpAvg), a node estimates the average lifetime l_i after the t^{th} session using the estimated lifetime l_i^{t-1} and the duration of the last session λ_i^t according to $l_i = (1-\beta) l_i^{t-1} + \beta \lambda_i^t$. When β is large, the exponential average weights recent session durations more heavily than previous sessions.

Our Hybrid approach is driven by the observation, from PlanetLab data, that nodes tend to alternate between periods of frequent activity, followed by long quiescent periods of either being up or down. We capture this behavior with a bimodal moving average, which uses a "low" moving average table for sessions below a certain threshold κ and a "high" moving average table for the rest. When the current session length is below κ, we use the estimate from the "low" moving average. Otherwise, we use the estimate from the "high" moving average. The constant κ is set to 24 hours in our implementation, and the low and high tables are updated in response to failures, as well as when a monitored node's current session length exceeds the current lifetime estimate.

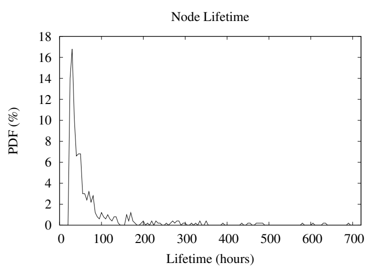
In our implementation, each node calculates the probing period for the nodes in its fail-detect set using the previously derived formulas, informed by estimates of node lifetime. The node lifetime estimates are computed using the Hybrid approach; we later provide a comparison of all three estimators and find empirically that it achieves the best results for the actual temporal variations in the PlanetLab data set. When a fresh new node is added, its lifetime estimate can be computed by querying the node (which will omit network failures but capture node uptime up to that point in time), by querying other nodes for their estimates (which assumes that the network characteristics are similar from different vantage points), or by simply assuming a representative default value, such as the average session duration for all nodes. The selection depends on the deployment scenario; in settings with moderate to high node churn, the initial estimate does not materially impact performance.

The polling frequency for each node needs to be recomputed whenever inputs change significantly. Our implementation recomputes the polling frequency for each node reactively, whenever the fail-detect set is modified in response to node joins or failures, as well as proactively, every $\Omega = 5$ minutes.

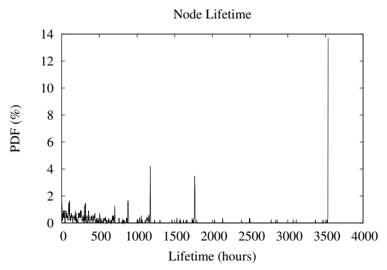
Note that, while our optimal solution caps the amount of bandwidth stemming from failure detection actively performed by a given host, it does not take into account the bandwidth that the same node consumes when responding to pings initiated by other hosts. This simplification is inten-



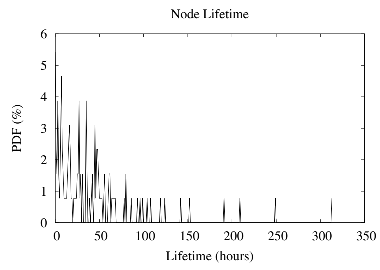
(a) Bimodal Lifetime Distribution



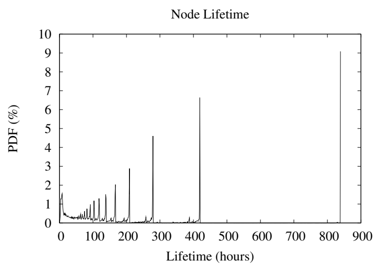
(b) Pareto Lifetime Distribution



(c) PlanetLab Lifetime Distribution



(d) Web Sites Lifetime Distribution



(e) Microsoft Desktops Lifetime Distribution

Figure 1: Average Lifetime distributions for the synthetic and real traces.

tional, since neighbor management algorithms in structured peer-to-peer systems typically provide (and themselves rely on) in-degree balancing, which ensures that the incoming load due to pings will not be concentrated at any one node and can be derived easily. A $\sqrt{s}$ failure detector implementation for an unstructured peer-to-peer system may need to pay attention to node in-degree imbalance if the overlay does not already guarantee in-degree balancing [38].

