End-to-end WAN Service Availability

*extended version**

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

Abstract

This study 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 network unavailability that includes key parameters such as failure location and failure duration. We then use trace-based simulation to evaluate several classes of techniques for coping with network unavailability. 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 average unavailability 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 are limited because they do not address all significant categories of network failures. By combining the techniques we examine, some systems may be able to reduce average unavailability by as much as one or two orders of magnitude.

keywords: Internet, WWW, availability, overlay routing, disconnected operation, replication, failure model.

1 Introduction

This study 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 [2, 4, 9, 18, 37, 39, 46] and overlay routing [1, 33].

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 [30], 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 [47] suggest

*This paper is an extended version of an article that will appear in the *IEEE/ACM Transactions on Networking*, which, itself, is an extended version of the paper that appeared at the *Third Usenix Symposium on Internet Technologies and Systems* (USITS01), March 2001.

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.

Although caching can improve file system availability [16, 19], there is reason to be concerned that caching alone will not significantly improve WAN service availability because much HTTP traffic is uncachable [11, 45]. This limitation motivates us to study the potential effectiveness of other techniques such as hoarding [19], push-based content distribution [13, 20], relaxed consistency, mobile extensions to ship service code to proxies or clients [2, 4, 9, 18, 37, 39, 46], anycast [3, 12, 46], and overlay routing [1, 33]. 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, we abstract away both the detailed design of the techniques and the semantic requirements of the services. By using these simplifications, we can determine upper bounds on improvements that different classes of techniques can yield. To refine these simple bounds, we then explore the sensitivity of the techniques to parameters representing factors that could limit their effectiveness. The second challenge is that available studies of WAN unavailability do not quantify several important parameters. To address this challenge, we analyze connectivity traces to develop a model suitable for evaluating techniques for coping with unavailability.

This work makes three contributions. First, we develop a WAN connectivity model that includes average unavailability, the distribution of durations of unavailability events, and the operational location of network failures. A key finding is that unavailability 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 average unavailability 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 average unavailability, but that near-client, near-server, and interior network failures all contribute significantly to average network unavailability, 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 article proceeds as follows. We first discuss related work in the areas of coping with network unavailability and modeling Internet failure patterns. 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 [16, 19, 35]. Odyssey [28] explores using application-specific adaptation to cope with disconnection by dynamically adjusting service semantics.

In the context of web services, previous studies have examined the performance benefits of caching [11, 36, 44], prefetching [10, 29, 22, 31], pushing updates [24, 36], push-based content distribution [13, 20], server replication [26], mobile code [2, 4, 9, 18, 37, 39], and overlay routing [33], but the impact on end-to-end service availability of these techniques has not been systematically quantified.

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 [18] 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 [30]. The study focuses on quantifying the prevalence and diagnosing the causes of IP-level failures. Our analysis builds on this study by studying these and other traces to determine metrics relevant to end-to-end service delivery: failure location and the duration of unavailability events.

Labovitz et al. [23] 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. [47] study NIMI and traceroute measurements taken during December 1999 and January 2000. They find that routing availability 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.

The study presented here is an extension of an earlier study by the same authors [6].

3 Network unavailability model

We seek to model parameters of network unavailability that most directly affect techniques to improve availability. This section first defines the key parameters of our model, then describes the trace workloads we study, then outlines our methodology for analyzing these workloads, and finally discusses the results of our analysis.

3.1 Definitions

We define several key concepts here drawing on terminology defined in more detail by Trivedi [38]. A service is *available* to a client when that client can communicate with it. A service is *unavailable* to a client when that client cannot communicate with it due to a network or end-host failure. For each client, a service alternates between being available and unavailable. We term a period of time when a service is continuously available to a given client an *availability event* and a period of time when a service is continuously unavailable to a given client an *unavailability event*.

Suppose a service's availability events with respect to a client have a sequence of life times T_0, T_1, T_2, $\ldots$, and a service's unavailability events have a sequence of down times D_0, D_1, D_2, $\ldots$. We refer to the probability that a lifetime is shorter than t units of time, $F(t) = P(T_i <= t)$, as the *time to failure distribution function*; we also refer to $F(t)$ as the *availability duration distribution*. Then, the system's mean time to failure is $MTTF = \int_0^\infty (1-F(t))dt$ from the start of an availability event. Under similar assumptions, the time to repair distribution function is $R(t) = P(D_i <= t)$, *Traceroute Data Sets*

Data Set	Year	Duration	nhosts	nsamples
Paxson-1	1994	45 days	27	7016
Paxson-1-na	1994	45 days	22	4903
Paxson-2	1995	48 days	33	28943
Paxson-2-na	1995	48 days	23	12613
uw-1	1999	34 days	36	54391
uw-3	1999	7 days	36	78816
uw-4a	1999	14 days	14	181151
uw-4b-all	1999	12 days	38	58488

HTTP Data Sets

Data Set	Year	Duration	nhosts	nsamples
Bo1	2000	16 days	1/194284	8142820
Rtp	2000	12 days	1/282830	18577435
Squid2	2000	3 days	9/327835	23490956

Table 1: 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.

and the mean time to repair is $MTTR = \int_0^\infty (1 - R(t))dt$. We also refer to $R(t)$ as the *unavailability duration distribution.*

A service's *average availability*, A_{av}, is the fraction of time when a service is available to an average client, and a service's *average unavailability*, U_{av}, is the fraction of time when a service is unavailable to an average client. We also consider *request-average availability* (or unavailability) — the fraction of requests in a data set that succeed (or fail) in accessing a service.

These definitions describe a binary on/off model of availability: if a service is reachable it is available; otherwise it is unavailable. An 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.

3.2 Data sets

Our basic methodology for quantifying availability patterns uses trace data sets that consist of large numbers of attempts by pairs of nodes to communicate.

We use two types of dataset. First, traceroute data sets consist of multiple traceroute measurements between pairs of nodes participating in the study. Second, HTTP data sets consist of logs of HTTP requests through public Squid [42] proxies to web servers. Table 1 summarizes the data sets we study.

