End-to-end WAN Service Availability*

Bharat Chandra, Mike Dahlin, Lei Gao, and Amol Nayate
University of Texas at Austin

DRAFT: 9/20/2000 – Please check
http://www.cs.utexas.edu/users/dahlin/papers.html
for the most current version.

Abstract

This paper seeks to understand how network failures affect the availability of service delivery across wide area networks and to evaluate classes of techniques for improving end-to-end service availability. Using several large-scale connectivity traces, we develop a model of failures that includes key parameters such as failure location and failure duration. We then evaluate several classes of techniques for coping with network failures using trace-based simulation. We find that caching alone is seldom effective at insulating services from failures but that the combination of mobile extension code and prefetching can improve failure rates by as much as an order of magnitude for classes of service whose semantics support disconnected operation. We find that routing-based techniques may provide significant improvements, but that the improvements of many individual techniques is limited because they do not address all significant classes of network failures.

1 Introduction

This paper seeks to understand how network failures affect the availability of service delivery across wide area networks (WANs) and to evaluate classes of techniques for improving end-to-end service availability. By providing a quantitative analysis of these techniques, we hope to provide a framework to help service designers select from and make best use of currently-available techniques. Further, we seek to evaluate the potential impact on availability from proposed extensions to the Internet infrastructure such as replication of active objects [1, 2, 10, 23, 24] and overlay routing [20].

Although several commercial hosting services today advertise 99.99% or 99.999% ("four 9's" or "five 9's") server availability, providing highly available servers is not sufficient for providing a highly available service because it is not an end-to-end approach: other types of failures can prevent users from accessing services. Internet connectivity failures, unfortunately, are not rare. Paxson [18], for example finds that "significant routing pathologies" prevent selected pairs of hosts from communicating about 1.5% to 3.3% of the time, and more recent measurements [29] suggest that availability has not significantly improved since then. In contrast with the 5 minutes per year of unavailability for a five-9's system, a typical two-9's Internet-delivered service will be unavailable for nearly 15 minutes per day from a typical client. Our analysis of measurements from HTTP clients to servers confirms the magnitude of this problem.

Although caching can improve file system availability [9, 11], there is reason to be concerned that caching alone will not significantly improve WAN service availability because much HTTP traffic is uncachable [5, 28]. This limitation motivates us to study the potential effectiveness of other techniques such as hoarding [11], push-based content distribution [6], relaxed consistency, mobile extensions to ship service code to proxies or clients [1, 2, 10, 23, 24] connecting clients and servers to the Internet via multiple ISPs, and overlay routing [20]. Although the performance benefits of many of these techniques have been studied, their potential impact on end-to-end availability has not been quantified.

Our analysis faces two challenges. First, we wish to evaluate the potential effectiveness of a wide range of techniques for a wide range of services. To do this,

*This work was supported in part by an NSF CISE grant (CDA-9624082), the Texas Advanced Technology Program, the Texas Advanced Research Program, and grants from Dell, IBM, Novell, Sun, and Tivoli. Dahlin was also supported by an NSF CAREER award (CCR-9733842) and an Alfred P. Sloan Research Fellowship.

we abstract away both the detailed design of the techniques and the semantic requirements of the services. We can then determine upper bounds on improvements that different classes of techniques can yield. Then we explore the sensitivity of the techniques to parameters representing factors that could limit their effectiveness. The second challenge is that available studies of WAN failure patterns do not address important parameters for evaluating these techniques. To address this, we analyze several large connectivity traces to develop a model suitable for evaluating techniques for coping with failures.

This work makes three contributions. First, we develop a WAN connectivity failure model that includes failure rate, failure location, and failure duration. A key finding is that failure duration distributions appear heavy-tailed, which means that long failures account for a significant fraction of failure durations. Second, we conclude that data-caching-based techniques for improving service availability will likely have little success, but that the combination of prefetching and shipping mobile extension code to clients and proxies has the potential to improve failure rates by over an order of magnitude. Unfortunately, three factors may significantly limit these gains: (i) compulsory misses to extension code and state, (ii) capacity misses due to limitations in the number of extensions a client or proxy can host, and (iii) service-specific semantic requirements that prevent some services from using these techniques. Finally, we find that routing-based approaches can significantly improve failure rates, but that near-client, near-server, and interior network failures all contribute significantly to the overall failure rates, which limits end-to-end improvements from efforts that address only one type of problem (e.g., multi-hosting a server with multiple ISPs).

The rest of this paper proceeds as follows. We first discuss related work in the area of modeling Internet failure patterns and coping with network failures. We then describe the network failure model we have developed. Section 4 evaluates classes of techniques for coping with network failures when delivering Internet services. Finally, Section 5 summarizes our conclusions.

2 Related work

The basic techniques we examine for improving robustness have been studied in other contexts. In file systems, caching, hoarding, and relaxed consistency can isolate clients from network and server failures [9, 11, 21]. Odyssey [16] has explored how to use application-specific adaptation to cope with disconnection and changing resource availability by dynamically adjusting service semantics.

In the context of web services, previous studies have examined the performance benefits of caching [5, 22, 27], prefetching [4, 17, 12, 19], pushing updates [14, 22], push-based content distribution [6], server replication [15], mobile code [1, 2, 10, 23, 24], and overlay routing [20], but the impact on end-to-end service availability of these techniques has not been systematically quantified.

Several systems implementing variations of some of these techniques have been built. The Netscape Navigator browser supports "off-line" browsing from its cache and and the Microsoft Internet Explorer Browser supports hoarding. Joseph et al.'s Rover toolkit [10] is designed to support disconnected operation for mobile clients accessing services. But these techniques have not been systematically applied to or evaluated for large numbers of services.

Paxson studies IP-level routing pathologies and finds that "major routing pathologies" thwart IP routing between a given pair of hosts 1.5% to 3.4% of the time [18]. This study focuses on quantifying the prevailance and diagnosing the causes of IP-level failures. Our analysis builds on this study by studying these traces to determine metrics relevant to end-to-end service delivery: failure location and failure duration.

Labovitz et al. [13] examine route availability by studying routing table update logs. They find that only 25% to 35% of routes had availability higher than 99.99% and that 10% of routes were available less than 95% of the time. They find that 60% of failures are repaired in a half hour or less, and that the remaining failures exhibit a heavy-tailed distribution. These results are qualitatively consistent with our end-to-end analysis and provide additional evidence that connectivity failures may significantly reduce WAN service availability.