While failure detectors are typically designed as stand-alone modules independent from the applications they serve, an implementation can save bandwidth through a tighter integration between the detector and higher-level system functionality. In particular, aggressively harvesting information on node failures from higher levels can enable the detector to delay or skip probes, saving bandwidth. A typical $\sqrt{s}$ failure detection implementation will keep a countdown timer associated with each node it is monitoring, set initially to τ_i for that host. When an indication is received that the target node is up, for instance, via an application-level invocation to or from the target node, or via a probe in the reverse direction from the target back to the monitoring node, the timer can be reset to τ_i. Hence a busy overlay can piggyback failure detection onto naturally occurring traffic. Since the bandwidth savings possible with this optimization are highly application-specific, we ignore this optimization in our evaluation.

We have implemented parameterizable failure detector toolkit for $\sqrt{s} - LM$ in Java and Python. The current implementations are approximately 300 lines of code. $\sqrt{s} - LM$ failure detector allows the application to dynamically add or remove nodes from the failure detector monitor set. $\sqrt{s} - LM$ calls back to application when it detects a node is not available. Figure 2 presents the interface for the failure detector module.

We believe this interface is flexible and easy to adopt. The actual implementation of the ping process is deferred to the application for maximum flexibility. And the interface enables the application to notify the failure detector of application level traffic with the node, thus indicating that the node is up and obviating the need to send explicit pings.

5. EVALUATION

In this section, we evaluate the efficacy of $\sqrt{s}$ failure detectors using long-term uptime measurements collected on PlanetLab, web servers and Microsoft Desktop PCs. We examine two metrics of importance, failure detection latency and bandwidth consumption. We compare our approach to common failure detection mechanisms deployed in real systems.

We compare our $\sqrt{s}$-LM and $\sqrt{s}$-BM failure detectors to the reference failure detector implementations used in FreePastry and Bamboo, two widely deployed distributed systems. In FreePastry (Version 1.3.2), each node periodically probes all nodes in its fail-detect set with a default period of 60 seconds. The node is marked as down when a ping packet is lost. In Bamboo (version 20050701), each node periodically pings all nodes in its fail-detect set every 20 seconds. An initial ping failure within 20 seconds marks the node as "suspect," which is followed immediately by a second ping with a 60-second timeout. A subsequent failure marks the node as having failed. Since the Pastry and Bamboo failure detectors use different probing schemes, they achieve different accuracies. We consequently adjust our algorithms to match the accuracy achieved by Pastry and Bamboo for a fair comparison.

We quantify the benefits of the $\sqrt{s}$-LM and $\sqrt{s}$-BM algorithms under five different lifetime distributions, two synthetic and three based on long-term uptime data from Plan-

```
// Add a node into the failure detector
public void addNode(Node node);

// Delete a node from the failure detector monitor
// set
public void deleteNode(Node node);

// Application should report when it receives
// a ping response from other nodes
public void pingResponse(Node node, long seqNum);

// Application can optionally report the node is alive
// to reduce ping traffic.
// e.g. regular data traffic was received from the node.
public void alive(Node node);

// Callback to the application to send a ping.
interface SendPingCB {
  public void sendPing(Node node, long seqNum);
}

// Callback to the application to notify the node is
// not available
interface NodeFailedCB {
  public void nodeFailed(Node node);
}
```

Figure 2: Interface for the failure detector module

etLab, web servers, and Microsoft Desktop PCs. We first examine a *bimodal* lifetime distribution, where nodes are either long- or short-lived. Such a scenario might be encountered in a system where there is a mix of client- and server-class nodes, such as a peer-to-peer system with superpeers. We pick a bimodal distribution with peaks at 30 and 300 minutes. We also examine a *Pareto* lifetime distribution, where the probability of a node dying before time t is

$$Pr(\text{lifetime} < t) = 1 - (\frac{\beta}{t})^{\alpha} \tag{9}$$

where α and β are the shape and the scale parameters of the distribution. We use $\alpha = 0.83$ and $\beta = 1560$ sec in our simulations. These parameters are used to create a synthetic Pareto distribution to closely simulate the previous data from the Gnutella network. While these parameters are inspired by [26, 34], we present results from the synthetic bimodal and Pareto lifetime distributions to provide insight into the operation of the optimal solution by showing the types of gains possible in two well-characterized settings.