Traceroute data sets. For the traceroute data sets, each traceroute episode comprises a series of probes from a source to a destination. Each probe is sent with a maximum hopcount; each router traversed by a probe packet decrements the packet's hopcount and either forwards the packet to the next router on the path (if the hopcount is nonzero) or sends a reply to the source of the probe (if the hopcount is zero). The source sets the maximum hopcount to one for the first probe and increases this maximum by one after each set of three probes. The source sends a probe after receiving a reply to the previous probe or after a 5 second timeout, and the traceroute episode ends after sending a set of three probes with the same hop count when (a) the hopcount exceeds 30 or (b) the source has received at least one reply from the final destination node.

We treat each traceroute episode as a sample of network connectivity. Following Paxson's terminology [30], we classify a traceroute episode as a *temporary failure* if (a) some packets succeed in contacting the target and (b) at least six packets in a row are dropped. A temporary failure thus indicates a connectivity interruption of at least 30 seconds and of not more than 750 seconds (the longest traceroute episode is 30 hops * 3 probes per hop * 5 seconds per probe). We classify a traceroute episode as a *persistent failure* if the traceroute fails to receive a reply from its destination. Otherwise, we regard the traceroute episode as a successful attempt by the client to communicate with the server.[1]

Paxson-1 and Paxson-2 are traceroute measurements taken and originally analyzed by Paxson [30]. In Paxson-1 each site executes a traceroute episode with a randomly chosen destination with an exponential inter-episode 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 exponential inter-episode intervals of 2 hours. The remaining 60% of a sites measurements are sent in "bursts" with the same 2-hour inter-episode interval but without changing the target from the previous episode.

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 [33] at the University of Washington. In uw-1, the inter-episode 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 each 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 distribution 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. Overall, we delete 1.6% of the requests from uw4a due to self-interference.

HTTP data sets. The HTTP data sets are traces of HTTP requests issued by HTTP proxies to HTTP servers. We post-process the trace to extract successful and unsuccessful attempts by the proxies to communicate with servers. We first filter the trace to remove the 22.6% of 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

[1] In the Paxson data sets, several traceroute end hosts are connected to the Internet via intermittent ISDN lines [30]. For those hosts, we treat at traceroute that succeeds in communicating with the last consistently reachable hop as if it has succeeded in reaching the end host.

of the time (80% to 90% of the TCP_REFRESH_MISS requests fail in most of the traces.) We ignore requests with reply code 400 or 500 (which account for 0.37% of all replies) because it is ambiguous whether connections were successful in these cases. We then count requests with code 504 ("Gateway time out") as failed connections, and we count the remaining requests as successful network connections from the proxy to the server.

As indicated in Table 1, we use three sets of HTTP traces. Bo1, Rtp, and Squid2 are traces of HTTP requests taken at proxy caches that are part of the Squid cache hierarchy [42]. Bo1 and Rtp are from individual proxies, and Squid2 combines requests from nine proxies.

Limitations. There are potential biases in our study resulting from limitations of our data sets.

First, the hosts and network paths that we trace may not be representative of typical Internet connectivity. Several of our traceroute data sets 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 [30]. Other traceroute data sets were gathered by Savage et. al [33] 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.

Second, 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 server machines' failure patterns may not be representative of those of HTTP server machines, we filter out "end-host" failures from the traceroute data sets. These factors mean that we may underestimate end-to-end failure rates.

Third, 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 partially compensate by focusing on the near-destination failures as representative of stub failures.

3.3 Analysis

Using these data sets, we examine average unavailability U_{av}, the failure event duration distribution $D(t)$, and the availability event duration distribution $F(t)$. We also study the location of failures, which we define below.

3.3.1 Average availability and unavailability

We use the request-average availability and unavailability from our traceroute data sets as our primary estimate of (time-)average availability and unavailability. The traceroute data sets (except uw-1) use exponentially-distributed random inter-measurement times. By the PASTA (Poisson Arrivals See Time Average) principle [43], the fraction of requests that fail in the traceroute traces should correspond to the fraction of time the network is down (neglecting the source failure sampling bias listed above).

	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 2: Fraction of requests that fail.

	Temp	Perst	Total
Paxson1	$(1.2 \pm 0.52)\%$	$(0.32 \pm 0.15)\%$	$(1.5 \pm 0.58)\%$
Paxson1-na	$(1.2 \pm 0.55)\%$	$(0.32 \pm 0.15)\%$	$(1.6 \pm 0.58)\%$
Paxson2	$(1.8 \pm 0.27)\%$	$(0.28 \pm 0.17)\%$	$(2.0 \pm 0.36)\%$
Paxson2-na	$(0.79 \pm 0.30)\%$	$(0.21 \pm 0.25)\%$	$(1.0 \pm 0.52)\%$

Table 3: 90-percent confidence intervals on average daily unavailability for Paxson data sets.

For completeness, we also report the request-average unavailability for the HTTP data sets. The HTTP data sets sample availability according to the request pattern from clients, and therefore the samples reflect the request-average behavior of the system. Unfortunately, this request-average behavior 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 one 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; such an effect could cause request-average unavailability to understate time-average unavailability. For another example, if networks are more likely to fail during periods of heavy load, then excess samples may be taken during periods during which requests are unusually likely to fail; such an effect could cause request-average unavailability to exceed time-average unavailability.

As Table 2 shows, for the Paxson data sets, we find that "temporary" failures (where connectivity is interrupted for at least 30 seconds during a traceroute episode but where the traceroute episode eventually succeeds in contacting its target) cause unavailability during 0.6% to 1.7% of the episodes, and "persistent" failures (where the traceroute fails to reach its destination) cause unavailability for 0.07% to 0.48% of the episodes. Combining these failures, the average unavailability of these traces ranges from 0.7% to 1.9%.

We find similar patterns for the uw and HTTP traces. For the uw data, unavailability due to persistent failures occurs on 0.0047% to 0.59% of the traceroute episodes; temporary failures are, unfortunately, not included in the uw data sets. The Rtp and Squid2 traces' request-average unavailability are similar to the overall traceroute request-average unavailability – 1.5% and 1.1%. The Bo1 trace shows worse request-average unavailability, and we note that the component traces of the Squid2 data set show considerable variability, with individual proxies exhibiting request-average unavailability of 0.37%, 0.5%, 0.67%, 0.85%, 1.2%, 1.6%, 1.8%, 3.6%, and 6.8%.