Zhang et al. [29] study NIMI and traceroute measurements taken during December 1999 and January 2000. They find that routing reliability has neither degraded nor improved significantly since Paxson's 1995 study. The focus of this study is on stationarity of network behavior, and it finds considerable variation in behavior at different network locations, at different times, and on different time scales. This points to a potential limitation of our current study, which uses average behavior across paths and over time to develop a model of average availability. Given the variability found by Zhang et al. for the metrics of route stability, packet loss, and throughput, future work to study the variability of connectivity would be valuable.

3 Network failure model

We seek to model parameters of network failures that most directly affect availability and techniques to improve availability. The most basic parameter is failure rate: what fraction of time are two nodes unable to communicate? We then analyze additional details about failure patterns along two dimensions: failure location and failure duration.

Failure duration is important because it influences the effectiveness of techniques for coping with failures. For example, it may be simpler to use caching or prefetching to mask short failures than long failures since masking long failures requires predicting access patterns across longer periods of time, transferring more data to the cache, and storing more data in the cache.

Failure location is important because it influences the effectiveness of routing-based strategies. We use a simple model that classifies failures into three operationally significant categories – "near-source," "in-middle," and "near-destination." Near-source failures represent failures of the client stub network that disconnect a source machine or source subnet from the rest of the Internet. Near-destination failures have a similar effect on destinations. In-middle failures represent a connectivity failure in the middle of the network that prevent a pair of nodes from contacting one another (on the default route), but where both nodes are able to contact a significant fraction of the remaining nodes on the Internet. We also user the term "stub failures" to refer to the combined near-source and near-destination categories.

This location model is admittedly simplistic. Most notably, it represents in-middle failures as the interruption of connectivity between a single pair of nodes that does not affect any other pairs' ability to communicate. In reality failures in the middle of the Internet infrastructure will typically affect more than one pair of nodes [13]. Thus, groups of such middle failures are likely to be correlated, and the correlation will depend on details of network topology that would be complex to model. However, we believe that our simple model provides a reasonable first-order approximation for evaluating routing based techniques: an in-middle failure represents the case where both the source and destination can connect to a nontrivial fraction of the Internet but cannot connect to each other by the default route. Assuming that the core Internet is not partitioned, routing-based techniques are likely to be able to find an alternate route between the nodes in such a situation.

Parameter	Default value	Comment
Rate	1.5% (all failures) 1.25% (> 30s)	Varies from .4% to 7.4% in different data sets
Location	Src: 25% Mid: 50% Dest: 25%	All locations significant. Ratio varies widely across traces.
Duration	avg = 609 sec. pdf(x) = $16x^{-1.85}$	Appears heavy-tailed
Interarrival	avg = 48111 sec.	

Table 1: Default parameters for failure model.

Our failure model uses a simple model for interarrival times that omits details. Given a failure rate (expressed as a fraction of time a particular class of failures occurs at a particular location) and an average failure duration, we calculate the average interarrival time for each class of failures. We then assume that failures arrive independently with exponentially distributed interarrival times with the given average arrival rate. We leave as future work extending the model to (a) model correlation in time of arrivals to a location, (b) model correlation of arrivals across different locations, (c) model time-of-day dependencies, and (d) model different failure patterns on different paths [29].

Another enhancement to the model left as future work is modeling quality of service. Whereas our simple model tracks periods of complete disconnection, for some applications, the network has "failed" if the bandwidth falls below a certain level or the latency rises above some level. A more sophisticated failure model might account for variations in quality of service as well as the coarse metric of connectivity on which we focus.

Table 1 summarizes key parameters for our model. The following subsections describe how these parameters are obtained.

3.1 Failure analysis methodology

Our basic methodology for quantifying failure patterns uses datasets that consist of large number of attempts by pairs of nodes to communicate. We identify attempts that succeed and those that fail and use this information to develop models for the frequency, duration, and location of failures in the Internet.

We use two types of dataset. First, traceroute datasets consist of multiple traceroute measurements between pairs of nodes participating in the study. Second, HTTP datasets consist of logs of HTTP requests through public Squid [25] proxies to web servers. The datasets used are described in more detail in the next subsection.

There are potential biases in our approach resulting from both limitations of our data sets and our analysis of them.

First, the hosts and network paths that we trace may not be representative of typical Internet connectivity. Several of our traceroute datasets were collected by Paxson, and he argues that the interior nodes measured may be representative of typical routes but that the end-hosts may not be [18]. Other traceroute datasets were gathered by Savage et. al [20] from sites selected for convenience. Although our HTTP traces are sent to a collection of servers dominated by publicly-available HTTP servers, requests are sent from regional Squid proxies. These Squid proxies may be unusual sources both in terms of their network connectivity and in terms of the user community they serve.

Although we seek to develop end-to-end failure models, our data sets are not, strictly speaking, end-to-end. In particular, the traceroute data sets track failures at the IP level but omit higher-level protocol failures such as DNS failures. Although the HTTP data sets do include DNS failures, the logs do not include enough information to distinguish them from certain other application-level errors that can occur when end-to-end network connectivity is present; thus, we also omit DNS failures from the HTTP analysis. Finally, because traceroute servers' failure patterns may not be representative of those of content servers, we filter out "end-host" failures from the traceroute data sets. These factors means that we may underestimate end-to-end failure rates.

Also, our data sets may under-report the number of failures that happen near the source node of a request. In both the HTTP and traceroute data sets, network disruptions near the intended source of a measurement may prevent requests from being issued during these periods when they are more likely than average to fail. We compensate by focusing on the near-destination failures as representative of stub failures.

Another source of bias is the data sampling patterns used in some data sets. The traceroute data sets (except uw-1) use exponentially-distributed random inter-measurement times. By the PASTA (Poisson Arrivals See Time Average) principle [26], the fraction of requests that fail in the traceroute experiment should correspond to the fraction of time the network is down (neglecting the source failure sampling bias listed above). The HTTP data sets sample routes according to the request pattern from clients and therefore the samples reflect the request-averaged behavior of the system, but this may differ from the time-average behavior of the system because the state of the network may affect whether a trace sample is taken or not. For example, if a user's first request to a server fails, it is unlikely the user will send additional requests to the server in the near future.