For realistic evaluations, we examine three traces, PlanetLab [37], web servers [4] and Microsoft Desktop PCs [7], which are used in [19]. We first examine the actual lifetime distribution collected via a long-running all-pairs ping measurement study over 561 nodes. We examine 5 months of all pairs ping data collected between June 1 and October 26, 2005. The data contains the result of 10 consecutive pings from each node to all other nodes, sent every 15 minutes. Using the raw all-pairs ping data, we establish a "ground truth" about the status of each node at each time interval.

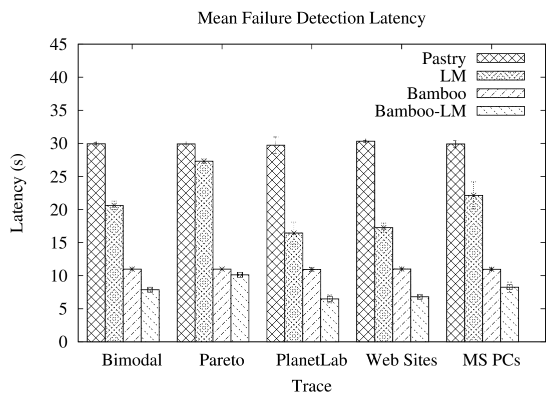


Figure 3: Failure detection latency of the latency-minimizing optimal failure detector.

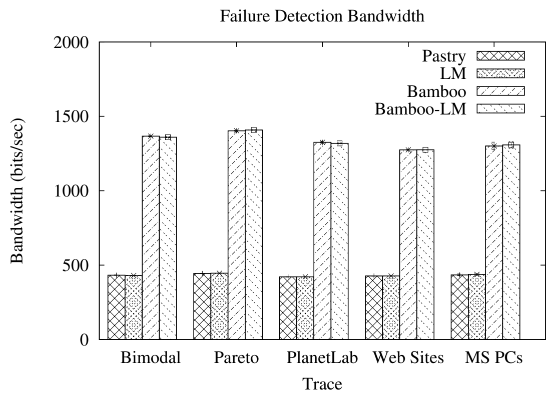


Figure 4: Bandwidth usage of the latency-minimizing optimal failure detector.

We consider a node to be down if none of the nodes can ping it during that interval. In some periods of the PlanetLab trace, there are times when nearly all the nodes are down, due to high load before major deadline conferences, planned system upgrades, and failures of the measurement system. To remove these cases, we elide the periods when less than half the average number of nodes are up, following previous work [19, 39]. We use a similar technique to establish the "ground truth" in other traces.

We also examine the actual lifetime distribution of 129 web sites collected using HTTP requests sent from a single machine at Carnegie Mellon every 10 minutes. We examine 7 months of data collected between September 2001 and April 2002. Finally, we examine the actual lifetime distribution of 51,662 desktop PCs within Microsoft Corporation that were pinged every hour, denoted as "MS PCs". The data spans 35 days beginning July 6, 1999. Figure 1 shows the distribution of node average lifetimes for the synthetic distributions and the realistic traces.

In the simulations, the start and end of a failure is randomized within the polling period in which it is detected.

While the polling period used in collecting the data may have missed failures with very short durations (as will data collected using any practical method based on polling), the three realistic traces represents the most extensive and realistic public uptime data available on geographically distributed systems, and the slight bias away from failures with very short durations affects periodic failure detectors and $\sqrt{s}$ failure detector equally.

