Exploiting Availability Prediction in Distributed Systems

James W. Mickens and Brian D. Noble
EECS Department, University of Michigan
Ann Arbor, MI, 48103
{*jmickens,bnoble*}@*umich.edu*

Abstract

Loosely-coupled distributed systems have significant scale and cost advantages over more traditional architectures, but the availability of the nodes in these systems varies widely. Availability modeling is crucial for predicting per-machine resource burdens and understanding emergent, system-wide phenomena. We present new techniques for predicting availability and test them using traces taken from three distributed systems. We then describe three applications of availability prediction. The first, availability-guided replica placement, reduces object copying in a distributed data store while increasing data availability. The second shows how availability prediction can improve routing in delay-tolerant networks. The third combines availability prediction with virus modeling to improve forecasts of global infection dynamics.

1 Introduction

Cooperative, opt-in distributed systems provide attractive benefits, but face at least one significant challenge: nodes can enter and leave the collective on a whim, because each machine may be separately owned and managed [3, 6, 29]. Such *churn* can lead to significant overheads [5]. However, if one could predict the availability of even a portion of the nodes with reasonable accuracy, one could reduce these costs by planning for changing availability instead of reacting to it.

This paper introduces new techniques for predicting node availability. Our predictors are less conservative than previous analytical models [2, 5] and more accurately capture phase relationships between the availability of different nodes [30]. We test our predictors using availability traces from the PlanetLab distributed test bed [33], the Microsoft Corporation [6], and the Overnet distributed hash table [3]. Each set of machines has a distinct predictability profile, and we explain the differences in predictability by uncovering the generic classes of uptime patterns found in each data set.

We then provide three applications of our prediction techniques. First, we show how to reduce object copying in a distributed hash table by biasing replicas towards highly available nodes. We can eliminate the majority of non-optimal copies incurred by the standard replication scheme while increasing data availability by at least a factor of two.

We then apply our predictors to two delay-tolerant networking scenarios [13], using them to approximate the role of *contact oracles* [16]. Under reasonable load factors, our predictive schemes reduce delivery latency to within 15% of the latency provided by an oracle.

Finally, we show how explicit representations of machine availability can improve the fidelity of epidemic spreading models [17, 25, 34]. These models typically assume that nodes are always online, so they over-estimate infection levels. By incorporating the availability fluctuations found in real systems, these models can capture diurnal variations in the spreading rate and forecast steady state infection levels that are much closer to those actually exhibited.

2 Related Work

There are many empirical studies of availability in distributed systems. Most used active network probing to detect uptime changes. Bolosky *et al* described the uptimes of over 50,000 PCs belonging to the Microsoft Corporation [6]. Saroiu *et al* studied Napster and Gnutella, popular peer-to-peer file trading services [29]. Douceur performed a meta-analysis of availability data [8], examining the Microsoft, Gnutella, and Napster traces, as well as a trace from a sampling of global Internet hosts [20]. Douceur posited two broad classes of machine availability; those in the first are almost always online, whereas those in the second have diurnal uptime periods. Bhagwan *et al* studied nodes in the Overnet DHT and also found diurnal uptime patterns [3]. We extend these binary categorizations by providing a richer taxonomy of uptime classes and automatic methods for identifying these classes in availability traces.

Rather than rely on coarse-grained probing, other studies have used operating system logs to infer downtime. For example, Simache's analysis of a Unix workstation cluster found a median downtime of 38.5 minutes [31]; roughly 35% of reboots were caused by 10% of the machines, primarily between 8AM and 6PM.

www.manaraa.com

Analytic models of cooperative systems typically include a parameter for host availability. Douceur and Wattenhofer expressed a machine's availability in terms of its fractional downtime [10]. To account for time-of-day effects on global aggregate availability, Bhagwan *et al* assumed a pessimistic mean availability based only on hosts online at night [2]. Blake and Rodrigues also used conservative, average-based metrics in their model [5].

Such conservative estimates are useful in provisioning against worst-case scenarios, but they cannot predict the evolution of the system over time, and they cannot predict individual node behavior well if a system contains heterogeneous availability patterns. To do these things, we need to estimate individual or aggregate availability at multiple points in the future; our new schemes do this. However, we do not investigate the long term admission/drop-out rate, which is also a major issue in peer-to-peer environments [3].

In the computational grid community, availability prediction is done by fitting empirically observed uptime traces to well-known statistical distributions such as the Pareto or Weibull distribution [24]. Using the derived model parameters, one can estimate how long a random machine will be online before failing. This approach suffers from limitations similar to those of the conservative estimates.

Several cooperative systems have been designed to deal with varying host availability. Total Recall is a peer-to-peer storage system which adjusts replication strategies to meet specified data availability goals [4]. It adapts to changing file workloads and node churn, providing two methods for maintaining data redundancy. In eager repair, the system reacts to a host going down by replicating its data elsewhere. With lazy repair, the system estimates worst-case host availability using Bhagwan's conservative estimates [2]; it then over-replicates data such that redundancy targets are still met in these worst case scenarios. Lazy repair trades increased disk requirements for reduced bandwidth. Using our availability predictors, we can reduce both bandwidth and disk consumption. By identifying nodes that are highly available and biasing data storage towards them, we decrease on-demand object regeneration while reducing the need for over-replication.

Mahajan *et al* showed how a node in a peer-to-peer system can estimate the mean session time by observing the churn rate in its neighbor set [21]. Machines can then reduce the rate at which they send keep-alive messages while maintaining the same level of consistency in their routing tables, reducing control traffic. Knowing the mean session time of the network at a particular moment cannot be used to predict the availability of individual machines in a fine-grained manner.

Schwarz *et al* proposed a distributed object store which biases data storage towards peers with high predicted availability [30]. Each node has a counter which is initialized to 0. During a periodic system-wide scan, a node's counter is incremented by 1 if it is online, otherwise the counter is decremented by 1. Data storage is biased towards nodes with high counter values. Such a system will correctly identify consistently online nodes as good storage hosts and consistently offline hosts as poor replica sites. However, the counter mechanism cannot consistently capture phase relationships within and between nodes, e.g., the fact that if a host has diurnal availability, then it will be online for the longest consecutive stretch starting in the morning, when its counter is lowest. In Section 5.1, we describe a data store which handles this correctly.

3 Availability Predictors

In this section, we present our models for availability prediction. Each individual predictor is presented, followed by a mechanism to combine them to the best possible advantage.

3.1 Saturating Counter Predictors

Our first predictor is the *RightNow* predictor. A node's current online status is used as the value of all predictions for all lookahead periods. RightNow predictors are attractive because they require only one bit of state, and they should work well for nodes which are predominantly online or predominantly offline. Unfortunately, they cannot accurately capture periodic availability patterns over medium or long term time scales.

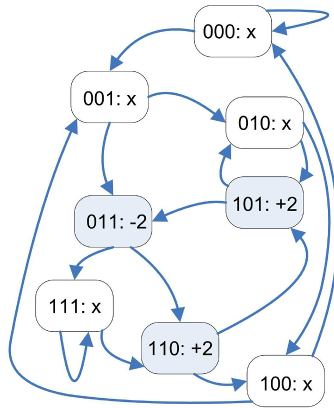
We can generalize the RightNow predictor to utilize n bits of state. For example, whereas the RightNow predictor represents uptime state using a single bit, the *SatCount-2* predictor uses a 2-bit saturating counter. Such a counter can assume four values (-2,-1,+1,and +2) which correspond to four uptime states (strongly offline, weakly offline, weakly online, and strongly online). During each sampling period, the counter is incremented if the node is online, otherwise it is decremented; increments and decrements cannot move the counter beyond the saturated counts of -2 and +2. Predictions for all lookahead periods use the current value of the saturating counter, i.e., negative counter values produce "offline" predictions, whereas positive values result in "online" predictions.

By using a few extra bits of storage, SatCount-x predictors are more tolerant than RightNow predictors to occasional deviations from long stretches of uniform uptime behavior. However, like the RightNow predictors, they are inaccurate for nodes with periodic uptimes.