If we assume that each day's failures are independent and that the overall system changes little over the time a data set is collected, then we can consider each day of a data set as a separate experiment that estimates average daily unavailability, and we can estimate how likely

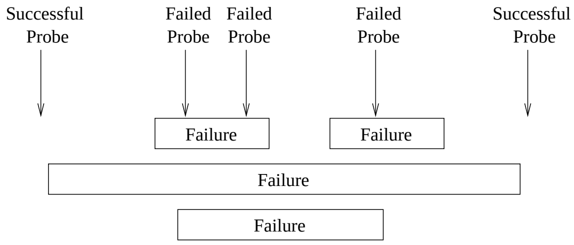


Figure 1: Illustration of ambiguity in unavailability event duration from probe samples.

it is that the sample-average average daily unavailability is close to the system's actual average daily unavailability. Unfortunately, the central limit theorem only allows us to calculate confidence intervals for samples of size greater than about 30 [17]. Thus, we examine only the Paxson data sets that exceed 30 days. Table 3 summarizes the 90% confidence intervals for the average daily unavailability due to temporary failures, persistent failures, and all failures.

Overall, the data suggest that typically 0.5% to 2% of requests fail to communicate with their server but that some proxies differ considerably from this typical behavior and see request-average unavailability as low as 0.36% or as high as 7% in our data sets. Based on the traceroute data sets, it appears likely that average daily average unavailability exceeds 0.5% but is less than 2.4% for the systems studied.

3.3.2 Duration of unavailability

We use the discrete connectivity probes in the trace to estimate the duration of periods of continuous unavailability. As Figure 1 illustrates, the relatively low sampling rate at some locations and data sets can lead to two ambiguities for estimating the duration of an unavailability event. First, the samples shown could either be from one long unavailability event or two (or more) short unavailability events. Second, the beginning of the unavailability event 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 analysis, we assume that any series of unsuccessful requests without an intervening successful one represents a single unavailability 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 data sets that provide better unavailability event duration information.

Our analysis draws on both the HTTP and traceroute data sets. Neither data set is ideal for our purpose, but the limitations of the different traces are of different types. The HTTP data set provides a large number of probes with relatively short spacing in time, which limits the duration ambiguities listed above. Unfortunately, the HTTP probes may not be sent with regularity, so the frequency with which HTTP probes encounter unavailability events of a given duration may not track the prevalence of events of that duration. On the other hand, the traceroute data sets' probes are Poisson distributed, so they are likely to sample unavailability events of a given duration in proportion to their contribution to overall unavailability time. Unfortunately, the traceroute data sets have large average inter-sample intervals, so duration ambiguities limit the precision of traceroute measurements of unavailability event duration.

As described above, we group consecutive failed connectivity probes into unavailability events. For each event, we count the number of unsuccessful probes encompassed by the event, and we determine the upper and lower bounds on the durations of the event. We then generate a count of unsuccessful probes that encounter unavailability events of any given duration.

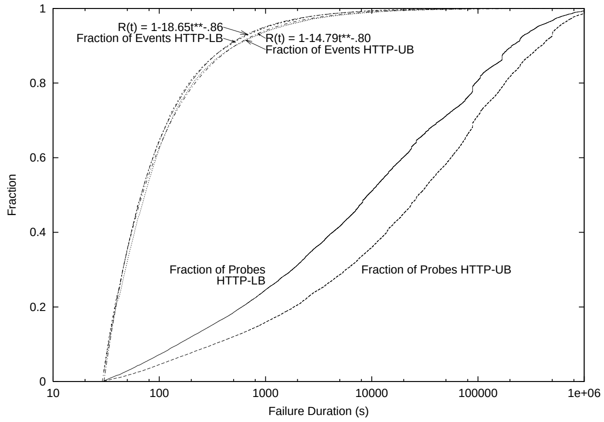


Figure 2: Cumulative distribution of the fraction of failed probes encountering unavailability events lasting (t) or fewer seconds (for $t > 30$) and the estimated unavailability event duration cumulative distribution function $R(t)$ for unavailability events lasting at least 30 seconds in the combined HTTP data sets.

The two right lines of Figure 2 show the fraction of probes that sample unavailability events of duration t seconds or shorter in the combined HTTP data set excluding unavailability events shorter than 30 seconds. Our rationale for only looking at 30 seconds or longer unavailability events is that short periods of unavailability may be better handled by transport-level retransmission than the more aggressive techniques we explore. Excluded sub-30-second events account for 28.8% of the failed probes and 70.7% of the sampled unavailability events for the lower bound list of durations; they account for 6.5% of the probes and 31.7% of the sampled events for the upper bound list.

Note that we are more likely to sample a long failure event than a short one. If we assume that probes are uncorrelated with failure durations and that the expected number of probes encompassed by a randomly selected unavailability event is proportional to the duration of the unavailability event, then the time to repair cumulative distribution function $R(t)$, which specifies the probability that an unavailability event selected uniformly at random takes t or fewer seconds to be repaired, is simply a cumulative histogram of these probe counts weighted by the inverse of each count's failure duration and normalized to make the total probability equal one. The two left lines in Figure 2 are calculated in this manner and represent an estimate of the cumulative distribution function of the duration of failure events lasting longer than 30 seconds for the environment sampled by the combined HTTP data sets.

The time to repair distribution $R(t)$ for unavailability events lasting longer than 30 seconds appears well modeled by equations of the form $R(t) = 1 - At^B$. Using Dataplot's [27] modified Levenberg-Marquardt best fit algorithm, we find $R_{LB}(t) = 1 - 18.65t^{-0.86}$ and $R_{UB}(t) = 1 - 14.79t^{-0.80}$ are the best fits for the lower-bound and upper-bound data, respectively.