Our simulations accurately reflect the costs and latencies of failure detection. We select fail-detect set size N to be 50, and take the size of a ping packet s to be 64 bytes. Nodes keep track of their own session durations and provide an initial estimate to other nodes when they initially join the network; subsequently, failure detectors perform their own measurements and estimates. We examined network loss rates p ranging from 0 to 0.05 [28] and found that loss rates in this range impact the results by less than a few percent. Consequently, we report only the conservative and realistic results from the 0.05 loss rate experiments. All reported measurements represent the average of 9 runs; bars indicate standard deviation for normally distributed data.

We first examine the central tradeoff between bandwidth consumption and achieved failure detection latency for latency-minimizing optimal failure detector ($\sqrt{s}$-LM) at two different accuracy levels, and compare them to Pastry and Bamboo under three different lifetime distributions. Figures 3 compare the average failure latency achieved by $\sqrt{s}$-LM to that of traditional, periodic failure detectors. $\sqrt{s}$-LM achieves a 40% improvement in update latency for both the real-world PlanetLab data, and web sites data while it achieves a 30% improvement in update latency for Microsoft PCs data. Both improvement in PlanetLab and web sites significantly exceeds the improvement $\sqrt{s}$-LM provides when node lifetimes are bimodal (32%) or Pareto (10%), partly because the nodes exhibit highly skewed session times. $\sqrt{s}$-LM has only 10% improvement in failure detection latency because most of the nodes in Pareto distrubtion has very short lifetime. Fig 4 shows that $\sqrt{s}$-LM consumes approximately the same amount of bandwidth as Pastry and Bamboo while achieving significantly better detection latencies. This is not surprising, as our solution was crafted to take the desired bandwidth consumption as a constraint.

Next, we examine the bandwidth consumption and achieved failure detection latency for $\sqrt{s}$-BM, parameterized to match the average latency of Pastry and Bamboo. Figures 5 and 6 show that our bandwidth-minimizing optimal failure detector consumes approximately more than 30% less bandwidth for the realistic traces, while approximately matching the average failure detection latency. The 20% discrepancy between the targeted and achieved failure detection latency is due to inaccuracy in estimated node lifetimes.

We next evaluate what kind of a strategy to use for estimating node lifetimes in a realistic $\sqrt{s}$-LM implementation. We examine moving average (MovAvg), exponential moving average (ExpAvg), and the hybrid approach (Hybrid). To provide a point of comparison, we also examine a lifetime estimator, called FullInfo, that has access to the full 5-month duration of the trace, and uses $l_i = \bar{l}$, the average of all session durations as its estimate. Note that such a failure detector is unrealistic, as it has access to future session lengths, and implicitly assumes that session lifetimes are distributed normally, but nevertheless provides a yardstick for the performance of other lifetime estimators. We examine exponential moving average with $\alpha = 0.75$; the size of the moving average windows for MovAvg and Hybrid are set to 3. Figure 7 shows that the Hybrid and ExpAvg estimators achieve results that are within a few percent of FullInfo.

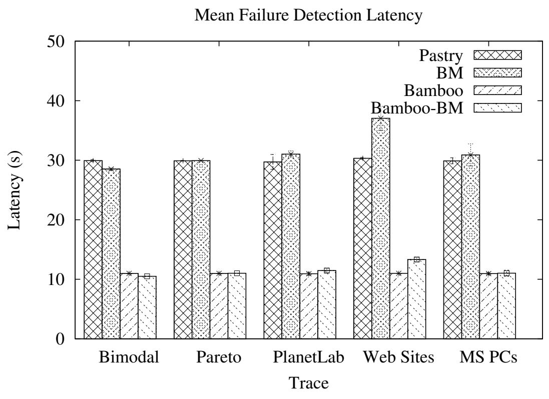


Figure 5: Failure detection latency of the bandwidth-minimizing optimal failure detector.