3.2 State-Based Predictors

To predict the behavior of nodes with periodic availabilities, we turn to state-based predictors. These predictors explicitly represent a node's recent uptime states using a de Bruijn graph. A de Bruijn graph over k bits has a vertex for each binary number in $[0, 2^k\text{-}1]$. Each vertex with binary label $b_1 b_2 ... b_k$ has two outgoing edges, one to the node labeled $b_2 b_3 ... 0$ and the other to the node $b_2 b_3 ... 1$. In other words, the transition from a starting node to a child node represents a left shift of the parent's label and an addition of 0 or 1.

www.manaraa.com



This is the de Bruijn graph for the uptime pattern $\{110\}^*$. Each vertex is labeled as `uptime_state:sat_counter_value`. A counter value of `x` means that the associated vertex has never been visited. Traversing an edge represents a left shift-and-fill of the starting node label.

Figure 1: History predictor example

Suppose that we represent a node's recent availability as a k-bit binary string, with b_i equal to 0 if the node was offline during the ith most recent sampling period and 1 if it was online. A k-bit de Bruijn graph will represent each possible transition between availability states. To assist uptime predictions, we attach a 2-bit saturating counter to each vertex. These counters represent the likelihood of traversing a particular outbound edge; negative counter values bias towards the 0 path, whereas positive values bias towards the 1 path. After each uptime sampling, the counter for the vertex representing the previous uptime state is incremented or decremented according to whether the new uptime sample represented an "online" edge or an "offline" edge.

To make an uptime prediction for n time steps into the future, we trace the most likely path of n edges starting from the vertex representing the current uptime state. If the last bit we shift in is a 1, we predict the node will be online in n time units, otherwise we predict that it will be offline.

Figure 1 depicts the state maintained by such a *History* predictor. In this example, the node's availability has a periodicity of 3 samples and the repeated uptime string is 110. The node does not deviate from this pattern, so only the three shaded vertices represent observable uptime states.

The de Bruijn graphs used by the History predictor resemble the file access trees used by certain cache prefetching algorithms [14, 19]. However, these algorithms weigh each edge using raw access counts instead of saturating counters. CPU branch predictors [22] associate saturating counters with previously observed branch histories, but they do not use the superposition idea described below.

3.3 Tolerating Noise in the State Space

Suppose that a node has a fundamentally cyclical uptime pattern, but the pattern is "noisy." For example, a machine might be online 80% of the time between midnight and noon and always offline at other times. If the punctuated downtime between midnight and noon is randomly scattered, the de Bruijn graph will accumulate infrequently visited vertices whose labels contain mostly 1's but differ in a small number of bit positions. As the length of time that we observe the node grows, noisy downtime will generate increasingly more vertices whose labels are within a few bit-flips of each other. Probabilistically speaking, we should always predict that the node will be online from midnight to noon. However, the many vertices representing this time interval are infrequently visited and thus infrequently updated. Their counters may have weak saturations (-1 or +1) that poorly capture the underlying cyclic availability.

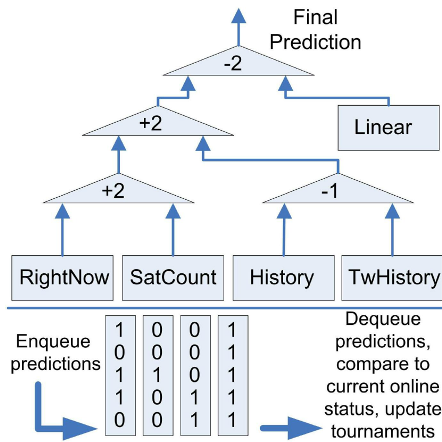
For nodes like this, we can nudge predictions towards the probabilistically favored ones by considering *superpositions* of multiple uptime states. Given a vertex v representing the current uptime history, we make a prediction by considering v's counter and the counters of all observed vertices whose labels differ from v's by at most d bits. For example, suppose that k=3 and d=1, and that each of the $2^k = 8$ possible vertices corresponds to an actually observed uptime history. To make a prediction for the next time step when the current vertex has the label 111, we average the counter values belonging to vertices 111, 110, 101, and 011. If the average is greater than 0, we predict "online," otherwise we predict "offline."

We call such a predictor a *TwiddledHistory* predictor, since it considers the current vertex and all "twiddled" vertices whose labels differ by up to d bits. The hope is that by averaging the counters of similar uptime states, we remove noise and discover stable underlying availability patterns. The TwiddledHistory strategy will perform worse than the regular History strategy when vertices within d bits of each other correspond to truly distinct uptime patterns. In these situations, superposition amalgamates state belonging to unrelated availability behavior, reducing prediction accuracy.

3.4 Linear Predictor

Linear prediction [15] is a common technique from digital signal processing and statistical time series analysis. It uses a linear combination of the last k signal points to predict future points. The k coefficients are chosen to reduce the magnitude of an error signal, which is assumed to be uncorrelated with the underlying "pure" signal. To make availability predictions for t time steps into the future, we iteratively evaluate the linear combination using the k most recent availability samples, shifting out the oldest data point and shifting in the predicted data point. Linear prediction produces good estimates for signals that are stable in the short term but oscillatory in the medium to long term [27]. We would expect this

www.manaraa.com



This figure depicts the tournament counters and update queue of a Hybrid predictor. The five bits in each queue entry represent the previous predictions of the five sub-predictors.

Figure 2: Hybrid predictor example

technique to work well with nodes having diurnal uptimes, e.g., machines that are online during the work day and offline otherwise.

3.5 Hybrid Predictor

A machine can transition between multiple availability patterns during its lifetime. Furthermore, some availability patterns are best modeled using different predictors for different lookahead periods. To dynamically select the best predictor for a given uptime pattern and lookahead interval, we employ a *Hybrid* predictor. Our approach is similar in spirit to the "mixture of experts" strategy of the Network Weather Service [35], but closer in design to hybrid branch predictors [22]. For each lookahead period of interest, the Hybrid predictor maintains tournament counters. These saturating counters determine the best predictor to use for that lookahead period. For example, Figure 2 depicts a three-level tournament. Negative counter values select the left input, whereas positive values select the right. In this example, the SatCount predictor is currently more accurate than the RightNow predictor. Similarly, the History strategy is outperforming the TwiddledHistory strategy. The best history-based approach is beating the best "simple" approach, and the Linear predictor performs worse than the best of the other four predictors. Thus, the final output of the Hybrid predictor is the History prediction.

Consider a Hybrid predictor making forecasts for an n-sample lookahead period. At the beginning of each time unit, the Hybrid predictor samples the current uptime state of its node. Its five sub-predictors are updated with this state, and each sub-predictor makes a prediction for n time units into the future. The final output of the Hybrid predictor is selected via tournaments as shown in Figure 2, and the individual sub-predictions are placed in a queue and timestamped with `curr_time + n`. If the head of the queue contains an entry whose timestamp matches the current time, the entry is dequeued and the tournament counters are updated using the dequeued predictions. A tournament counter remains unchanged if both of the relevant dequeued predictions match the current uptime state or both do not match. Otherwise, one prediction was right and the other was wrong, and the tournament counter is incremented or decremented appropriately. In the last stage of the update, the `curr_time` value is incremented.

A Hybrid predictor responsible for multiple lookahead periods keeps a separate update queue and tournament counter set for each period. Each queue and counter set is maintained using the algorithm described above.

4 Predicting Individual Node Availability

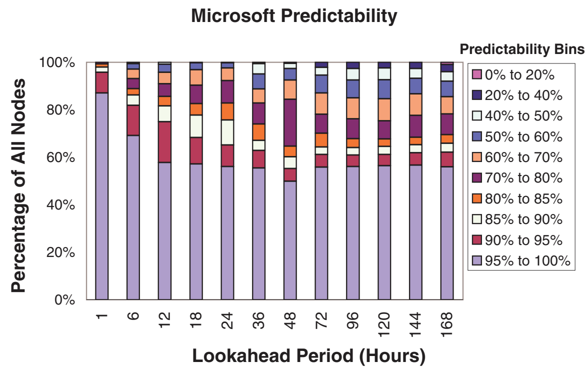
We first evaluate our predictors using the PlanetLab and Microsoft availability traces. Extending a Fourier transform technique [8], we create a taxonomy of uptime classes and explain the different availability behaviors of the two traces by examining their constituent uptime classes. We then test our predictors against the Overnet availability trace and show that our models do not fare as well. We provide evidence to suggest that this is not an artifact of our predictors, but instead the result of a fundamental unpredictability in the nodes themselves.