Dataset	Year	Duration	nhosts	nsamples	type
Paxson-1	1994	45 days	27	7016	traceroute
Paxson-1-na	1994	45 days	22	4903	traceroute
Paxson-2	1995	48 days	33	28943	traceroute
Paxson-2-na	1995	48 days	23	12613	traceroute
UW-1	1999	34 days	36	54391	traceroute
UW-3	1999	7 days	36	78816	traceroute
UW-4a	1999	14 days	14	181151	traceroute
UW-4b-all	1999	12 days	38	58488	traceroute
bo1	2000	16 days	1/194284	8142820	HTTP
rtp	2000	12 days	1/282830	18577435	HTTP
squid2	2000	3 days	9/327835	23490956	HTTP

Table 2: Network failure traces. For traceroute traces, nhosts is the number of participating nodes; each node acted as both a source and a destination. For HTTP traces, nhosts shows {the number of proxy caches traced}/{the number of servers they contacted.} Nsamples shows the number of attempts to communicate in each trace.

3.2 Datasets

Table 2 summarizes the traces we used to construct our network failure model.

Paxson-1 and Paxson-2 are traceroute measurements taken and originally analyzed by Paxson [18]. In Paxson-1 each site sends a probe to a randomly chosen target with exponential inter-probe interval of 2 hours. The number of sites varies over the course of the trace up to a maximum of 27 nodes. In Paxson-2 40% of measurements from a site are to a randomly chosen target site with inter-probe interval of 2 hours. The remaining 60% of a sites measurements are sent in "bursts" with the same 2-hour inter-probe interval but without changing the target from the previous probe.

Paxson-1-na and Paxson-2-na represent the subset of measurements in the Paxson traces that both begin and end in North America.

UW-1, UW-3, UW-4a, and UW-4b-all are traceroute traces collected by Savage et. al [20] at the University of Washington. In UW-1, the inter-measurement time is a uniform distribution with a mean of 15 minutes and each measurement is between a random pair of hosts. In UW-3 and UW-4b-all a random pair of hosts is selected for measurement using an exponential distribution with a mean of 9 and 150 seconds, respectively. In UW-4a, every server sends requests to every other server at the same time; these episodes are scheduled using an exponential dis-

	Temp	Perst	Total
paxson1	1.3%	0.43%	1.7%
paxson1-na	1.4%	0.48%	1.9%
paxson2	1.7%	0.19%	1.9%
paxson2-na	0.60%	0.072%	0.7%
uw1	NA	0.15%	NA
uw3	NA	0.027%	NA
uw4a	NA	0.61%	NA
uw4b-all	NA	0.0047%	NA
bo1			7.4%
rtp			1.5%
squid2			1.1%

Table 3: Fraction of requests that fail.

tribution with mean of 1000 seconds.

A problem with uw4a is self-interference. Approximately 10 requests are issued by each node "simultaneously", which may increase packet losses. To reduce this effect, we filter obvious cases of self interference: if at least one outbound packet in a burst of requests from a node makes it to its destination, then we conclude that connectivity from that node to the Internet is available at the time of the burst. If any other traceroute during the burst fails to make it beyond the source node subnet or the "bottleneck" routers that are traversed on all successful outbound requests from that node, we conclude that traceroute was a victim of self-interference and discard it from the trace set. We use a similar procedure to filter bursts of inbound traceroutes to destinations.

Bo1, rtp, and squid2 are traces of HTTP requests taken at proxy caches that are part of the squid cache hierarchy [25]. Bo1 and rtp are from individual proxies, and squid2 combines requests from nine proxies. We first filter the trace to remove requests satisfied locally (e.g., a cache hit) or indirectly (e.g., via a sibling cache). We then filter all TCP_REFRESH_MISS requests from the trace because such requests fail a disproportionate fraction the time (80% to 90% of the TCP_REFRESH_MISS requests fail in most of the traces.) We then count codes between 200 and 399 as successful network connections from the proxy to the server and count codes 504 ("Gateway time out") as failed connections. We ignore records with other result codes.

3.3 Failure rates

For the Paxson data sets, we find "temporary" failure rates (where connectivity was interrupted for at least 30 seconds during a traceroute episode but where the traceroute episode eventually succeeds in contacting its target) of 0.6% to 1.7%, and find "persistent" failure rates (where the traceroute failed to reach its destination) of 0.07% to 0.48% when end-host failures are excluded. The overall failure rates of these traces is 0.7% to 1.9%.

We find similar failure rates for the uw and HTTP traces. As Table 3 shows, the uw data sets show similar persistent failure rates to the Paxson traces (though the temporary failures are, unfortunately, not included in the uw data sets.) The rtp and squid2 traces' failure rates are similar to the overall traceroute rates – 1.5% and 1.1%. The bo1 trace shows a higher rate, and we note that the component traces of the squid2 data set show considerable variability, with individual proxies exhibiting failure rates of 0.37%, 0.5%, 0.67%, 0.85%, 1.2%, 1.6%, 1.8%, 3.6%, and 6.8%.

Overall, the data suggest that typically 0.5% to 2% of requests fail to communicate with their server, but some proxies differ considerably from this typical behavior and see rates as low as 0.36% or as high as 7% in our data sets. Our simulations in Section 4 will use a default failure rate of 1.25%. This means that a given pair of nodes is unable to communicate 1.25% of the time due to network failures lasting 30 seconds or longer.

3.4 Failure locations

We focus on the traceroute data sets because they include hop-by-hop routing information, and we use the following heuristics to classify failures into the near-source, in-middle, and near-destination categories. We define the *source bottleneck set* as the set of routers that are visited by all successful outgoing requests from a node and the *source subnet* as the set of routers whose IP addresses match the source node's in the top 24 bits. We define the destination bottleneck and subnet sets similarly. A failed request is classified as a near-source failure if (a) the request only succeeds in reaching nodes in the source bottleneck set or source subnet set or (b) two or more successive requests from the same source to different destinations fail. We use a similar definition for the near-destination failures. We classify all remaining failures as in-middle failures.

Figure 1 summarizes the fraction of failures classified as near-source, near-destination, or in-middle by these criteria.

As noted earlier, the methodology used for gathering the traces may tend to undersample during periods when the network near the intended source of the measurement is malfunctioning. As one might therefore expect, the near-destination failure rate is higher than the near-source failure rate in most of the data sets. Given that for these data sets the source

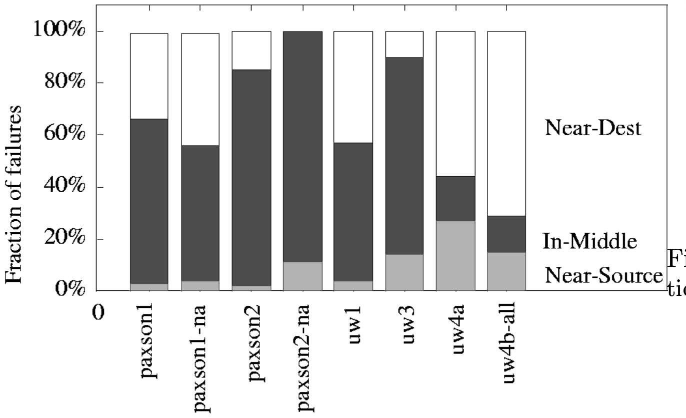


Figure 1: Location of failures. The segments of each bar show the fractions of failures that occur at particular locations.