Visual inspection suggests that these equations are excellent approximations of the data. Unfortunately, because of the transformation used to convert the fractions of samples to fractions of events, we cannot apply standard goodness-of-fit hypothesis testing techniques such as the Chi-Square goodness of fit test or Anderson-Darling goodness of fit test [8] to these distributions. However, to put these fits in perspective, several observations are worth noting. First, the maximum positive and negative differences from the data to the fitted lines are $D^+_{LB} = .029$, $D^-_{LB} = .003$, $D^+_{UB} = .007$, and $D^-_{UB} = .031$. Second, the maximum difference between the lower bound and upper bound data are $D^+_{LB-UB} = .038$ and $D^-_{LB-UB} = 0$. Thus, the inaccuracy of fitting these equations to these data are similar in scale to the sampling ambiguity in the underlying data. Third, the large number of samples in the original untransformed data set reduces the amount of

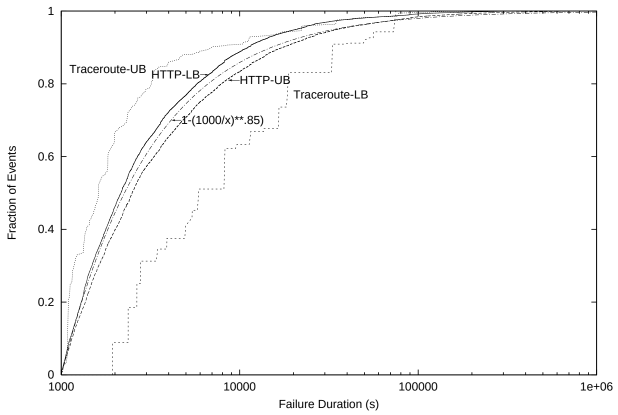


Figure 3: Cumulative probability function of the duration of failure events lasting longer than 1000 seconds. The x axis is the duration and the y axis is the probability that an unavailability event will have less than the specified duration.

variation we would expect between the sample distribution of durations and the sampled environment's true underlying distribution of durations. Thus, even relatively small deviations between the fitted equations and the data may indicate that the equations do not completely describe the environment or that the sampling ambiguities noted above have introduced significant noise in the measurements.

Fourth, the traceroute data sets appear to exhibit similar unavailability-duration behavior to the HTTP data sets despite the differing limitations of the sampling techniques. For these data, the long inter-probe times make it difficult to precisely characterize the duration of short persistent failures. Figure 3 therefore compares the duration of long unavailability events – 1000 seconds or more – between the traceroute and HTTP data sets. For clarity, we combine all of the traceroute unavailability events into one data set and all of the HTTP unavailability events 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 to the right of 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.

Based on the data in Figures 2 and 3, it appears that the duration of 30+-second HTTP and 1000+-second traceroute unavailability events display cumulative distribution functions of the form $D(t) = 1 - (k/t)^\alpha$ with $\alpha \approx 0.85$. This function corresponds to a *heavy-tailed* distribution typified by a significant number of long unavailability events, decreasing recovery rate, large variance, and high mean [14]. In particular this function corresponds to a Pareto distribution, which has an undefined mean and variance. Both data sets appear consistent with this behavior; however, it should be noted that the ambiguities in the data sets prevent us from conclusively stating that the equation models the underlying recovery process.

Besides looking at the fraction of unavailability events of a given duration, we also look at the fraction of unavailability time stemming from unavailability events of a given duration. The unavailability time distribution is simply the unavailability event duration distribution weighted by the duration of each event and normalized to a total probability of one; it is thus equivalent to the sample-weighted distribution, which is indicated by two lines on the right in Figure 2. Figure 2 suggests that although long unavailability events are rare (indicated by two left lines), they contribute a significant fraction of Internet unavailability time (indicated by two right lines).

As noted above, the traceroute and HTTP data sets each have significant limitations for the purposes of modeling unavailability event duration: the HTTP data sets may contain sampling-

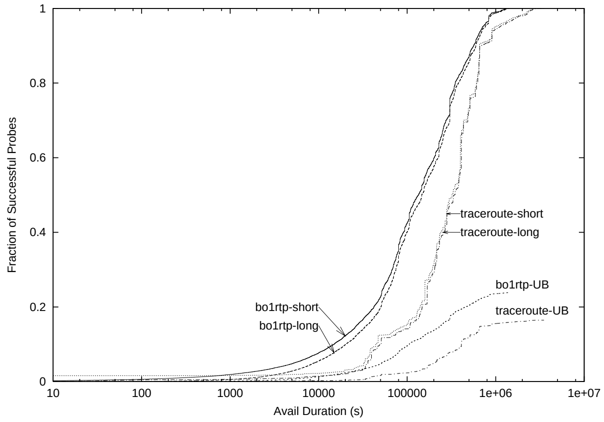


Figure 4: Fraction of successful probes encountering availability events of duration t.

interval biases, while 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.

3.3.3 Duration of availability events

Unfortunately, the data do not allow us to precisely characterize the distributions of the durations of availablity events. It is difficult to place an upper bound on failure event durations because more than 75% of successful probes in the HTTP data sets and more than 83% of successful probes in the traceroute data sets encounter availability events that may span the beginning or end of the trace. Furthermore, it is difficult to place a lower bound on failure event durations because the coarse-grained sampling in our traces may entirely miss some failure events. This sampling ambiguity seems likely to be more of a problem in estimating availabilty event duration than in estimating unavailability event duration because we expect most unavailability events to be considerably shorter than most availability events.

Figure 4 shows the cumulative distribution of the fraction of successful probe samples that encountered availability events of length t or shorter. We show data for the combined traceroute data sets and for a HTTP data set that combines the bo1 and rtp traces.[2] For the *upper bound* lines (traceroute-UB and bo1rtp-UB), we assume that each series of consecutive successful probes represents a single availability event. We also assume that the availability event begins just after the last unsuccessful probe before the event and ends just before the first unsuccessful probe after the event; if there is no unsuccessful probe to bound the event (i.e., if the event spans the beginning or end of the trace), we assign the event an infinite duration as its upper bound. For the *finite long* lines (traceroute-long and bo1rtp-long), we use the same assumptions as for the upper bound lines but exclude unbounded events. Finally, for the *finite short* lines (traceroute-short and bo1rtp-short), we assume that each series of consecutive successful probes represents a single availability event, that the availability event begins just before the first successful probe of the event and ends just after the last successful probe of the event; we exclude unbounded events from this line. Due to sampling ambiguity problem listed above, we do not attempt to estimate a lower bound. As indicated by the figure, the data leave considerable ambiguity about availability event duration distributions.

[2]The squid2 data set exceeded the memory capacity of our analysis machines for this set of experiments.