4.1 PlanetLab and Microsoft Nodes

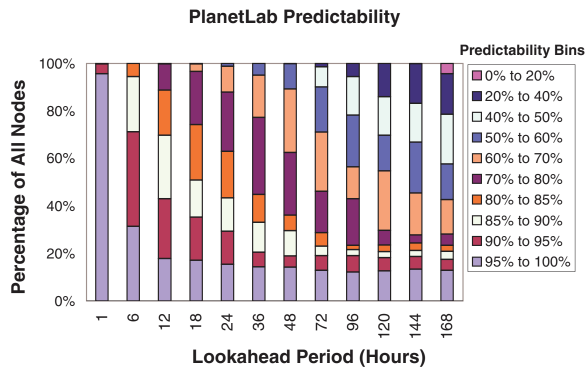
To determine the real-world applicability of our predictors, we tested them on two empirically gathered availability traces. The first trace followed 51,662 PCs in the Microsoft corporate network [6], and the second captured the behavior of 321 nodes in the PlanetLab distributed testbed [1]. Each machine in the Microsoft trace was pinged hourly. In the PlanetLab traces [33], machines were pinged every 15 minutes, but we sampled every fourth measurement to provide a fair comparison with the Microsoft data. The lifetimes of PlanetLab nodes were long enough that such sampling did not distort the underlying availability patterns. Our PlanetLab data spanned the five week period from July 1 to August 4, 2004. The Microsoft data spanned the five week period from July 6 to August 9, 1999.

For each availability trace, we associated a unique Hybrid predictor with each node. We used the first two weeks of a node's uptime data to train its predictor. Two weeks was a reasonable training length because it gave a predictor two chances to observe uptime patterns with a periodicity of a full week. After training the predictors, we evaluated their accuracy by comparing their predictions to the remaining three weeks of availability data. During each hour in the evaluation period, the predictors made forecasts for multiple lookahead intervals and were updated with uptime samples from that hour. We say that a node is p-predictable for a certain looka-

المنارة للاستشارات
www.manaraa.com



(a) A stable core of Microsoft nodes remains highly predictable across all lookahead periods.



(b) PlanetLab nodes start out highly predictable, but their predictability quickly degrades. Overall, these nodes are less predictable than the Microsoft set.

Figure 3: Microsoft and PlanetLab predictability

head period if we predicted its uptime behavior with at least accuracy p.

All Hybrid predictors used 3-bit saturating tournament counters. The organization of the tournaments resembled the structure shown in Figure 2. However, instead of a single Linear predictor, two Linear predictors competed with each other—one tracked 168 bits of state and the other tracked 336 bits. Also, the History and TwiddledHistory predictors were replaced with two two-level tournaments, allowing the same predictor type with different k values to compete. The single History predictor was replaced with a two-level tournament comparing k values of 6, 24, 48, and 56; the same k values competed in the TwiddledHistory multi-level tournament. In all experiments, the SatCount predictors used 2 bits of uptime state, and the TwiddledHistory predictors used a d of 1.

Figure 3 bins the predictability of individual Microsoft and PlanetLab nodes for several lookahead intervals. For a 1-hour lookahead period, 95.6% of PlanetLab nodes can be predicted with greater than 95% accuracy, as compared to only 87.0% of Microsoft nodes. However, as the lookahead period increases, the percentage of PlanetLab nodes that are 95%-predictable quickly drops below 20%. In contrast, slightly over half of the Microsoft nodes are 95%-predictable across all lookahead intervals.

Node Type	Microsoft	PlanetLab
Always On	60.66%	15.58%
Always Off	1.22%	5.60%
WW Periodic	9.79%	0.00%
Long Stretch	20.48%	67.60%
Unstable 70-90	2.05%	1.25%
Unstable 50-70	1.67%	1.56%
Unstable 10-50	4.12%	8.41%

Figure 4: Uptime Class Categorization

As the lookahead period increases, the predictability decay of the Microsoft nodes is more graceful than that of the PlanetLab nodes. For example, with a 144 hour lookahead period, 72.5% of the PlanetLab nodes have worse than 70% predictability; in the Microsoft data set, only 22.6% of nodes are this bad. In fact, for all of the studied lookahead periods, no more than a fourth of the Microsoft nodes are ever worse than 70%-predictable.

Why do the two data sets have different predictabilities? To answer this question, we must determine the distinct uptime classes contained in each set. We extract these classes by extending a technique proposed by Douceur [8]. Douceur wanted to identify nodes with diurnal uptime patterns, e.g., machines that were online from 9 AM to 5 PM during the work week. Douceur treated each node's uptime trace as a digital signal where bit n was a 1 if the node was online during hour n. He then applied a Fourier decomposition [15] to each signal, determining the set of sine waves whose sum equaled the original signal. Nodes with diurnal uptime patterns had spikes in the daily and weekly frequency spectra.

We can generalize this technique to detect multiple types of uptime regimes. To classify a node's availability behavior, we convert its uptime string into a digital signal and apply several tests to it. Once an uptime signal passes a test, we consider it categorized and we do not apply the remaining tests.

First, we classify a node as *always on* or *always off* if it's availability signal contains 90% ones or zeros, respectively. Second, nodes are subjected to Douceur's technique to detect diurnal periodicity. Nodes that pass this test are *work-week periodic*. Third, if the Fourier decomposition resembles the curve $1/f$, the node's uptime pattern is the summation of low frequency sine waves. This means that the node's online and offline stretches are long running, and we label these *long stretch nodes*. While it is possible that unstable nodes have spikes in frequency domains other than those explored, we have not found such spikes in our traces. Therefore, a node failing all of these tests is labeled *unstable*. We further bin unstable nodes according to the percentage of time that they are online. This creates the uptime classes *unstable70to90*, *unstable50to70*, and *unstable10to50*.

Figure 4 describes the constituent uptime classes in the PlanetLab and Microsoft traces. We see that PlanetLab is dominated by long stretch nodes. Long stretch nodes are highly predictable in the short term—given the current uptime

المنارة للاستشارات
www.manaraa.com

state of a long stretch node, we can confidently predict that it will remain in that state for the next few hours. However, long stretch nodes are increasingly unpredictable for larger lookahead intervals because their uptime regimes lack periodicity. Once such a node changes uptime state, it will keep that state for many hours, but the arrival of these changes are random. Thus, the predictability of the PlanetLab system as a whole degrades for long lookahead periods.

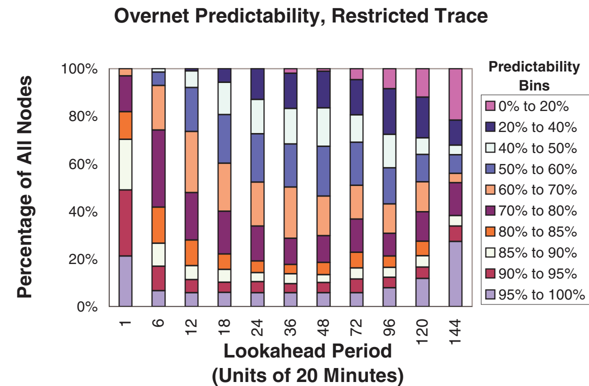
In the Microsoft data set, 61% of the machines are always on. These nodes represent the stable core which is 95%-predictable across all lookahead intervals. Whereas the PlanetLab system has no work-week periodic nodes, 9.79% of the Microsoft machines have these diurnal uptime patterns [1]. These machines are often highly predictable, although this is not always the case. For example, some machines are work-week periodic only to the extent that they are always offline during non-work day hours; during the actual work day, these machines have highly variable availability and thus are difficult to predict during these times. As another example, some work-week periodic machines are occasionally left online for multiple work days or left online during the weekend. Such aperiodic behavior is also difficult to forecast.

4.2 Predictability of Overnet Nodes

Microsoft and PlanetLab machines have fairly long session times; a node that has just come online will likely stay online for multiple consecutive hours. Nodes in other peer-to-peer networks like Gnutella or Napster have much higher churn rates [3, 29], typically on the order of tens of minutes. A natural question is whether these uptime patterns can be modeled using our prediction techniques.