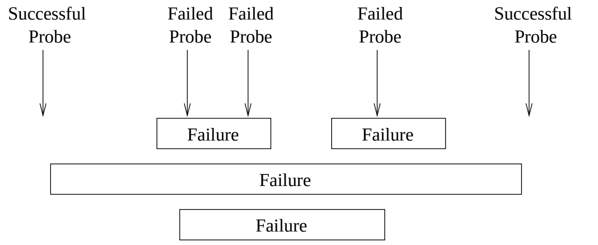


Figure 2: Illustration of ambiguity in failure event duration from probe samples.

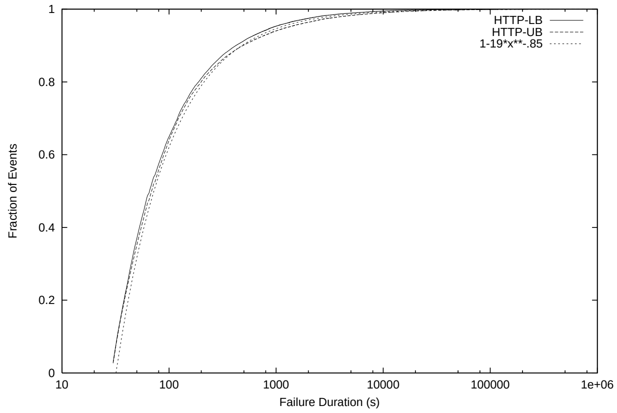


Figure 3: Cumulative distribution function of failures lasting longer than 30 seconds in the combined HTTP data sets.

and destination nodes were selected from the same collection of traceroute hosts, we speculate that the near-destination failure rates reported above are more representative of the stub network disconnection rate than the near-source rates.

Overall, we observe that both stub network and interior failures contribute significantly to failures but that the relative prevalence of interior compared to stub failures varies. Paxson1-na and uw3 are dominated by interior failures, paxson1 and uw1 have similar amounts of interior compared to stub failures, and the other traces are dominated by stub failures. Our simulation by default evenly distributes failures across near-source, in-middle, and near-destination categories, but we also examine the impact of more skewed distributions.

3.5 Failure durations

As Figure 2 illustrates, the relatively low sampling rate at some locations and data sets can lead to two ambiguities for estimating the duration of a failure event. First, the samples could either be from one long failure or two (or more) short failures. Second, the beginning of the failure could have occurred soon before the first probe that failed or soon after the last probe that succeeded; there is a similar ambiguity for the ending time. For our baseline model, we assume that any series of failures without an intervening success represents a single failure event, and we use the data to provide both upper and lower bounds on the duration of each such event. An area for future work is developing failure data sets that provide better failure-duration information.

Figure 3 shows the cumulative distribution function of the duration of failure events lasting longer than 30 seconds for the http data sets. Our rationale for only looking 30 second or longer failures is that short failures may be better handled by transport-level retransmission than the more aggressive techniques we explore. Excluded sub-30-second failures account for 28.8% of the failed probes and 70.7% of the failure events for the lower bound list of durations; they account for 6.5% of the probes and 31.7% of the events for the upper bound list.

This distribution appears well modeled by the function $F(x) = 1 - 19x^{-0.85}$. The average duration of this function is unbounded. For our simulations, we arbitrarily place an upper limit on failure durations of 500,000 seconds, which yields an average failure duration of 609 seconds.

For the traceroute data sets, the long inter-probe times make it difficult to precisely characterize the duration of short persistent failures. Figure 4 compares the duration of long failure events – 1000 seconds or more – between the traceroute and HTTP data sets. For clarity, we combine all of the traceroute failures into one data set and all of the HTTP failures into another one and show the upper and lower bounds of the cumulative distribution function for each. Note that the traceroute lower bound line is lower than the upper bound line for much of the range because the two lines track different sets of data once the sub-1000-second events are excluded.

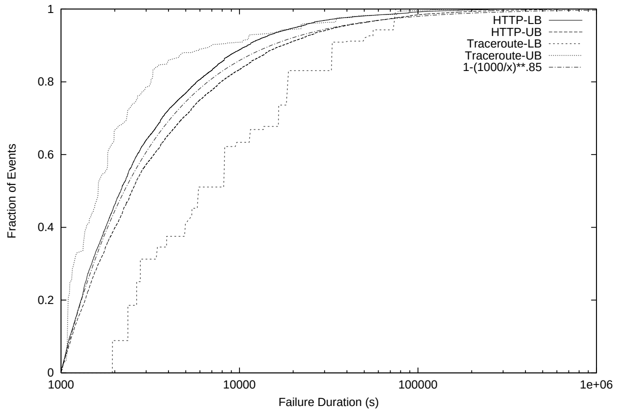


Figure 4: Cumulative probability function of the duration of failures lasting longer than 1000 seconds. The x axis is the duration and the y axis is the probability that a failure event will have less than the specified duration.

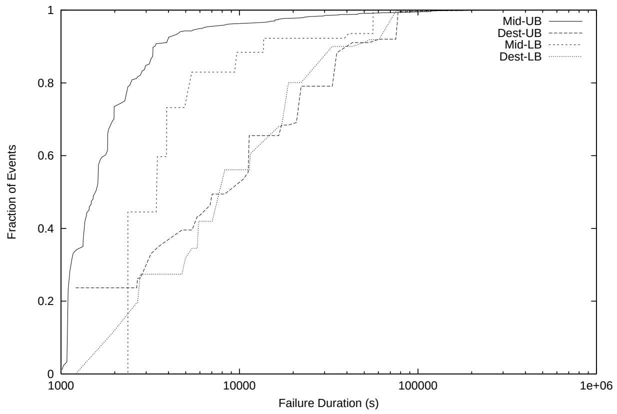


Figure 5: Cumulative duration distribution of traceroute in-middle and near-dest failures that last at least 1000 seconds.