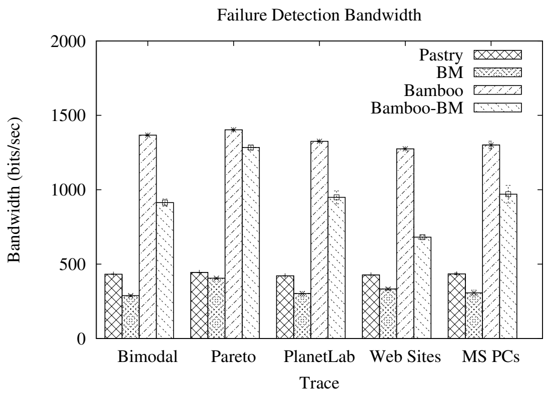


Figure 6: Bandwidth usage of the bandwidth-minimizing optimal failure detector.

Figure 8 provides insights into the operation of $\sqrt{s}$-LM by plotting the average ping period $\overline{\tau}_i$ for each of the nodes sorted by average lifetime. $\sqrt{s}$-LM preferentially pings the nodes with shorter lifetimes more often, dedicating the required bandwidth to quickly detect failures, while Pastry's periodic failure detector pings every node with the same ping period. The probe period ranges from 3.4 seconds for the node with an average lifetime of 15 minutes to 3 minutes for the node with 3537 hours average lifetime.

Finally, we examine the behavior of $\sqrt{s}$-LM as a function of the available bandwidth. Figure 9 shows that $\sqrt{s}$-LM achieves an average of 40% improvement in failure detection latency under bandwidth constraints ranging from 0.5

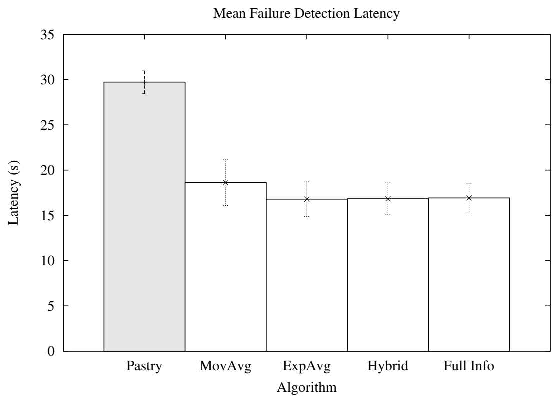


Figure 7: Failure detection latency using different lifetime estimators.

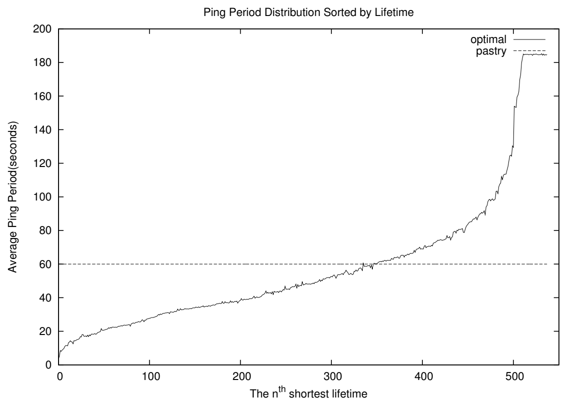


Figure 8: Average ping period sorted by lifetime, as computed by $\sqrt{s}$-LM.

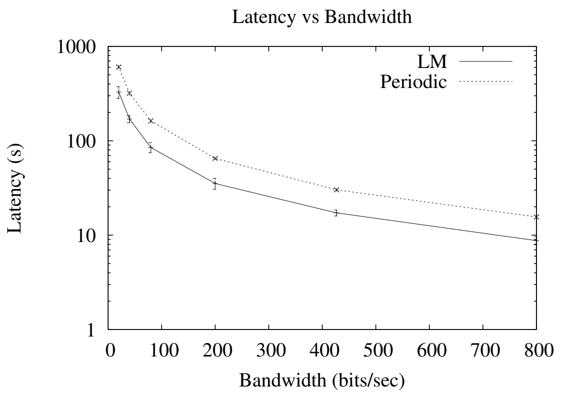


Figure 9: Failure detection latency as a function of bandwidth for the PlanetLab trace.