To answer this question, we fed our predictors an availability trace of the Overnet DHT [3]. The trace covered 7 days with a sampling period of 20 minutes, providing 504 sample points. The trace contained 1,468 nodes that responded to at least one probe from January 15 to January 21, 2003. To evaluate the predictors on the Microsoft and PlanetLab data, we trained them for 2 weeks, which at a sample rate of once an hour resulted in 336 training samples and 504 evaluation samples. For a fair comparison, we also trained our Overnet predictors for 336 samples, leaving only 168 samples for evaluation purposes. We did not selectively pick hourly samples as we did for the PlanetLab traces because a probing granularity of 20 minutes is appropriate for a network with high churn rates. However, we did filter out nodes that were not online at least once in the first 100 samples and the last 100 sampling periods. This created a more challenging prediction

[1]Note that Douceur reported that 14% of Microsoft nodes have cyclical availability patterns [8], whereas we say that only 9.79% of them are work-week periodic. We report a lower percentage because we use a higher energy cutoff in the daily and weekly spectra for a node to classify as work-week periodic. From the perspective of evaluating our predictors, this more stringent cutoff is reasonable. For example, a node which is always offline except from noon to 2 PM during the work-week looks more like an always off node to our predictors. Thus, we categorize it as such.



Our availability predictions are less accurate for the Overnet trace than for the Microsoft and PlanetLab data sets.

Figure 5: Overnet predictability

environment, since many Overnet nodes were almost always offline and thus easy to predict. Bhagwan also estimated a non-trivial attrition rate of 32 hosts per day in the full node set [3]. Our paper is not concerned with long-term attrition effects, so the smaller trace set (haphazardly) filters out some of these "permanently" lost nodes.

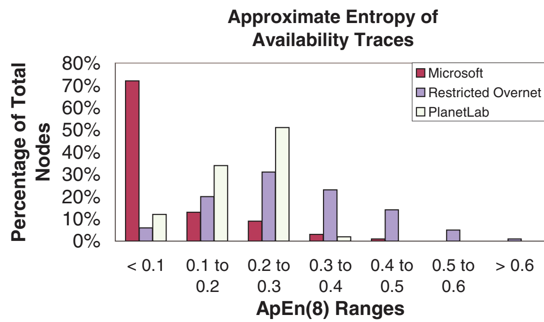
In the full Overnet trace, about a third of all nodes are 95% predictable for an arbitrary lookahead period; these are primarily nodes which are almost always off. As depicted in Figure 5, less than a tenth of the nodes in the restricted trace are 95%-predictable for an arbitrary lookahead period. In fact, the overall predictability of the restricted Overnet trace is much worse than that of the Microsoft or PlanetLab system. A natural question arises: what properties of the Overnet data set make it less predictable?

4.3 Entropy and Predictability

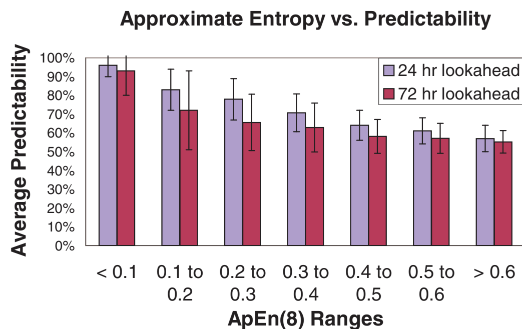
We must interpret the Overnet results from Section 4.2 with caution, since we have fewer evaluation samples than in the PlanetLab/Microsoft experiments and we cannot fully control for long term node attrition. We also do not have enough samples to confidently categorize nodes as work-week periodic, unstable, etc. However, manual inspection of the three uptime traces reveals qualitative differences in availability patterns. Overnet nodes appear to have more long stretch downtime than Microsoft or PlanetLab nodes, but the online stretches of Overnet nodes seem more randomly punctuated by bursts of downtime. This implies that Overnet nodes with non-trivial amounts of uptime should be more difficult to predict than Microsoft or PlanetLab nodes with similar uptime percentages.

To quantify this intuition, we use the information theoretic concept of approximate entropy [26], denoted `ApEn(x)`. Given an arbitrary length input string and an integer `m`, `ApEn(m)` represents the additional information provided by the last symbol of an `m`-character substring, given that we already know the first `m-1` characters. Approximate entropy is

www.manaraa.com



(a) This chart categorizes the approximate entropies of the availability strings in each trace set.



(b) This graph shows the relationship between the entropy of a Microsoft availability string and its predictability. Each y-bar represents a standard deviation.

Figure 6: Approximate Entropy Results

highest when all m-character substrings have equal frequencies. When `ApEn(m)` is low, we conclude that the string has repeated patterns and is non-random. As a simple example, consider the string $\{10\}^*$. Knowing the first bit of a two bit substring allows perfect prediction of the following bit. Thus, `ApEn(2)` is close to 0.

Figure 6(a) bins the approximate entropies of the uptime strings in the Microsoft, PlanetLab, and restricted Overnet traces. This figure validates our intuition that Overnet hosts have less regular availability patterns. The `ApEn(8)` values of the Microsoft and PlanetLab nodes are primarily smaller than 0.3, but 42% of Overnet nodes have an `ApEn(8)` greater than 0.3. Figure 6(b) plots `ApEn(8)` versus predictability for the Microsoft data set, confirming that higher entropy values are indeed correlated with lower predictability.

Note that the PlanetLab trace has higher entropy than the Microsoft trace. Referring to Figure 4, we see that the PlanetLab system has twice as many unstable10to50 nodes, which we would expect to be quite random. More importantly, PlanetLab is dominated by long stretch nodes instead of always on nodes. This should also increase system entropy, since long stretch nodes have less regularity than always on machines.

Machines with high uptime entropy may be difficult to predict individually, but a possible salvation may lie in the ability to identify nodes with correlated uptimes. Nodes displaying erratic behavior when considered singly may show emergent periodic behavior when considered in aggregate. This notion is supported by the Overnet trace, which shows diurnal periodicity at the global scale. By expanding the notion of superposition to include clusters of machines, we may be able to diminish the impact of entropy upon prediction accuracy for single nodes. We are currently investigating such methods.

4.4 Discussion

The Microsoft, PlanetLab, and Overnet traces were collected using a centralized probing infrastructure. For reasons of scalability or trust, such an architecture may be undesirable in some deployment scenarios. When we discuss our availability-aware applications in Section 5, we describe several mechanisms for decentralized dissemination of availability data. Dealing with malicious hosts seems to be a more difficult problem. It is not immediately obvious how a host can prove that its actual uptime history is equivalent to one gathered through centralized pinging or self-reporting. Developing threat models for availability-aware systems is an important area of future research.

Network partitions or outages may cause gaps in availability histories. For example, if histories are compiled via centralized probing, then probes may be dropped in a correlated, system-wide manner. Alternatively, if histories are self-maintained, network outages will hamper attempts to disseminate these records to peers. We are currently developing middleware to collect and distribute availability data in the face of such events.

The Microsoft and PlanetLab traces used pings to determine availability. If machines receive IP addresses in a non-static way, e.g. from DHCP or NAT, and the probing infrastructure is unaware of such dynamic assignments, then ping-based probing can lead to an overestimation of the number of hosts and an underestimation of host availability [3]. Fortunately, IP aliasing should be rare in both of these traces. The Microsoft study performed name lookups before each round of pinging so that availability strings could be assigned to specific machines instead of specific IP addresses. Furthermore, as stated in the instruction manual for PlanetLab administrators, the primary IP address for each PlanetLab node should be a static one.

Aliasing is not a problem if per-host operating system logs are used to infer availability [31]. Such an approach also allows one to measure availability exactly, as opposed to estimating it via sampling. However, this extra knowledge is not necessarily useful for availability prediction, since very brief downtime is generally "noise" and should be ignored. For example, most downtime due to software upgrades or rejuvenation rebooting should be aperiodic (making it difficult to predict) and fairly brief (so that it has little impact on a host's overall availability profile). Thus, such events should be omitted from the history that is used to make predictions, since they will only obscure essential availability trends. If

المنارة للاستشارات
www.manaraa.com

such events are substantial contributors to system downtime, then the sampling interval can be decreased. The sampling interval should also be selected with an eye towards typical session times. For example, session times are shorter in Overnet than in PlanetLab, so Overnet nodes should be sampled more than PlanetLab ones.