Based on the data in Figures 3 and 4, it appears that the duration of 30+-second HTTP and 1000+-second traceroute failure events display cumulative distribution functions of the form $F(x) = 1-(k/x)*\alpha$ with $\alpha \approx 0.85$. This function corresponds to a *heavy-tailed* distribution typified by a significant number of long failures, decreasing recovery rate, large variance, and high mean [7].

The traceroute and HTTP data sets each have significant limitations for the purposes of modeling failure duration: the HTTP data sets may contain sampling-interval biases, and the traceroute data sets' sparse sampling interval leaves uncertainty about the duration of individual events. Despite these limitations, the data sets appear qualitatively consistent with one another, which suggests that the results are not anomalous.

Finally, we examine the question of whether there is evidence that durations are significantly different for stub failures than for in-middle failures. Figure 5 plots the upper and lower bounds of the failure-duration CDFs for the long (1000+ second) in-middle and near-destination failures in the combined traceroute data sets. To a first approximation, it appears that the duration distributions are similar at least for the hosts examined.

3.6 Failure arrivals

The remaining parameter for our model is the failure interarrival period. We use a simple model: we assume that failures arrive independently with exponentially-distributed distributions. Future work is needed to examine this assumption.

The duration model above yields an average failure duration of 609 seconds. Since 28.8% and 6.5% of failed probes encounter sub-30 second failures (for the lower bounds and upper bounds, respectively), if the failure rate for all failures is 1.5%, then the failure rate if only 30-second or longer failures are considered is between 1% and 1.5%. Our simulations assume a default failure rate of 1.25% for 30-second or longer failures, yielding an average inter-failure interval for our default model of 48111 seconds.

4 Masking network failures

This section studies two classes of techniques for improving end-to-end service availability by masking network failures. Client-independence techniques – such as data caching, prefetching, and mobile code – provide a (possibly degraded) version of a service using local resources when the remote server cannot be contacted. Routing and connectivity techniques use alternate network paths to route around failures.

These experiments focus on two goals. First, they seek to quantify the potential effectiveness of these techniques at improving service availability. In order to provide information about of a broad range of techniques, our experiments abstract away implementation details and thus provide an upper bound on the techniques' effectiveness. The second goal of our experiments, therefore, is to understand what factors may limit specific instantiations of these techniques and to quantify their impact.

Although this paper focuses on service-level techniques for improving availability, researchers will certainly work to improve reliability at the hardware and transport layers as well. Indeed, achieving the goal of four- or five-nine services will likely require advances at all layers. So, in addition to the experiments described above, we assess the sensitivity of our results to changes in the reliability of the underlying infrastructure.

4.1 Client independence

A range of client independence techniques are available.

1. **Caching.** Caching hides network and server failures by serving requests from a nearby cache rather than a distant server [9]. Most web clients today include some form of caching.

2. **Relaxed consistency and push-updates.** Relaxed consistency can improve availability by allowing caches to serve potentially stale data during failures rather than requiring the cache to use (unavailable) current data. Alternately, under a push-updates protocol [14, 22], servers may update cached copies before clients issue reads requesting the new versions. Push-updates thus improves the chance that a cache will contain current data during a disconnection.

3. **Prefetching.** Prefetching brings objects close to a client before the client accesses them. **Hoarding**, a form of prefetching in which a user identifies groups of objects to fetch, is effective for disconnected operation in file systems [11], and the Microsoft Internet Explorer browser implements a hoarding option for web pages. **Server push** [6] such as the content distribution networks becoming commercially available can be thought of as a form of server-directed prefetching.

4. **Replication of active objects.** Several researchers have proposed systems in which active service objects may be cached or replicated and then executed[1, 2, 10, 23, 24]. These techniques may provide ways to extend the benefits of caching, relaxed consistency (or "application-specific adaptation" [16]), and prefetching to the significant fraction of web services that are not cachable [5, 28].

This set of experiments examines the potential effectiveness of using these client independence techniques to improve robustness of Internet services by transforming *failed sessions* that are interrupted by network disconnections into *degraded sessions* that are served by the cache or by downloading mobile extensions. Clearly, the relative advantage of degraded sessions over failed sessions will vary from service to service: some services can provide full service while disconnected, others can provide tolerable service across short disconnections, and still others require continuous on-line communication with a remote site to be effective. To cope with this wide range of service behaviors, this experiment does not attempt to quantify the benefit of degraded service over failed service; instead it seeks to quantify how often services have the option to use caching, relaxed consistency, prefetching, or mobile extensions to improve their robustness to network disconnections.

Workload	Date	Clients	Servers	Sessions
Squid-P	3/28/00–4/03/00	1	131193	1557875
Squid-C	3/28/00	107	52526	403235
BU-P	1/17/95–5/17/95	1	4614	56789
BU-C	1/17/95–5/17/95	33	4614	68949

Table 4: Web access trace parameters.

Workload and methodology. In addition to the failure model described above, the simulator uses two sets of web service access traces to represent Internet service access patterns. Table 4 summarizes key parameters for these traces. We examine both the bo1 Squid trace described earlier and a four-month trace taken at clients at Boston University [3]. This trace is old, but it includes client cache hits, and the client-ID mappings are not changed over the trace period. We examine both traces from the point of view of a proxy shared by all clients in the trace (Squid-P and BU-P) and from the point of view of individual client machines (Squid-C and BU-C) with no shared proxy. Because the Squid traces change the client-ID mappings daily, we only look at the first day of the Squid-C trace.

For our simulations, we post-process the traces to group individual accesses into *sessions.* We define a session as a set of accesses from a client (-C traces) or proxy (-P traces) to a single server in which the maximum gap between successive requests is 60 seconds. Our figure of merit for availability is the fraction of sessions that complete without interruption.

Our simulator tracks the references to objects in the traces and uses trace information to classify the objects as cachable or uncachable and to identify when objects change. It assumes that each simulated client (-C traces) or proxy (-P traces) has an infinite cache that stores all objects accessed previously in the simulation.

To evaluate prefetching techniques and mobile code as a class without knowing the details of each service, we use the simulation parameter *install_time* to represent the amount of time from the first access by a client or proxy to a service until the service has downloaded sufficient state or programs or both to the cache to cope with a network disconnections. Our default *install_time* is 100 seconds. During the *install_time*, clients and proxies must access the service from cached data or via the origin server.

If the network remains up during an entire session, the simulator classifies the session as *No Failure*. For sessions in which the network fails, the simulator examines the objects referenced in the session and classifies the session as follows: *Cache Hit* if all requests are for fresh cached web objects; *Stale Hit* if all requests are for cached web objects and if some of those objects require updates from the server; *Hoardable Degraded* if the *install_time* for the service has completed at time of the network failure and all requests are for cachable objects but some miss; *Dynamic Degraded* if the *install_time* has completed at the time of the failure but not all session data are cachable; and *Fail* if the *install_time* has not completed at the time of the failure and either some data are not cachable or some data are cache misses.