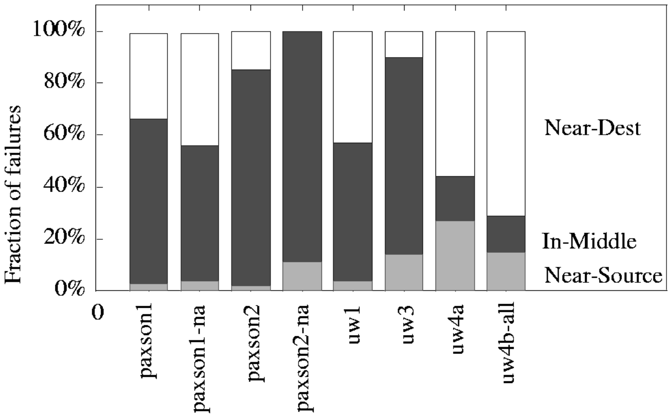


Figure 5: Location of disconnections. The segments of each bar show the fractions of failed samples encountering disconnections that occur at specified locations.

3.3.4 Failure location

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 connectivity failures 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 use 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 [23]. 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.

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 set* 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 5 summarizes the fraction of samples that encounter persistent disconnections and are classified as near-source, near-destination, or in-middle by these criteria.

Parameter	Default value	Comment
Average unavailability	$U_{av} = 1.25\%$ (> 30sec.)	Request-average uavailability varies from .4% to 7.4% in different data sets
Time to recover	$R(t) = 1 - 19t^{-0.85}$ $MTTR = 609$ sec.	Appears heavy-tailed. Truncated Pareto (max 500,000) used to give finite MTTR.
Time to failure	$F(t) = 1 - e^{-\frac{t}{48111}}$ $MTTF = 48111$ sec.	Data ambiguous. Exponential distribution assumed.
Location	Src: 25% Mid: 50% Dest: 25%	All locations significant. Ratio varies across traces.

Table 4: Default parameters for the synthetic unavailability model.

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 average unavailability rate is higher than the near-source average unavailability rate in most of the data sets. Given that for these data sets the source and destination nodes were selected from the same collection of traceroute hosts, we speculate that the near-destination unavailability 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 disconnections contribute significantly to unavailability but that the relative prevalence of interior compared to stub failures varies across traces. A Chi-square test of the frequencies [34] fails to support the null hypothesis that all 8 traces exhibit the same ratio of stub to middle disconnections at even the 1% confidence level, suggesting that the environments measured by the traces do in fact differ [47]. Qualitatively, Paxson2, Paxson2-na, and uw3 appear dominated by interior disconnections; Paxson1, Paxson1-na, and uw1 have similar amounts of interior compared to stub disconnections; and the other traces are dominated by stub disconnections.

3.4 Synthetic model

Based on the analysis above, we develop a simple synthetic unavailability model that we use to evaluate end-to-end strategies for masking network unavailability in our experiments in Section 4. Table 4 summarizes key parameters for our model.

Because the Paxson traceroute data sets are Poisson distributed and include both "temporary" and "persistent" disconnections, we draw on them to set our default average unavailability for the simulations in in Section 4, where we use a default average unavailability 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. In our above analysis of the trace data sets, we find that the average unavailability rate can vary significantly, so our experiments also vary the average network unavailability to evaluate the sensitivity of our results to this parameter.

Our analysis in Section 3.3.2 suggests that unavailability event durations for events lasting longer than 30 seconds are well modeled by $R(t) = 1 - 19t^{-0.85}$. The average duration of this Pareto function is unbounded. For our simulations, we use this function but in order to ensure a finite mean duration and to allow us to calculate a MTTF, we truncate the distribution at 500,000 seconds. This truncated Pareto distribution has a MTTR of 609 seconds.

As noted in Section 3.3.3, our analysis of the data do not provide tight bounds on $F(t)$, the time to failure distribution function. In order to make the MTTF consistent with the average unavailability and MTTR selected above, we use the equation $A_{av} = \frac{MTTF}{MTTF+MTTR}$, which lets us derive $MTTF = 48111$ seconds. Because we do not have good data about the distribution of $F(t)$, for simplicity we assume that failures arrive independently with exponentially-distributed interarrival times. Testing this assumption with better availability episode duration data is important future work.

Our analysis of disconnection location suggests that both stub and in-middle disconnections contribute significantly to overall unavailability. However, our analysis also finds significantly different fractions of different categories of unavailability events. For the experiments where disconnection location is considered (Sections 4.2 and 4.3), we assume a default ratio of near-source to in-middle to near-dest disconnections of 1:2:1, and we discuss the sensitivity of the results as we vary these ratios.

4 Masking network unavailability

This section studies two classes of techniques for improving end-to-end service availability by masking network unavailability. 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 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 availability 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 availability 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 [16]. 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 [24, 36], 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 [19], and the Microsoft Internet Explorer browser implements a hoarding option for web pages. **Server push** [13, 20, 41] such as the

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 5: Web access trace parameters.

content distribution networks becoming commercially available can be thought of as a form of server-directed prefetching. Note that prefetching is more aggressive than the "push update" approach described in the previous paragraph. "Push update" only distributes new versions of objects that have already been referenced by a cache, while prefetching can distribute unreferenced objects in order to avoid compulsory misses.

4. **Replication of active objects.** Several researchers have proposed systems in which active service objects may be cached or replicated and then executed [2, 4, 9, 18, 37, 39]. These techniques may provide ways to extend the benefits of caching, relaxed consistency (or "application-specific adaptation" [28]), and prefetching to the significant fraction of web services that are not cachable [11, 45].

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 downloaded 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 and methodology. In addition to the availability model described in Section 3, the simulator uses two sets of web service access traces to represent Internet service access patterns. Table 5 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 [7]. 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 network disconnections. Our default $install\_time$ is 100 seconds. During the $install\_time$, clients and proxies must access the service from demand-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 is unavailable for part or all of the session, 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 demand-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 before the network unavailability event begins and all requests are for cachable objects but some miss; *Dynamic Degraded* if the $install\_time$ has completed before the beginning of the network unavailability event but not all session data are cachable; and *Fail* if the $install\_time$ has not completed at beginning of the network unavailability event and either some data are not cachable or some data are cache misses.