5 Applications of Availability Prediction

For our first application of availability prediction, we describe a distributed hash table which preferentially stores objects on highly available nodes; the modified DHT transmits fewer objects for regeneration and has greater data availability. We then show how availability predictors can improve routing performance in delay-tolerant networks. Finally, we integrate availability prediction with models of computer virus propagation and show how we can capture diurnal fluctuations in infection intensity. For each application, we use Hybrid predictors having the parameterizations and training times described in Section 4.1.

5.1 Availability-aware Replica Placement

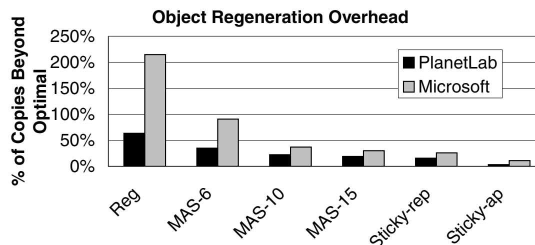
In a distributed data store, objects are replicated for reliability and availability. When a replica site goes offline, its objects typically must be copied from another machine and regenerated at a new site. By biasing data placement towards highly available nodes, we reduce the number of objects that must be shipped for regeneration. The bandwidth savings will be substantial if objects are large or there are many objects.

We frame our investigation of replication strategies within the context of the Chord routing infrastructure [32]. Each Chord node has a 160-bit overlay identifier, typically the hash of its IP address. The 2^{160} possible identifiers form a circular address space; the *successor* of a node is the first online machine with a larger identifier $\texttt{mod}\ 2^{160}$, and the *predecessor* is defined similarly. Each node tracks s immediate successors as well as several routing table peers. Through clever selection of routing table entries, nodes only need to maintain $O(logN)$ entries to provide $O(logN)$ route length.

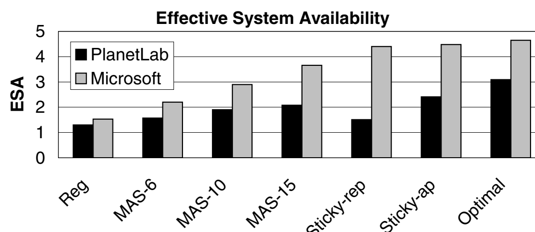
In a Chord-based DHT, an object is stored on the first node whose identifier is larger than the hash of the object. If replication is desired, the k replicas are stored on the first k nodes with larger identifiers [32]. The node immediately preceding the replica sites for an object is that object's *replica manager*. Queries for that object are routed to the replica manager, who responds to the initiator with the IP addresses of the replica sites.

We investigate five replication strategies. The first two do not use availability prediction. In the *regular* replication method described above, objects must be regenerated whenever a replica site leaves or a node join causes the first k successors of an object id to change. In the *sticky replica* strategy, a newly entering node N places replicas on its first k join-time successors. N continues to use a replica site until that site leaves the overlay, at which point it is replaced with the first successor of N that is not already a replica site for N. The sticky replica strategy requires fewer object copies than the standard scheme since only node leaves cause replicas to be transferred. Unfortunately, the overlay identifiers for an object's replica sites are no longer a simple function of that object's id. If a replica manager goes down unexpectedly, the pointers to its replica nodes are lost, and the associated data cannot be rediscovered in an efficient way [7]. To guard against this, each node backs up its replica site pointers on its first k successors. When a node leaves the overlay, its predecessor can find the replica sites for the new objects it manages and copy the necessary data to its replica sites.

The next two replication strategies use availability prediction to guide replica placement. Each node has a Hybrid predictor that it updates every hour. After each update, the node estimates the remaining number of hours that it will remain online, making iterative availability predictions for



(a) Availability-guided data placement reduces the copy overhead incurred when replica sites depart.



(b) By biasing object placement towards nodes which will be online for several consecutive hours, data availability also increases.

Figure 7: DHT simulation results

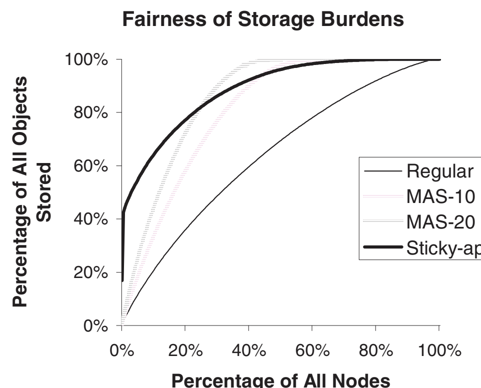


Figure 8: Storage skew in the Microsoft DHT

www.manaraa.com

1 hour into the future up to some maximum lookahead period. During each iteration, the estimate is incremented by 1 if the Hybrid predictor outputs “online” *and* the observed accuracy of predictions for that lookahead period surpasses a minimum threshold; if either condition is false, iteration terminates. Nodes periodically exchange their estimated availabilities with the other peers in their routing tables. These values are piggybacked atop standard routing stabilization messages [32]. In the simulation results given later, the maximum lookahead period was 15 hours and the minimum confidence level was 90%.

In the *most-available successors* replication strategy, abbreviated MAS-j, a replica manager places objects on the k most available of its first j successors, where $k \leq j \leq s$. A node’s replica site is sticky as long as it remains one of the first j successors. The *sticky replicas with availability prediction* scheme, abbreviated sticky-ap, features unconstrained attachment to replica sites as in the regular sticky strategy. Additionally, when a node picks a new replica site, it picks the most available of its immediate s successors that is not already a replica site.

For comparison purposes, we also study the *optimal* placement strategy. This strategy is like sticky-ap, but the availability predictor is an oracle. When a new replica site must be picked, the optimal scheme selects the node in the first s successors that will definitely be online for the longest consecutive period.

Figure 7 shows DHT performance when availability is driven by the Microsoft or the PlanetLab trace. The results were produced using a derivative of the well-known Chord simulator [32]. Leaves and joins that happened in the same hour in an availability trace were uniformly and randomly distributed across the corresponding hour in the simulation. All simulations ran for 504 virtual hours. The simulated Microsoft DHT contained 1000 nodes and the simulated PlanetLab one contained 321 nodes. 2% of DHT operations were writes and 98% were reads. In aggregate, the Microsoft DHT issued about 660 requests each minute according to a Poisson distribution. The PlanetLab DHT used the same per-node request rate, but due to its smaller size, it only issued about 210 aggregate requests per minute. The replication factor was 4 in both DHTs, and each node’s routing cache tracked 20 immediate successors. Each replication strategy was tested five times. The ith run for each strategy used the same set of nodes, but this set was changed for each value of i. Standard deviations were less than 3% amongst the trials for a particular replication strategy.

Figure 7(a) shows that availability-guided replication strategies result in many fewer copies due to replica regeneration. The savings are largest in the Microsoft DHT, with the regular replication strategy requiring 215% more copies than optimal and the sticky-ap strategy requiring only 11% more copies than optimal. The gains are smaller in the PlanetLab system, with the regular replication strategy requiring 65% more copies than optimal and the sticky-ap strategy requiring 3.4% copies beyond optimal. The discrepancy in savings is primarily explained by the fact that the Microsoft system had more nodes that were always online. Biasing data storage towards such nodes will lower overhead more than biasing objects towards long stretch nodes that will be online for several consecutive hours, but will still eventually go offline and require object copying. The Microsoft DHT also had work-week periodic nodes, unlike the PlanetLab one. Although work-week periodic nodes may often be offline, we can still take advantage of phase-shifted diurnal patterns to reduce object copying (see the example in Section 2).

We should distinguish between the savings derived from having sticky replica sites and the savings produced by clever choice of these sites. For example, in the Microsoft DHT, if we compare the sticky-rep strategy with sticky-ap, we see that sticky-ap required roughly 7.3 million copies, whereas the sticky replicas strategy required about 8.3 million copies. This reduction of a million copies can be understood as the savings from quickly identifying highly available nodes, as opposed to hoping that your first k successors are highly available and having to regenerate their replicas if you are wrong.