For these experiments, we set the failure-location distribution to make all failures "in-middle" failures, and we conduct five trials with different random seeds for the network failure model and graph the mean and standard deviation of results. We describe improvements to failure rates in terms analogous to the common definition of "speedup" [8]: $improvement = \frac{failureRate_{orig}}{failureRate_{new}}$.

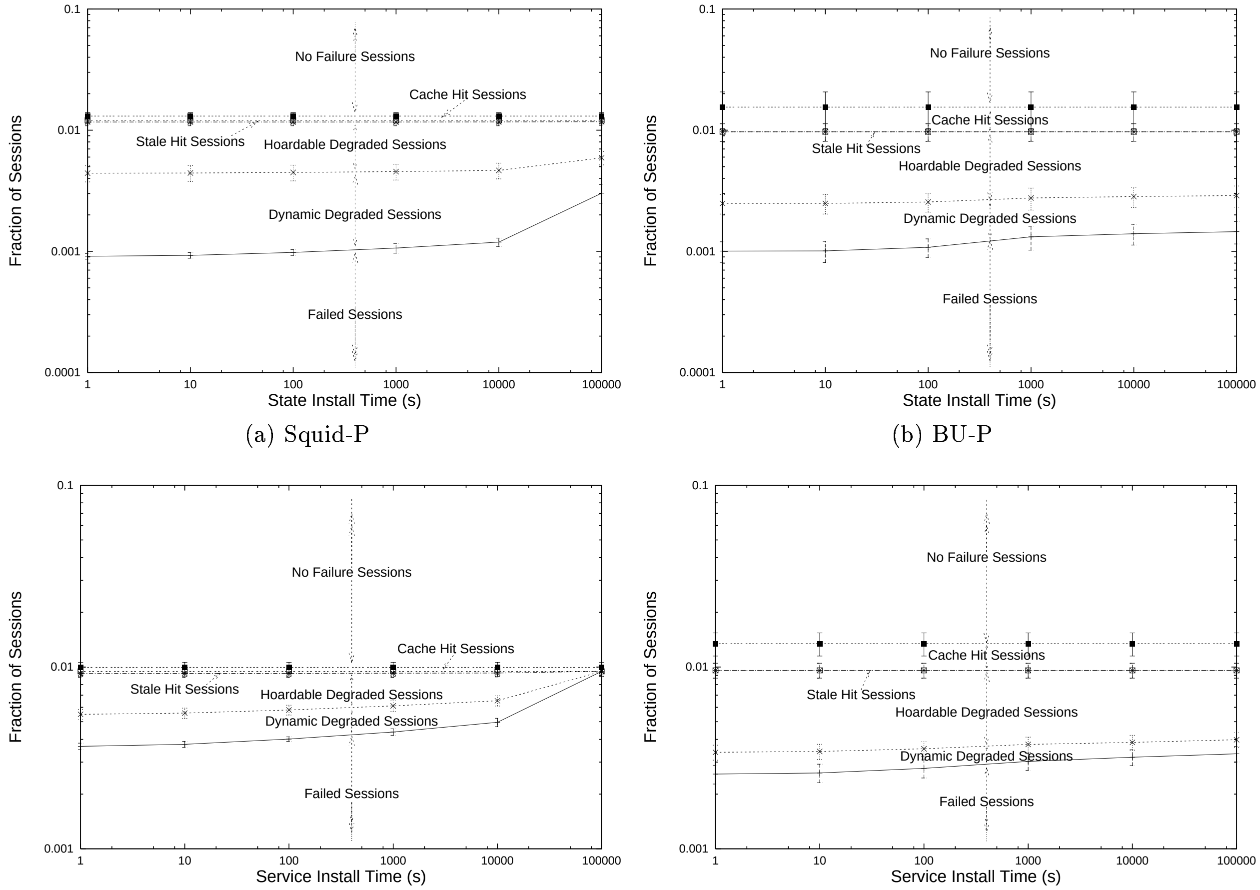
Results. First, we examine the effectiveness of these general techniques as well as the extent that installation time limits performance. The y-axis of Figure 6 shows the fraction of sessions classified in the categories listed above on a logarithmic scale so that equal intervals reflect equal improvements to failure rates. The x-axis shows the *install_time* for each service also using a log scale, and each graph shows these results for a different workload. When installation times are short, the combined effect of all techniques is to improve the failure rate by at most factors of 14.4 (Squid-P), 15.4 (BU-P), 2.7 (Squid-C) and 5.22 (BU-C) for the four workloads compared to the failure rate that would be encountered if each request were sent to the origin server. The improvements available from caching alone appear small (improvements to failure rates of 1.1, 1.6, 1.1, and 1.4 for caching and of 1.1, 1.6, 1.1, and 1.4 for caching plus relaxed consistency or push-updates) although in the Squid workload's lower-level caches may hide sessions that only reference cached data, causing us to understate the benefits of caching alone. Conversely, aggressive prefetching plus caching may be able to achieve significant improvements for those services where prefetching is feasible; the simulations indicate upper bounds of 3.0, 6.2, 1.8, and 4.0 for this combination.

The only limiting factor to the mobile extensions line in this model is our assumption that each service requires different extension code and that extensions cannot be downloaded until a service is first accessed. Under this assumption, improvements to failure rates are limited to about an order of magnitude for these traces because if the network is down when a service is first accessed or during the first *install_time* of accesses, no code is available to mask the failure. These "compulsory misses" also limit the prefetching line in these graphs. If compulsory misses and initialization times are ignored, mobile extensions could, in principal, provide at least degraded service at all times and prefetching could provide improvements in failure rates of up to 3.7, 9.7, 4.7, and 12.2.

The available benefits fall gradually as installation time increases and compulsory misses become more severe. At a 10,000 second installation time the upper bound on availability improvements are 11.0, 11.1, 2.0, and 4.2 for the four workloads. This result is promising: it suggests that services that need to download significant amounts of state to provide acceptable disconnected service may have the opportunity to do so.

Next, we examine the sensitivity of our results to the underlying infrastructure failure rate. Figure 7 shows session results for Squid-P as we vary infrastructure failure rates by reducing the time between failures and leaving the failure duration distribution unchanged. The other workloads (not shown) are qualitatively similar. These data suggest the improvement in session failure rate provided by caching, prefetching and mobile extensions are relatively insensitive to the underlying network failure patterns between failure rates of 12.5% to .0125%. At failure rates below that, the traces are so short that relatively few failure events occur, and our results have too much variance to reach definitive conclusions.