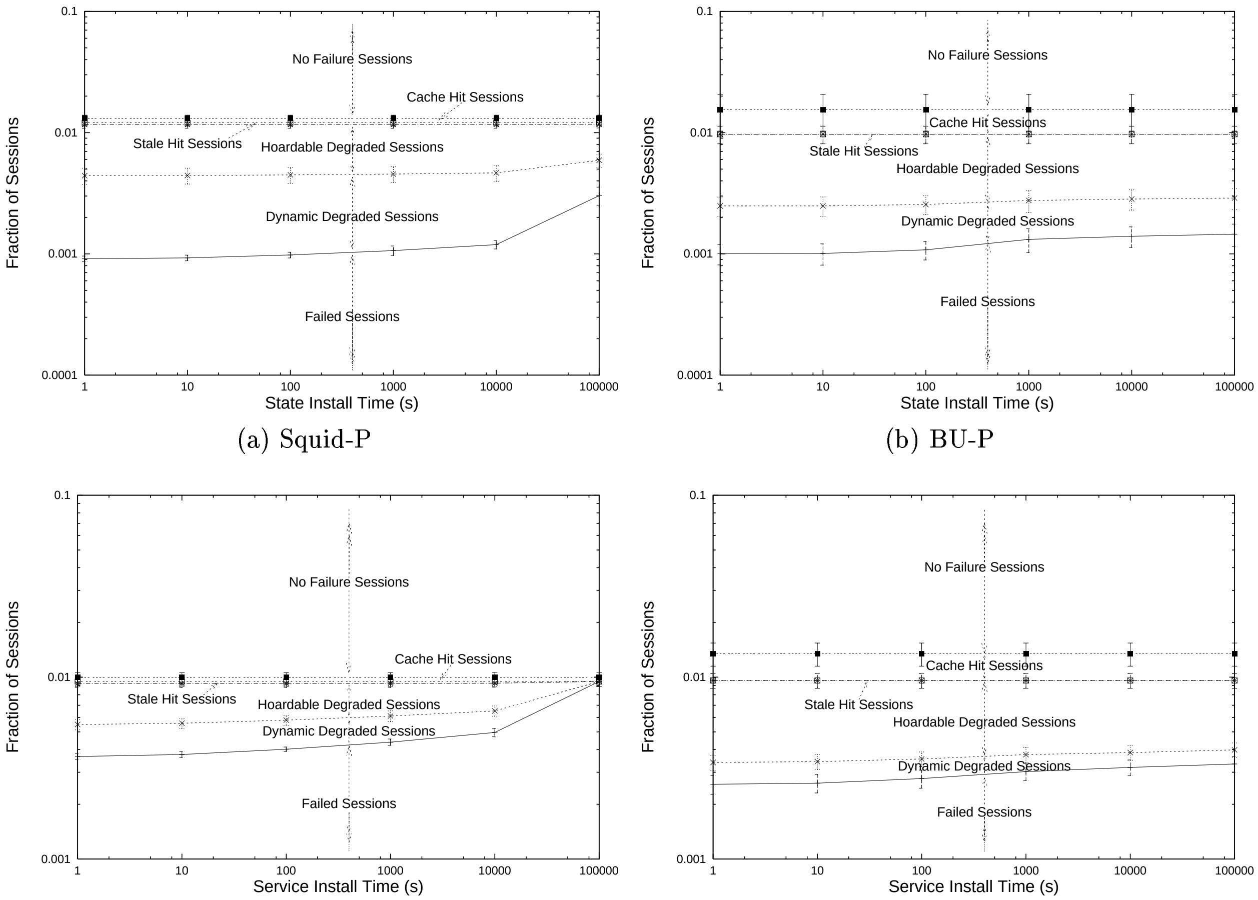
For these experiments, we set the failure-location distribution to make all failures "in-middle" failures, and we conduct five trials with different random seeds for the network unavailability model and graph the mean and standard deviation of results. We describe improvements to service unavailability in terms analogous to the common definition of "speedup" [15]: $improvement = \frac{Uav_{orig}}{Uav_{new}}$.

Results. First, we examine the effectiveness of these general techniques as well as the extent that installation time limits improvements. 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 average unavailability. 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 average unavailability 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 average unavailability that would be encountered if each request were sent to the origin server.

The improvements available from caching alone appear small (improvements to average unavailability 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). Note that 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, the BU trace are not filtered by caches, but they are old and may reflect a workload that is unrealistically easy to cache. It seems likely that caching's benefits lie between these values.

In contrast with caching alone, aggressive prefetching plus caching may 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 active object replication in this model is our assumption that each service requires different extension code and data, and that extensions cannot be downloaded until a service is first accessed. Under these assumptions, improvements to average unavailability 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 and data are available to mask the failure. These "compulsory misses" also limit the prefetching line in these graphs. If compulsory misses and initialization times are ignored – or if a longer trace were used to reduce compulsory misses – prefetching could provide improvements in average unavailability of at most 3.7, 9.7, 4.7, and 12.2 and replication of active objects and their data could, in principle, provide