Figure 7(b) describes the effective system availability (ESA) of the two DHTs using various replication strategies. ESA expresses global object availability in units of “nines.” For example, if we expect an arbitrary object to be accessible via some replica site 99% of the time, the system-wide object availability is 0.99 or 2 nines of availability; more detailed discussion of ESA is provided elsewhere [9]. As expected, ESA goes up as the DHT has more freedom to bias object storage towards highly available nodes, and the improvement is greater in the Microsoft system. For example, in the Microsoft DHT, MAS-15 more than doubles the baseline ESA, adding 2.17 nines. In the PlanetLab DHT, MAS-15 improves ESA from 1.31 to 2.08.

If nodes use a replication strategy with unconstrained stickiness, a node may place objects on peers that “move beyond” its first s successors. In these scenarios, nodes will have to devote extra heartbeat messages to ensure that these replica sites are online. If objects are relatively small, then the relative cost of these heartbeat messages may justify the use of schemes such as MAS-10 which place replicas on peers which would already be pinged.

Also note that there is a tension between reducing object copies and maintaining equitable storage burdens. This tension is depicted in Figure 8, which shows the cumulative distribution of object storage with respect to the number of nodes in the Microsoft DHT. The line $y = x$ represents a perfectly equitable storage burden, i.e., $X\%$ of the total objects would be stored on exactly $X\%$ of the nodes. The regular replication strategy was close to this line, although it was slightly convex since real-life storage burdens will never be exactly uniform. In the standard replication scheme, 1.4% of nodes

www.manaraa.com

stored less than 100 objects per online hour, and 59.7% of nodes stored between 300 and 900 objects per online hour. The distributions for the availability-guided replication strategies were much less equitable. For example, with MAS-20, 57.0% of nodes stored less than 100 objects per online hour. The storage skew was even more dramatic for the sticky-ap scenario, where 10% of nodes stored 64% of the objects.

The system designer must balance competing requirements for high ESA, low bandwidth usage, and equitable storage burdens. The threat model must also be considered. If nodes are untrusted, it is unwise to bias too many objects towards a few nodes, since this multiplies an attacker's ability to corrupt the object store. Studying these trade-offs is an important area for future work.

5.2 Delay-tolerant networks

In the traditional wired Internet, machines often have the luxury of persistent, high quality connections to their peers. In contrast, *delay-tolerant networks* (DTNs) [13] are composed of heterogeneous devices with vastly differing networking and storage capabilities. Delay-tolerant networks must deliver messages in spite of intermittent device connectivity and differences in link bandwidth and latency that may span orders of magnitude. We provide two examples of DTNs later in this section.

The simplest DTN routing algorithm forwards each message along the first available link providing forward progress. Given the heterogeneous link qualities and intermittent connectivities that characterize a DTN, we would expect such a naive routing scheme to perform far worse than optimal. Jain *et al* describe how to use *resource oracles* to decrease message delivery times [16]. For example, they define a contact oracle which has perfect knowledge of link characteristics. Given two devices and an arbitrary point in the future, the contact oracle can output whether a link will exist between the two devices. Using our availability prediction techniques, we can approximate such an oracle and plug it into the routing algorithm described in [16].

We evaluate our contact oracle by simulating the behavior of two DTNs. The first one is reminiscent of an example provided by Jain *et al*. Imagine that a remote village without wired Internet access wishes to fetch web pages. Further suppose that the village is willing to tolerate asynchronous delivery latencies on the order of a day; such latencies are quite reasonable if the web pages are required to teach a class whose syllabus is known in advance. The remote village has a much larger sister city with a wired Internet connection, but this city is several hours away by ground transportation. Luckily, the buses that travel between the two cities can act as data mules, with outbound vehicles from the village carrying web requests and inbound vehicles carrying web data to be downloaded in the city. We assume that there are three round trips between the village and the city each day. The time required for each one way trip is chosen uniformly from the range [100 minutes, 140 minutes]. The first bus leaves the village at 8 AM, and the last bus is expected to return to the village at 8 PM. Each bus has a data capacity of 128 MB (equivalent to a small USB memory card), and each bus stays in network contact at the village or the city for five minutes. We assume that the bandwidth of the USB device is 1 Mbps.

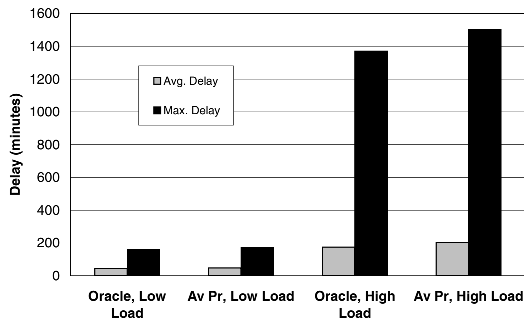
The village has two additional means of communication. First, the village and the city are periodically connected by a satellite link. The satellite is close enough to both locations to form a direct link every six hours, and the link persists for ten minutes. The satellite bandwidth is 10 kilobits per second and the latency is 3 seconds. Second, the village has access to a slow dial-up modem which, for reasons of expense, is only accessible from 11 PM to 6 AM. Due to an unreliable telephone infrastructure, this link is offline for 10% of its ostensibly available period, with the unexpected disconnections scattered uniformly between 11 PM and 6 AM.

Figure 9 shows simulation results for the DTN described above. Web requests were 1KB on average and web responses were 10KB on average, as suggested by empirical studies of web traffic [28]. Predictors were trained on two weeks of synthetic availability data with a sampling granularity of 20 minutes. Messages were then generated at randomly chosen times for 5 simulated days; simulation termination occurred once all messages had been delivered. Routing was reactive, i.e., when a message arrived at a node to be forwarded, the node used the most recently observed availability data to calculate the route for that message.

In the low load scenario of Figure 9, the village and the city exchanged 200 messages a day (i.e., 200 web requests were sent to the city and 200 web fetches were sent to the village). Using our availability predictors led to average message latencies that were only 6.7% worse than those incurred by optimal oracles. The worse-case delay was only 7.5% worse. In the high load scenario, the village and the city exchanged 1000 messages a day. Uptime mispredictions resulted in greater penalties in this scenario, since a single poor prediction could result in many messages being routed through an erratically online node. However, average message delays in our predictor system were still within 16% of those in a system with infallible contact oracles. Worse case delays were within 9.6% of optimal.

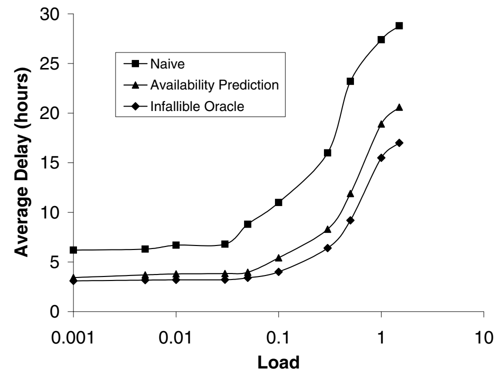
Our second evaluation DTN represents a collaborative sensor node system. We suppose that each user in the system possesses a laptop, a desktop PC, and a set of trusted laptops and desktops belonging to friends. When a user moves to a new location and begins to work on her laptop, the laptop may "notice" something interesting about the surrounding environment, e.g., the availability of a new wireless access point. The user wants to share this information in an effort-free (and secure) method with her friends. Thus, her laptop acts as a store-and-forward node for the interesting piece of information. If the laptop is online when a friend's device is online, the laptop can directly transmit the information to the friendly

www.manaraa.com



Using our availability predictors as DTN contact predictors results in message delays that are close to those generated by an infallible contact oracle. These results represent the outcomes of twenty simulation runs.

Figure 9: Message delays in the village DTN.



In low to medium load situations, our availability prediction scheme is within 15% of optimal. The performance gap widens for loads greater than 0.6, with our scheme 40% worse than optimal under a load of 1.5. However, we still perform much better than a naive routing scheme which simply waits for the sender and the receiver to come online at the same time.

Figure 10: Message delays in a collaborative sensor DTN.

device. Otherwise, the laptop tries to forward the data across a path of trusted machines.