The experiments above suggest that to improve overall service availability significantly, services may need to resort to prefetching and mobile extensions rather than relying on caching alone. Unfortunately, these techniques can dramatically increase the demand for resources at a client, proxy, or network. A key limiting factor, therefore, may be how many resources a cache can devote to each hosted service or, equivalently, how many services a cache can simultaneously host. Figure 8 shows session results when the BU-P proxy maintain only a finite number of local copies of prefetched services and mobile extensions and evicts the rest using an MFU policy (results for



(a) Squid-P (b) BU-P

(c) Squid-C (d) BU-C

Figure 6: Session result v. state installation time. Each region between two lines represents the fraction of sessions that can be handled by the specified technique plus those above it in the graph.

LRU replacement and exponentially decaying average MFU are similar but not shown.) For the other workloads (not shown), the results are qualitatively similar, but the cache size needed for full benefits is larger for the Squid-P workload and smaller for the squid-C and BU-C workloads due to the differing number of services accessed by each of these workloads.

These experiments suggest that to take full advantage of client independence for improving availability, client and proxy virtual machines must be scalable to handle hundreds or thousands of simultaneously downloaded extensions in order to replicate a significant fraction of accessed sites. This paper does not explicitly address the issue of the scalability of Java virtual machines to such large numbers of extensions nor the resource management challenges in partitioning resources across so many untrusted extensions, and these appear to be an important subjects for future work.

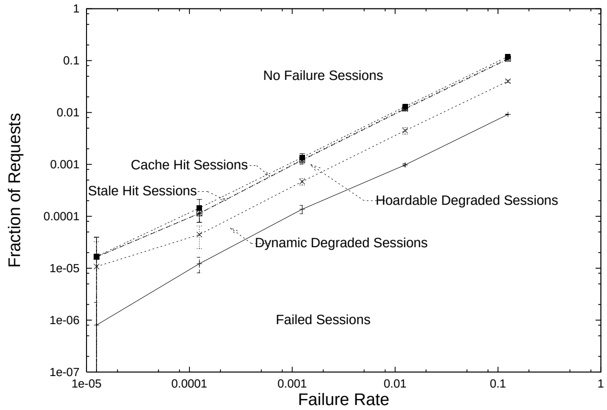


Figure 7: Session results (squid-P) as network failure rates vary.

4.2 Network routing

In this section we evaluate strategies that route around network failures. These techniques still send their requests to the service's origin server, but may

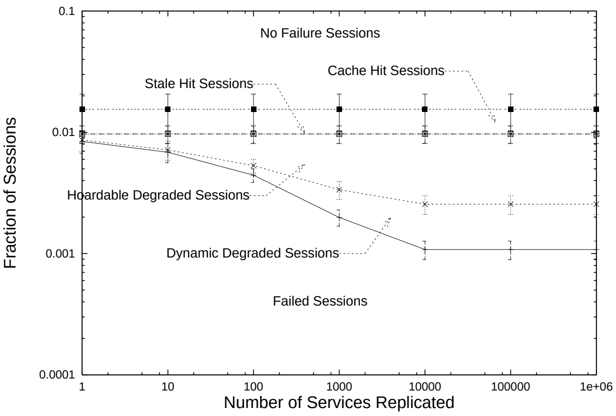


Figure 8: Service failure rate v. number of cached service extensions.

use alternate routes when failures occur.

To simplify the analysis, we classify these techniques into three broad categories according to the types of network failures they can avoid.

1. **Server stub failures.** To cope with server stub failures, services may, for example, connect to the Internet via multiple Internet Service Providers (ISPs), via multiple physical connections maintained by the same ISP, or may maintain multiple geographically-distributed replicas.

2. **In-middle failures.** Failures in the middle of the network may be avoided by, for example, dynamic routing or overlay networks as well as by choosing a different service network address (ISP or geographic replica) to contact.

3. **Client stub failures.** Clients may avoid client stub network failures by, for example, connecting to the Internet via multiple ISPs or multiple physical connections.

As in our analysis of client independence techniques, we abstract implementation details of these classes of techniques and focus on bounding improvements that they may provide. Key factors that will limit these gains include the time needed to detect a failure and switch to a different route as well as performance degradation as potentially less performance-optimal routes are used to improve robustness. Thus, these results can be interpreted in the context of a given implementation technique by using these experiments to quantify the fraction of sessions served by that technique and considering those sessions "degraded" according to the implementation details of that technique. For example, if an overlay routing algorithm typically takes 45 seconds to detect and route around an in-middle failure, then the in-middle requests shown in the graphs would be degraded by a 45-second pause. As with the client-independence techniques, the impact of such a degradation must be interpreted on a service-by-service basis.

We use the same workloads and basic methodology as for the client-independence experiments. We group requests into sessions, and classify each session as a success if no network failures occur or if the request can be satisfied from the cache. If a network failure occurs, we classify the session according to whether the failure occurred near-source, near-destination, or in-middle.

Results. In our first set of experiments, we vary the fraction of failures in each location category. These graphs are omitted. Across a wide range of ratios, the findings are es expected: the fraction of failures that each class of techniques can handle varies in proportion to the fraction of failures assigned to a particular location category. For example, when in-middle failures account for 50% of all failures, techniques that avoid in-middle failures but not others can improve failure rates by about a factor of two.

Although our failure data do not allow us to precisely characterize the fraction of failures that go to each location category, the data do suggest a lower bound on this fraction between 5% and 10% for each category. This is unfortunate, since it appears to limit the peak benefits we can expect from any technique that fails to address any one of these categories to at most an order of magnitude improvement in failure rate (and perhaps much less.) We speculate that from a service provider's perspective, deploying solutions to server stub failures may be relatively simple, deploying solutions to in-middle failures may be somewhat more complex, and deploying solutions to client stub failures may be difficult.

Figure 9 shows the sensitivity of these results as we vary the infrastructure failure rate. As for the client-independence strategies, the relative improvements to failure rate provided by these techniques remains stable over a wide range of underlying failure rates.

5 Conclusions

Although Internet services can deploy highly available servers, deploying highly available *services* remains problematic due to connectivity failures. A typical client may not be able to reach a typical server for 15 minutes per day.