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

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

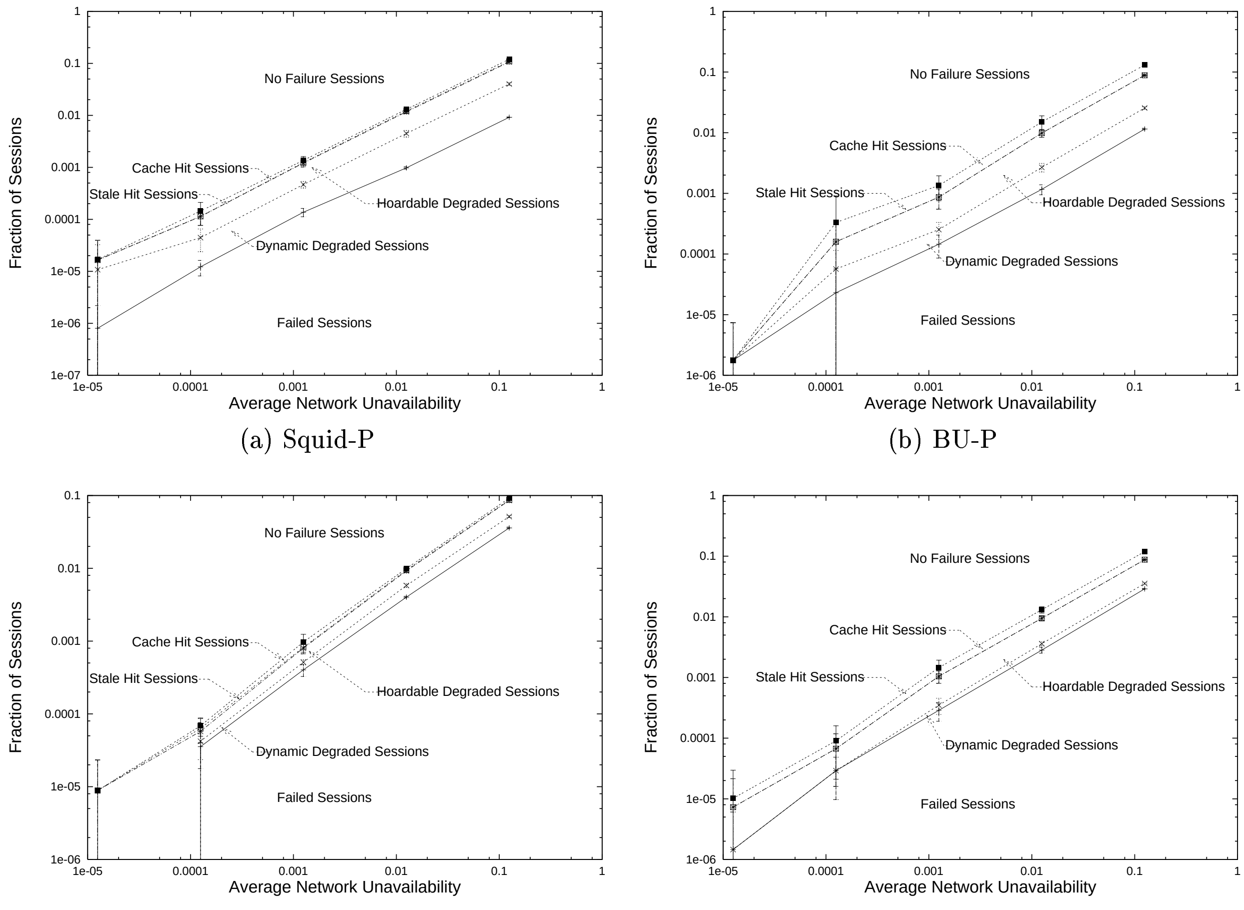
Figure 6: Session result v. state installation time. Each region between two lines represents the fraction of sessions that can be handled by the specified technique.

at least degraded service up to 100% of the time.

The available benefits fall gradually as installation time increases and compulsory misses become more expensive. 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 average network unavailability. Figure 7 shows session results as we vary average network unavailability by reducing the time between failures and leaving the unavailability duration distribution unchanged. The results for all four system configurations/workloads are qualitatively similar. In particular, these data suggest the improvement in session average unavailability provided by caching, prefetching, and replication of active objects are relatively insensitive to the underlying network unavailability patterns between average network unavailability probabilities of .0125% and 12.5%. At average network unavailabilities 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 significantly improve overall service availability, 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,



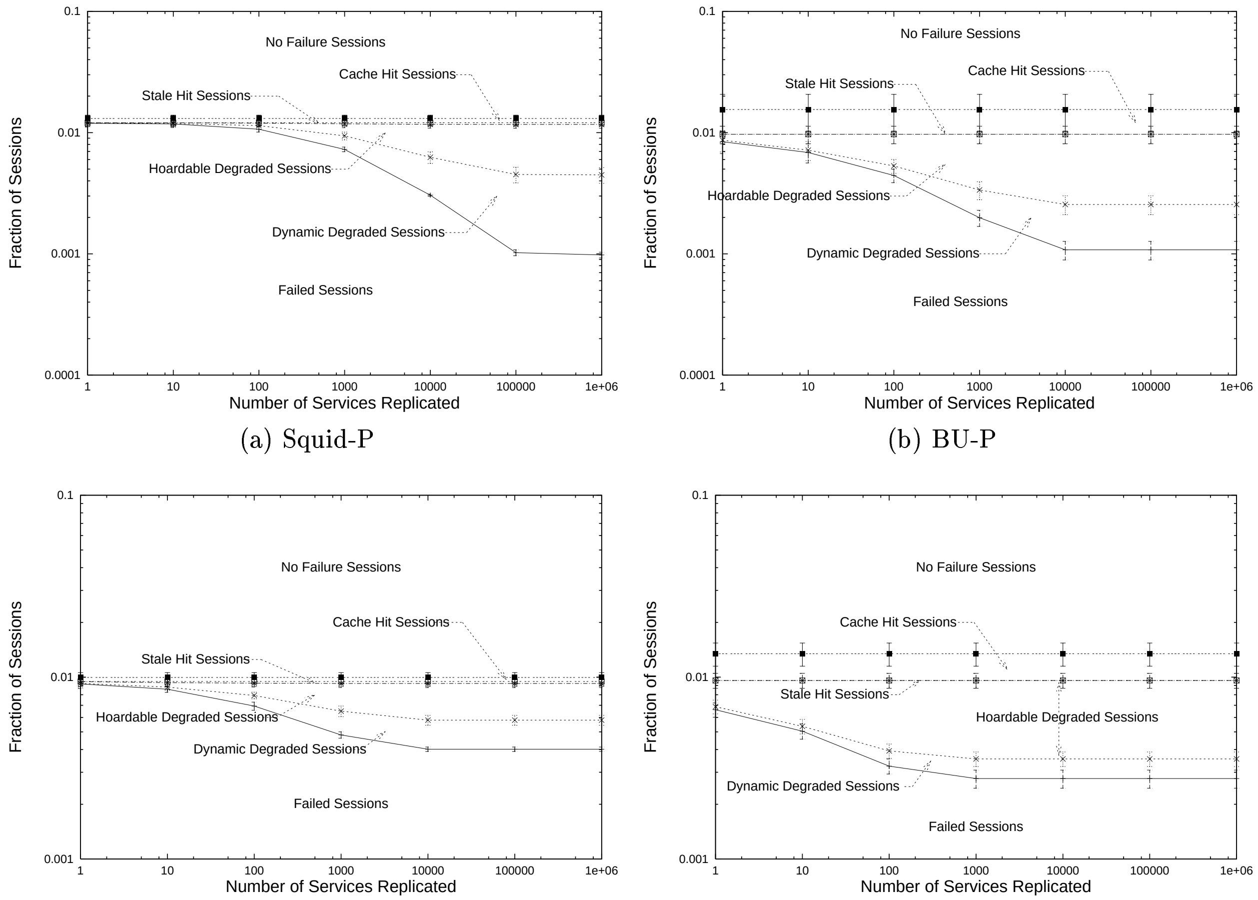
(a) Squid-P (b) BU-P

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

Figure 7: Session results as average network unavailability varies. Each region between two lines represents the fraction of sessions that can be handled by the specified technique.

proxy, or network. A key limiting factor, therefore, may be how many resources a cache can devote to each hosted service and how many services a cache can simultaneously host. Figure 8 shows session results when the simulated clients and proxies maintain only a finite number of local copies of prefetched services and mobile extensions with MFU policy to evict the rest; results for LRU replacement and exponentially decaying average MFU are similar but not shown. Graphs (a) and (b) show configurations for proxies shared by all clients in the trace; graphs (c) and (d) show per-client configurations. For all four workloads 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.