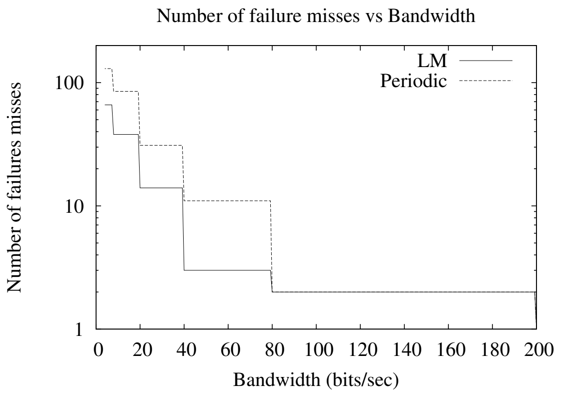


Figure 10: Failure detection accuracy as a function of bandwidth for the PlanetLab trace.

bytes/s to 800 bytes/s. Note that the choice of τ_i may impact the accuracy of the failure detector if the node fails and recovers between successive probes. Figure 10 shows the number of missed failures under different bandwidth limits. With an extremely tight bandwidth limit of 1 byte/s, the periodic failure detector misses 85 failures while $\sqrt{s}$-LM only misses 38, and $\sqrt{s}$-LM achieves much better accuracy across a range of bandwidths, mostly because it concentrates its polling activity on nodes likely to fail. While this experimentally confirmed behavior is encouraging, missed failures are not a significant issue in settings with modest amounts of bandwidth dedicated to monitoring. In the PlanetLab trace, neither periodic nor $\sqrt{s}$-LM miss any failures when bandwidth used for failure detection exceeds 0.2 Kbits/sec,

Overall, $\sqrt{s}$ failure detectors utilize the critical resource, bandwidth, optimally, achieving significant improvements in detection time for a targeted bandwidth or savings in overhead for a targeted latency compared to traditional periodic failure detectors.

6. CONCLUSIONS

In this paper, we examined the problem of how to allocate bandwidth to monitoring nodes in a multi-node failure detector. We formalized the problem, expressed it analytically in a form tractable for mathematical optimization, and derived closed-form solutions. We developed solutions to two variants of the problem, suitable for two different settings: our *latency-minimizing* optimal failure detector achieves the lowest failure detection latency on average given a bandwidth constraint, while our *bandwidth-minimizing* optimal failure detector meets a targeted detection latency using the least amount of bandwidth necessary to reach that goal. Evaluation of our approach using real-world trace data spanning several months from PlanetLab hosts, web sites, and Microsoft PCs indicate that the approach is effective. Compared to traditional failure detectors, our optimal approach can reduce detection latencies by 40% while using the same amount of bandwidth, or reduce bandwidth consumed by 30% for a target detection latency.

Overall, the proposed approach lends itself to a straightforward implementation, is complementary to decades of work in distributed failure detectors, and is suitable for recently emerging peer-to-peer systems. We have built a parameterizable failure detector toolkit $\sqrt{s}-LM$ and are currently integrating it into our peer-to-peer systems [30, 29, 35]. We hope that such a toolkit will lead to a more principled, less ad hoc implementations of failure detectors among distributed system practitioners.

Acknowledgements

We would like to thank Jeremy Stribling and Brighten Godfrey making PlanetLab, web servers and Microsoft Desktop PCs uptime data publicly available. This work was supported in part by National Science Foundation Grants 0430161, CCF-0424422 (TRUST), and CAREER award 0546568.

7. REFERENCES

[1] M. K. Aguilera, W. Chen, and S. Toueg. Heartbeat: a Timeout-free Failure Detector for Quiescent Reliable Communication. In *Proceedings of the International Workshop on Distributed Algorithms*, Saarbrücken, Germany, Sept. 1997.