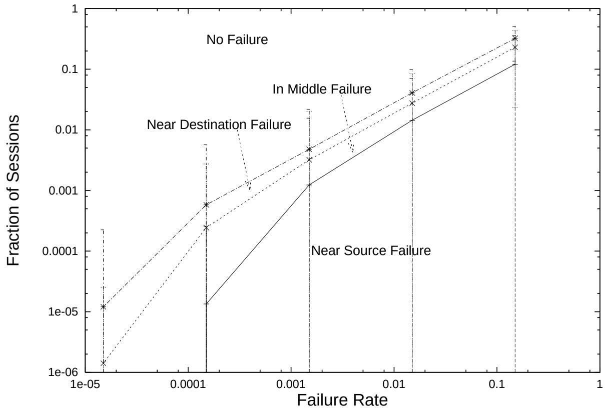


Figure 9: Session failure rate v. infrastructure failure rate (BU-P trace).

In this paper, we develop a network failure model and an evaluation stategy for studying broad classes of techniques for coping with connectivity failures.

These experiments suggest that end-to-end techniques have the potential to significantly improve WAN service availability. Although the gains from traditional caching appear limited, the potential gains from more aggressive techniques – such as prefetching, content distribution, and replication of active objects – appear substantial, at least for those services whose semantics allow for disconnected operation and whose data footprints are small enough for replication. Quantifying the fraction of services that meet these requirements is an important topic for future work, as is developing a scalable extensible proxy design that can host hundreds or thousands of downloaded extensions.

As one might expect, it appears that routing-based techniques may provide substantial availability improvements by finding alternate paths to servers. However, we find that a significant fraction of network failures appear to come from each major segment of the network infrastructure – near-source, in-middle, and near-destination. This suggests that techniques to avoid a single type of failure such as multi-hosting a server with multiple ISPs may have only modest end-to-end impact.

We speculate that for most services, the path to high end-to-end availablity most likely runs through a combination of approaches. For example, routing-based techniques might be used to reduce the impact of "compulsory" misses on mobile extensions while mobile extensions might be used to mask network failure classes not addressed by a particular routing strategy.

References

[1] Network Appliance. Internet content adaptation protocol (icap). DS-2326, June 2000.

[2] P. Cao, J. Zhang, and Kevin Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proc. of Middleware 98*, 1998.

[3] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW Traces. Technical Report TR-95-010, Boston University Department of Computer Science, April 1995.

[4] D. Duchamp. Prefetching Hyperlinks. In *Proc. of the Second USENIX Symposium on Internet Technologies and Systems*, October 1999.

[5] B. Duska, D. Marwood, and M. Feeley. The Measured Access Characteristics of World-Wide-Web Client Proxy Caches. In *Proc. of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[6] J. Gwertzman and M. Seltzer. The case for geographical pushcaching. In *HOTOS95*, pages 51–55, May 1995.

[7] M. Harchol-Balter. The Effect of Heavy-Tailed Job Size Distributions on Computer System Design. In *Proc. of ASA-IMS Conf. on Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, June 1999.

[8] J. Hennessy and D. Patterson. *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.

[9] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[10] A. Joseph, A. deLespinasse, J. Tauber, D. Gifford, and M. Kaashoek. Rover: A Toolkit for Mobile Information Access. In *Proc. of the Fifteenth ACMSymposium on Operating Systems Principles*, December 1995.

[11] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[12] T. Kroeger, D. Long, and J. Mogul. Exploring the Bounds of Web Latency Reduction from Caching and Prefetching. In *Proc. of the USENIX Symposium on Internet Technologies and Systems*, December 1997.

[13] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental Study of Internet Stability and Backbone Failures. In *FTCS99*, June 1999.

[14] D. Li and D. Chariton. Scalable Web Caching of Frequently Updated Objects Using Reliable Multicast. In *Proc. of the Second USENIX Symposium on Internet Technologies and Systems*, pages 1–12, Oct 1999.

[15] A. Myers, P. Dinda, and H. Zhang. Performance Characteristics of Mirror Servers on the Internet. In *Proc. of IEEE Infocom*, 1999.

[16] B. Noble, M. Satyanarayanan, D. Narayanan, J. Tilton, J. Flinn, and K. Walker. Agile Application-Aware Adaptation for Mobility. In *Proc. of the Sixteenth ACM Symposium on Operating Systems Principles*, October 1997.

[17] V. Padmanabhan and J. Mogul. Using Predictive Prefetching to Improve World Wide Web Latency. In *Proc. of the ACM SIGCOMM '96 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 22–36, July 1996.

[18] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, April 1997.

[19] J. Pitkow and P. Pirolli. Mining Longest Repeating Subsequences to Predict World Wide Web Surfing. In *Proc. of the Second USENIX Symposium on Internet Technologies and Systems*, pages 139–150, Oct 1999.

[20] S. Savage, A. Collins, E. Hoffman, J. Snell, and T. Anderson. The End-to-end Effects of Internet Path Selection. In *Proc. of ACM SIGCOMM '99*, pages 289–299, September 1999.

[21] D. Terry, M. Theimer, K. Petersen, A. Demers, M. Spreitzer, and C. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proc. of the Fifteenth ACMSymposium on Operating Systems Principles*, pages 172–183, December 1995.

[22] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design Considerations for Distributed Caching on the Internet. In *Proc. of the Nineteenth International Conf. on Distributed Computing Systems*, May 1999.

[23] G. Tomlinson, H. Orman, M. Condry, J. Kempf, and D. Farber. Extensible proxy services framework. IETF-Draft draft-tomlinson-epsfw-00.txt, IETF, July 2000. Expires January 11, 2001.

[24] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active Naming: Flexible Location and Transport of Wide-Area Resources. In *Proc. of the Second USENIX Symposium on Internet Technologies and Systems*, October 1999.

[25] D. Wessels. Squid Internet Object Cache. http://squid.nlanr.net/Squid/, August 1998.

[26] R. Wolff. Poisson Arrivals See Time Averages. *Operations Research*, 30(2):223–231, 1982.

[27] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy. Organization-Based Analysis of Web-Object Sharing and Caching. In *Proc. of the Second USENIX Symposium on Internet Technologies and Systems*, October 1999.

[28] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy. On the scale and performance of cooperative web proxy caching. In *Proc. of the Seventeenth ACM Symposium on Operating Systems Principles*, December 1999.

[29] Y. Zhang, V. Paxson, and S. Shenkar. The Stationarity of Internet Path Properties: Routing, Loss, and Throughput. Technical report, AT&T Center for Internet Research at ICSI, http://www.aciri.org/, May 2000.