In our simulation of this DTN, each PC's uptime was driven by a trace from the Microsoft corporate network. Each laptop's availability was driven by a trace from Kim *et al*'s wireless availability study [18]. We filtered out laptops and PCs which were not online at least once during the first 20% and the last 20% of their respective trace period. Laptop availability was sampled every 30 minutes and PC availability was sampled every hour. Each user sent messages to a trusted collection of ten laptops and ten PCs. Message sizes varied uniformly between 1KB and 20KB, and messages were randomly generated by each laptop using a Poisson distribution with a λ of 0.5 messages per online hour. When a laptop generated a message, it first delivered it to all trusted nodes which were also online. If the laptop predicted that an offline friend would come online before it left the network, it would wait to deliver the data directly. Otherwise, it would instruct one or more of its online buddies to forward the data to the remaining friendly devices. If, for a particular message, several destination devices shared a common next hop from the current machine, only one copy of the data was forwarded to the next hop. Laptops communicated with desktops using 11 Mbps wireless links, and desktops communicated with each other using 100 Mbps Ethernet connections. We assumed that a path (i.e., link) existed between two machines if they were online at the same time, and all routing was reactive.

Figure 10 shows simulation results for the collaborative sensor DTN. Each data point represents the average of 20 trials, and during each trial, the DTN was comprised of a random subset of 300 laptops and 300 PCs from the respective availability traces. Each trial ran for 504 simulated hours, and predictors were pre-trained on 336 hours of data. We used Jain *et al*'s definition of load [16], such that the load over the duration of a simulation was the sum of the traffic demand divided by the sum of the available transmission bandwidth during this time. Note that some of this bandwidth would lie idle if a node had nothing to transmit at a particular moment.

As in the village DTN, when loads became high, the delay differences between the availability prediction system and the infallible oracle system grew. This difference was at worst 40% for a load of 1.5, but was closer to 10-15% for more reasonable loads. Additionally, our availability prediction system always had better performance than a naive scheme in which nodes never forwarded data using intermediate nodes and always waited for the destination machine to come online. Thus, we believe that our availability predictors can provide a meaningful improvement in DTN performance.

DTNs are a fairly new idea, so there is no consensus on the best way to maintain distributed DTN routing state. One could imagine that hosts with the necessary computational resources engage in a BGP-like protocol [12] to exchange link state. Each routing table entry would contain the next hop to a particular destination and the predicted availability profiles of nodes along the route. Each device would track its own availability history, but computationally weak nodes would push the tasks of uptime prediction and route selection to more powerful ones.

5.3 Virus Modeling

Traditional analytic models of computer virus propagation [17, 25, 34] assume that machines are always online. This assumption is often incorrect—the non-trivial churn rates found in real distributed systems result in network topologies with rich time-sensitive dynamics. At any given moment, some infected machines are offline (and thus effectively non-contagious), and some susceptible machines are offline (and therefore temporarily protected from infection). In such a fluid topology, the infection rate is no longer a

المنارة للاستشارات
www.manaraa.com

simple function of the virulence of the malicious code and the time it takes to discover an infected node and install the relevant software patch. Now, we must incorporate a time-varying availability function which describes node churn. Such time-dependent availability diminishes the aggressiveness of viral propagation, since infected hosts will be unable to spread the virus when they are offline. However, we cannot determine that a diseased node is sick until it comes online. Thus, the availability dynamic also impedes the discovery of infected hosts and subsequent application of the "cure." The net result is a quantitative and qualitative impact on the infection dynamic, an impact which has been observed in the real world. For example, an empirical study of the Code-Red worm discovered strong diurnal patterns in viral behavior, with the number of active diseased hosts spiking at the start of the workday and ebbing as some people applied patches and many people turned off their workplace computers and left for their homes [23].

As a first step towards more expressive viral models, we have derived an availability-aware version of the Kephart-White framework [17]. The classic Kephart-White model uses a differential equation to describe computer virus propagation. It assumes a susceptible-infected-susceptible (SIS) environment—a machine enters the system in a healthy state, and it can catch and subsequently be cured of the infection an infinite number of times [2]. The Kephart-White model assumes a homogeneous network topology in which all nodes have similar levels of connectivity or "out-degree." In such a network, the fraction f of infected nodes is given by:

$$\frac{df}{dt} = \beta\langle k\rangle f(1-f) - \delta f \quad (1)$$

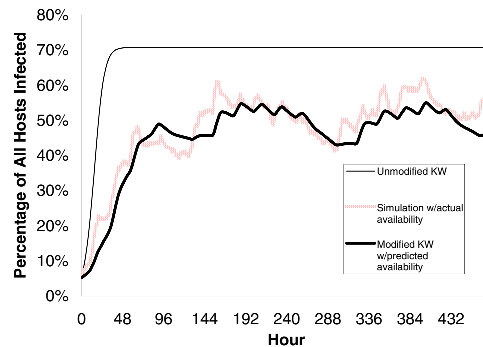
where t is time, β is the viral birth rate along every edge from an infected node, δ is the cure rate at each infected node, and $\langle k\rangle$ is the average connectivity of a node. β, δ, and $\langle k\rangle$ are assumed to be constant.

To represent the notion of machine availability, we define a time-varying *activity percentage*, denoted a. Just like f, a assumes values in the range [0.0, 1.0]. At time t, we let $a(t)$ represent the fraction of all machines in the distributed system which are currently online. This results in the following differential equation:

$$\frac{df}{dt} = \beta\langle k\rangle (fa)[(1-f)a] - \delta fa. \quad (2)$$

With respect to the standard Kephart-White equation, we replaced the infected fraction f with fa and the susceptible fraction $(1-f)$ with $(1-f)a$. These new quantities represent the fact that nodes must be active to transmit or receive the virus.

[2] The simple SIS model has no conception of permanent immunity, so it only crudely models the deployment of remedies like software patches. We use the SIS model here for pedagogical clarity, but it is straightforward to incorporate availability-awareness into more realistic models such as susceptible-infected-removed.



Accuracy of viral models for the Microsoft corporate network (β=0.008, δ=0.07,$\langle k\rangle$=30). The forecasts of our new model are much closer to the results of discrete time simulation driven by real availability data. For the availability-aware results, the two humps after hour 144 represent peaks in infection activity during two Monday to Friday work weeks. The lookahead period for predicted availability was 24 hours.

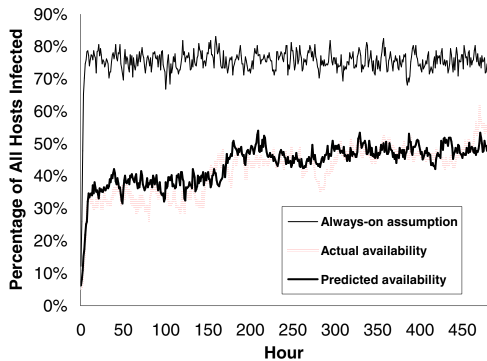
Figure 11: Epidemics in homogeneous topologies

Figure 11 shows that fluctuating node availabilities have a marked impact upon infection dynamics. Given β=0.008, δ=0.07, and $\langle k\rangle$=30, the standard Kephart-White model predicts a steady state infection fraction of 70%. However, if we remove the erroneous assumption of constant network connectivity and use real machine availabilities, viral propagation changes in two ways. First, there are cyclical fluctuations in the number of infected nodes corresponding to the diurnal work-week patterns in the underlying availability trace. Second, the average steady state infected percentage is depressed, relative to an environment in which nodes are always on.

What explains these new phenomena? Remember that a node can be cured or infected only if it is online. Given that a node is online, the probability of being cured is proportional to the constant δ, whereas the probability of being infected is proportional to the constant β and $\langle k\rangle fa$, the number of infected neighbors that are currently online. The likelihood of being cured is unrelated to the availability of its neighbors. However, its chances of being infected will diminish if its neighbors ever go offline. Thus, the unavailability of machines effectively strengthens the cure "force." This strengthening is dependent on the rate at which machines enter and leave the network. Since this rate has diurnal fluctuations, a diurnal infection dynamic emerges.

The Kephart-White model assumes that each node has the same number of neighbors. Such assumptions of connectivity homogeneity are reasonable when analyzing malicious code that spreads indiscriminately, e.g., via random IP scanning. The homogeneity assumption may be unwarranted for viruses which spread via application-level vectors that are governed by social relationships. For example, email contact graphs have a power-law connectivity distribution, meaning that most people have few contacts and a small number of people have many contacts [11]. In these situations, epidemi-

www.manaraa.com



Viral simulation results for a 321 node power-law topology and the PlanetLab availability trace (β=0.2,δ=0.24). The average node connectivity was 5.89, with a minimum degree of 3 and a maximum of 56. The lookahead period for predicted availability was 24 hours.