[2] M. K. Aguilera, W. Chen, and S. Toueg. Failure Detection and Consensus in the Crash-Recovery Model. In *Proceedings of the International Symposium on Distributed Computing*, Andros, Greece, Sept. 1998.

[3] Y. Amir, D. Dolev, S. Kramer, and D. Malkhi. Transis: A Communication Sub-system for High Availability. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, Boston, Massachussetts, July 1992.

[4] M. Bakkaloglu, J. J. Wylie, C. Wang, and G. R. Ganger. On Correlated Failures in Survivable Storage Systems. Technical Report CMU-CS-02-129, School of Computer Science, Carnegie Mellon University, May 2002.

[5] M. Bertier, O. Marin, and P. Sens. Implementation and Performance Evaluation of an Adaptable Failure Detector. In *Proceedings of the International Conference on Dependable Systems and Networks*, Washington, DC, June 2002.

[6] M. Bertier, O. Marin, and P. Sens. Performance Analysis of Hierarchical Failure Detector. In *Proceedings of the International Conference on Dependable Systems and Networks*, San Francisco, California, June 2003.

[7] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. In *Proceedings of the SIGMETRICS Conference*, Santa Clara, California, June 2000.

[8] M. Castro, M. Costa, and A. I. T. Rowstron. Performance and Dependability of Structured Peer-to-Peer Overlays. In *Proceedings of the International Conference on Dependable Systems and Networks*, Florence, Italy, June 2004.

[9] T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of the ACM*, 43(2):225–267, Mar. 1996.

[10] W. Chen. *On the Quality of Service of Failure Detectors*. PhD thesis, Cornell University, May 2000.

[11] W. Chen, S. Toueg, and M. K. Aguilera. On the Quality of Service of Failure Detectors. *IEEE Transactions on Computers*, 51(5), May 2002.

[12] X. Defago, P. Felber, and A. Schiper. Optimization Techniques for Replicating CORBA Objects. In *Proceedings of the IEEE International Workshop on Object-oriented Real-time Dependable Systems*, Santa Barbara, California, Jan. 1999.

[13] X. Defago, P. Urban, N. Hayashibara, and T. Katayama. Definition and Specification of Accrual Failure Detectors. In *Proceedings of the International Conference on Dependable Systems and Networks*, Yokohama, Japan, June 2005.

[14] J. Dunagan, N. J. A. Harvey, M. B. Jones, D. Kostic, M. Theimer, and A. Wolman. FUSE: Lightweight Guaranteed Distributed Failure Notification. In *Proceedings of the Symposium on Operating System Design and Implementation*, San Francisco, California, Dec. 2004.

[15] S. A. Fakhouri, G. S. Goldszmidt, and I. Gupta. Gulfstream - A System for Dynamic Topology Management in Multi-domain Server Farms. Technical Report RC21954, IBM T.J. Watson Research Center, Feb. 2001.

[16] C. Fetzer, M. Raynal, and F. Tronel. An Adaptive Failure Detection Protocol. In *Proceedings of the Pacific Rim Symposium on Dependable Computing*, Seoul, Korea, Dec. 2001.

[17] B. Glade, K. P. Birman, R. Cooper, and R. van Renesse. Light-Weight Process Groups in the ISIS System. *Distributed Systems Engineering*, 1(1):29–36, Sept. 1993.

[18] Gnutella. The Gnutella Protocol Specification. `http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf`.

[19] P. B. Godfrey, S. Shenker, and I. Stoica. Minimizing Churn in Distributed Systems. In *Proceedings of the SIGCOMM Conference*, Pisa, Italy, Sept. 2006.

[20] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Rennesse. Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. In *Proceedings of the International Workshop on Peer-to-Peer Systems*, Berkeley, California, Feb. 2003.

[21] I. Gupta, T. D. Chandra, and G. S. Goldszmidt. On Scalable and Efficient Distributed Failure Detectors. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, Newport, Rhode Island, Aug. 2001.