Figure 12: Epidemics in power-law topologies

ological models which assume contact homogeneity will have poor predictive power.

Adding availability prediction to an analytic model for power-law epidemics is an important area of unfinished work. However, we can already use simulation-based approaches to discern the impact of fluctuating uptimes upon the viral dynamic. Figure 12 shows simulation results for a 321 node power-law network with uptimes driven by the PlanetLab trace. Although global availability in the PlanetLab system showed no diurnal periodicity, more than 40% of all nodes were offline at any given moment. This should lead to a large depression in the infection level that would have resulted if machines were always online. Simulations using predicted availability captured this phenomenon well and were quantitatively similar to simulations that used actual (oracle) availability. This accuracy was achieved despite the fact that, as Figure 3(b) shows, only 17% of PlanetLab nodes are 95%-predictable for a 24 hour lookahead. These results imply that availability prediction can still provide useful benefits in environments that are relatively unpredictable in the medium-to-long term.

When availability data is used to assist antiviral efforts, the dissemination mechanism for this data will depend on the configuration of the antivirus system. For example, enterprise-level antiviral systems often use a small number of servers to receive new virus definitions. These centralized servers push new definitions to clients whenever they please, and they can force clients to scan for malware. Since end-users cannot stop these forced scans, the centralized servers have complete control over enterprise-wide antiviral policy. In such a scenario, the servers can also act as the global repository for availability data. They may ping clients directly, aggregate client-reported availability data, and/or infer availability through inspection of DHCP requests, ARP traffic, etc. Using this data to predict future availability, the servers can then prioritize patch distribution. For example, patches should preferentially be pushed to clients which are always on, since these are the machines which, if infected, will have the most opportunities to infect other hosts. If the system spans time zones, then one might want to create a "time zone firewall" by preferentially patching work-week periodic hosts in time zones where the work day is about to begin.

6 Conclusion

Loosely-coupled distributed systems are comprised of nodes that can join and leave the collective at any time. Previous models of peer-to-peer availability [2, 5, 10] provide conservative estimates of uptime, but these models cannot predict changes in availability over time. To achieve true insights into the behavior of peer-to-peer systems, availability must be a first class concern.

In this paper, we introduce new techniques for availability prediction. Our predictors track fine-grained, per-node uptime state to estimate future availability, leveraging the most accurate estimation mechanism for each situation. To quantitatively characterize differences in availability between multiple distributed systems, we use techniques from signal analysis and information theory to create uptime taxonomies.

We describe three useful applications of availability prediction. By biasing replica storage towards highly available nodes, we can reduce network bandwidth consumption and increase data availability. Using our uptime predictors as contact oracles, we can reduce message latencies in delay-tolerant networks. Finally, by incorporating availability into epidemiological models, we can capture empirically observed infection dynamics.

Acknowledgments

We thank the NSDI reviewers, and especially Doug Tygar, our shepherd, for their many helpful comments and suggestions. This work was supported in part by the National Science Foundation under Grant No. CNS-0509089. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale services. In *Proceedings of NSDI*, San Francisco, CA, March 2004.

[2] R. Bhagwan, S. Savage, and G. Voelker. Replication strategies for highly available peer-to-peer systems. Technical Report CS2002-0726, UCSD, November 2002.

[3] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proceedings of the 2nd IPTPS*, Berkeley, CA, February 2003.

[4] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker. Total Recall: system support for automated availability management. In *Proceedings of NSDI*, March 2004.

المنارة للاستشارات
www.manaraa.com

[5] C. Blake and R. Rodrigues. High availability, scalable storage, dynamic peer networks: pick two. In *Proceedings of the 9th HotOS*, May 2003.

[6] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proceedings of ACM SIGMETRICS*, Santa Clara, CA, June 2000.

[7] K. Chen, L. Naamani, and K.Yehia. miChord: decoupling object lookup from placement in DHT-Based Overlays. http://www.loai-naamani.com/Academics/miChord.htm.

[8] J. Douceur. Is remote host availability governed by a universal law? *SIGMETRICS Performance Evaluation Review*, 31(3):25–29, 2003.

[9] J. Douceur and R. Wattenhofer. Large-scale simulation of replica placement algorithms for a serverless distributed file system. In *Proceedings of the 9th MASCOTS*, pages 311–319, Cincinnati, Ohio, August 2001.

[10] J. Douceur and R. Wattenhofer. Optimizing file availability in a secure serverless distributed file system. In *Proceedings of the 20th IEEE SRDS*, pages 4–13, New Orleans, LA, October 2001.

[11] P. Drineas, M. Krishnamoorthy, M. Sofka, and B. Yener. Studying e-mail graphs for intelligence monitoring and analysis in the absence of semantic information. In *Proceedings of the Symposium on Intelligence and Security Informatics*, Tuscon, AZ, June 2004.

[12] B. Duvall. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006.

[13] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of ACM SIGCOMM*, pages 27–34, August 2003.

[14] J. Griffioen and R. Appleton. Performance measurements of automatic prefetching. In *IEEE Parallel and Distributed Computing Systems*, pages 165–170, September 1995.

[15] S. Haykin. *Adaptive filter theory*. Prentice Hall, Englewood Cliffs, NJ, 3rd edition, 1996.

[16] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *Proceedings of ACM SIGCOMM*, pages 145–158, September 2004.

[17] J. Kephart and S. White. Directed-graph epidemiological models of computer viruses. In *Proceedings of the IEEE Computer Symposium on Research in Security and Privacy*, pages 343–359, May 1991.

[18] M. Kim and D. Kotz. Modeling users' mobility among WiFi access points. In *Proceedings of the International Workshop on Wireless Traffic Measurements and Modeling*, pages 19–24, Seattle, WA, June 2005.

[19] T. Kroeger and D. Long. The case for efficient file access pattern modeling. In *Proceedings of the 7th HotOS*, Rio Rico, AZ, March 1999.

[20] D. Long, A. Muir, and R. Golding. A longitudinal survey of internet host reliability. In *Proceedings of the 14th IEEE SRDS*, 1995.

[21] R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *Proceedings of the 2nd IPTPS*, Berkeley, CA, February 2003.

[22] S. McFarling. Combining branch predictors. Technical Note TN-36, DEC WRL, June 1993.

[23] D. Moore, C. Shannon, and J. Brown. Code-red: a case study on the spread and victims of an Internet worm. In *Proceedings of the Second Internet Measurement Workshop*, pages 273–284, November 2002.

[24] D. Nurmi, J. Brevik, and R. Wolski. Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments. In *Proceedings of EUROPAR*, 2005.

[25] R. Pastor-Satorras and A. Vespignani. Epidemic Spreading in Scale-Free Networks. *Physics Review Letters*, 86(14):3200–3203, April 2001.

[26] S. Pincus. Approximate entropy as a measure of system complexity. In *Proceedings of the National Academy of Science*, pages 2297–2301, USA, March 1991.

[27] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical recipes in C*. Cambridge University Press, New York, NY, 2nd edition, 1992.

[28] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An analysis of internet content delivery systems. In *Proceedings of OSDI*, pages 315–327, December 2002.

[29] S. Saroiu, P. Gummadi, and S. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of the Multimedia Computing and Networking Conference*, pages 18–25, January 2002.

[30] T. Schwarz, Q. Xin, and E. Miller. Availability in global peer-to-peer storage systems. In *Proceedings of the 2nd IPTPS*, Lausanne, Switzerland, July 2004.

[31] C. Simache and M. Kaaniche. Measurement-based Availability Analysis of Unix Systems in a Distributed Environment. In *Proceedings of the International Symposium on Software Reliability Engineering*, pages 346–355, Hong Kong, China, November 2001.

[32] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalabale peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, San Diego, CA, August 2001.

[33] J. Stribling. All-pairs PlanetLab Ping Data. http://www.pdos.lcs.mit.edu/~strib/pl_app/.

[34] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos. Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint. In *Proceedings of the Symposium on Reliable Distributed Computing*, pages 25–34, Florence, Italy, October 2003.

[35] Rich Wolski. Experiences with predicting resource performance on-line in computational grid settings. *SIGMETRICS Performance Evaluation Review*, 30(4):41–49, 2003.

المنارة للاستشارات
www.manaraa.com