[22] N. Hayashibara, X. Defago, and T. Katayama. Two-ways Adaptive Failure Detection with the φ-failure Detector. In *Proceedings of the International Workshop on Adaptive Distributed Systems*, Italy, Oct. 2003.

[23] M. G. Hayden. *The Ensemble System.* PhD thesis, Cornell University, Jan. 1998.

[24] M. Larrea, S. Arevalo, and A. Fernandez. Efficient Algorithms to Implement Unreliable Failure Detectors in Partially Synchronous Systems. In *Proceedings of the International Symposium on Distributed Computing*, Bratislava, Slovak Republic, Sept. 1999.

[25] M. Larrea, A. Fernandez, and S. Arevalo. Optimal Implementation of the Weakest Failure Detector for Solving Consensus. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, Portland, Oregon, July 2000.

[26] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek. Bandwidth-Efficient Management of DHT Routing Tables. In *Proceedings of the Symposium on Networked System Design and Implementation*, Boston, Massachussetts, May 2005.

[27] L. E. Moser, P. M. Melliar-Smith, D. A. Argarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: A Fault-tolerant Multicast Group

Communication System. *Communications of the ACM*, 39(4):54–63, Apr. 1996.

[28] V. Paxson. End-to-End Internet Packet Dynamics. In *Proceedings of the SIGCOMM Conference*, France, Sept. 1997.

[29] V. Ramasubramanian, R. Peterson, and E. G. Sirer. Corona: A High Performance Publish-Subscribe System for the World Wide Web. In *Proceedings of the Symposium on Networked System Design and Implementation*, San Jose, California, May 2006.

[30] V. Ramasubramanian and E. G. Sirer. The Design and Implementation of a Next Generation Name Service for the Internet. In *Proceedings of the SIGCOMM Conference*, Portland, Oregon, Aug. 2004.

[31] R. V. Renesse, Y. Minsky, and M. Hayden. A Gossip-Style Failure Detection Service. In *Proceedings of the International Conference and Distributed Systems Platforms and Open Distributed Processing*, Vienna, Austria, Sept. 1998.

[32] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proceedings of the USENIX Technical Conference*, Boston, Massachussetts, June 2004.

[33] A. Rowstorn and P. Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany, Nov. 2001.

[34] S. Saroiu, K. Gummadi, and S. D. Gribble. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Systems Journal*, 9(2):170–184, Aug. 2003.

[35] Y. J. Song, V. Ramasubramanian, and E. G. Sirer. Optimal Resource Utilization in Content Distribution Networks. Technical Report TR2005-2004, Cornell University, Computing and Information Science, Ithaca, New York, Nov. 2005.

[36] P. Stelling, I. Foster, C. Kesselman, C. Lee, and G. von Laszewski. A Fault Detection Service for Wide Area Distributed Computations. In *Proceedings of the IEEE Symposium on High Performance Distributed Computing*, Chicago, Illinois, July 1998.

[37] J. Stribling. PlanetLab All Pairs Ping Data. `http://pdos.csail.mit.edu/~strib/pl_app`.

[38] V. Vishnumurthy and P. Francis. On Heterogeneous Overlay Construction and Random Node Selection in Unstructured P2P Networks. In *Proceedings of the INFOCOM Conference*, Barcelona, Spain, Apr. 2006.

[39] H. Weatherspoon, B.-G. Chun, C. W. So, and J. Kubiatowicz. Long-Term Data Maintenance in Wide-Area Storage Systems: A Quantitative Approach. Technical Report UCB/CSD-05-1404, University of California-Berkeley, July 2005.

[40] M. Yajnik, S. B. Moon, J. F. Kurose, and D. F. Towsley. Measurement and Modeling of the Temporal Dependence in Packet Loss. In *Proceedings of the INFOCOM Conference*, pages 345–352, New York, New York, Mar. 1999.

[41] S. Zhuang, D. Geels, I. Stoica, and R. Katz. On Failure Detection Algorithms in Overlay Networks. In *Proceedings of the INFOCOM Conference*, Miami, Florida, Mar. 2005.