Then, we evaluate the sensitivity of different services to different client-independence techniques. In the previous set of experiments, we average across all web services to evaluate their availability improvements after applying unavailability masking techniques. We now explore the benefit that different individual services could receive. Figure 9 shows the cumulative distribution function of average unavailability improvements for the individual servers in the BU trace. We run our experiment 50 times with different random seeds for the failure model and track average session unavailability for each server across the runs. Each line shows the cumulative distribution of the fraction of services that receive at least the unavailability improvement indicated by the x-axis value. Note that the histograms for the different techniques were computed separately so that the



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

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

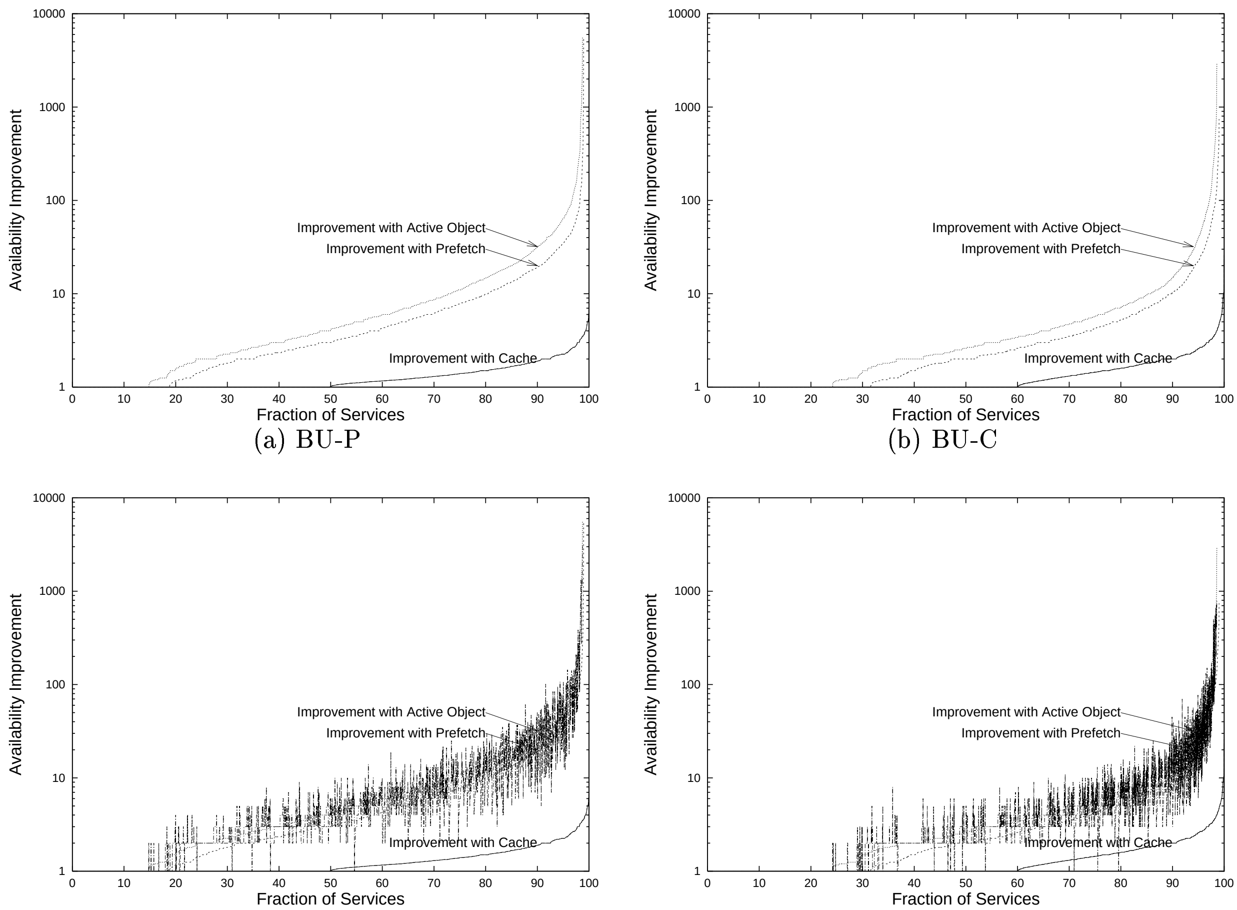
Figure 8: Session average unavailability v. number of cached service extensions. Each region between two lines represents the fraction of sessions that can be handled by the specified technique.

collection of machines at, say, the 90-percentile of cache improvement may be a different set of machines than at the 90-percentile of prefetching improvement.

In these traces, the bottom 15-25% of services receive little or no benefit from client independence techniques, about 50% improve by a factor between 1 and 10, and 10-20% receive improvements over a factor of 10. Furthermore, although caching alone generally provides little improvement, there are a few services for which caching can improve average unavailability by more than a factor of 2.

The primary factor determining the improvements provided to a service by the replication of active objects strategy is the popularity of the service. Services that are visited more often are proportionally less likely to suffer a compulsory miss during a network failure. This relationship is shown in graphs (c) and (d) in Figure 9, which overlays the number of visits in our trace to the service corresponding to the x-axis bucket.

Finally, we examine the maximum duration requirements for client-independent techniques to improve service availability to different degrees. Figure 10 shows the service improvement as we increase the maximum disconnection time that client-independent techniques support. For both proxy traces (Squid-P and BU-P), we see one order of magnitude improvement at 10,000 to 100,000 seconds. This means that in order to improve average service unavailability by an order of magnitude, the system must mask some failures lasting tens of thousands of seconds.



(a) BU-P (b) BU-C

(c) BU-P-checksum (d) BU-C-checksum

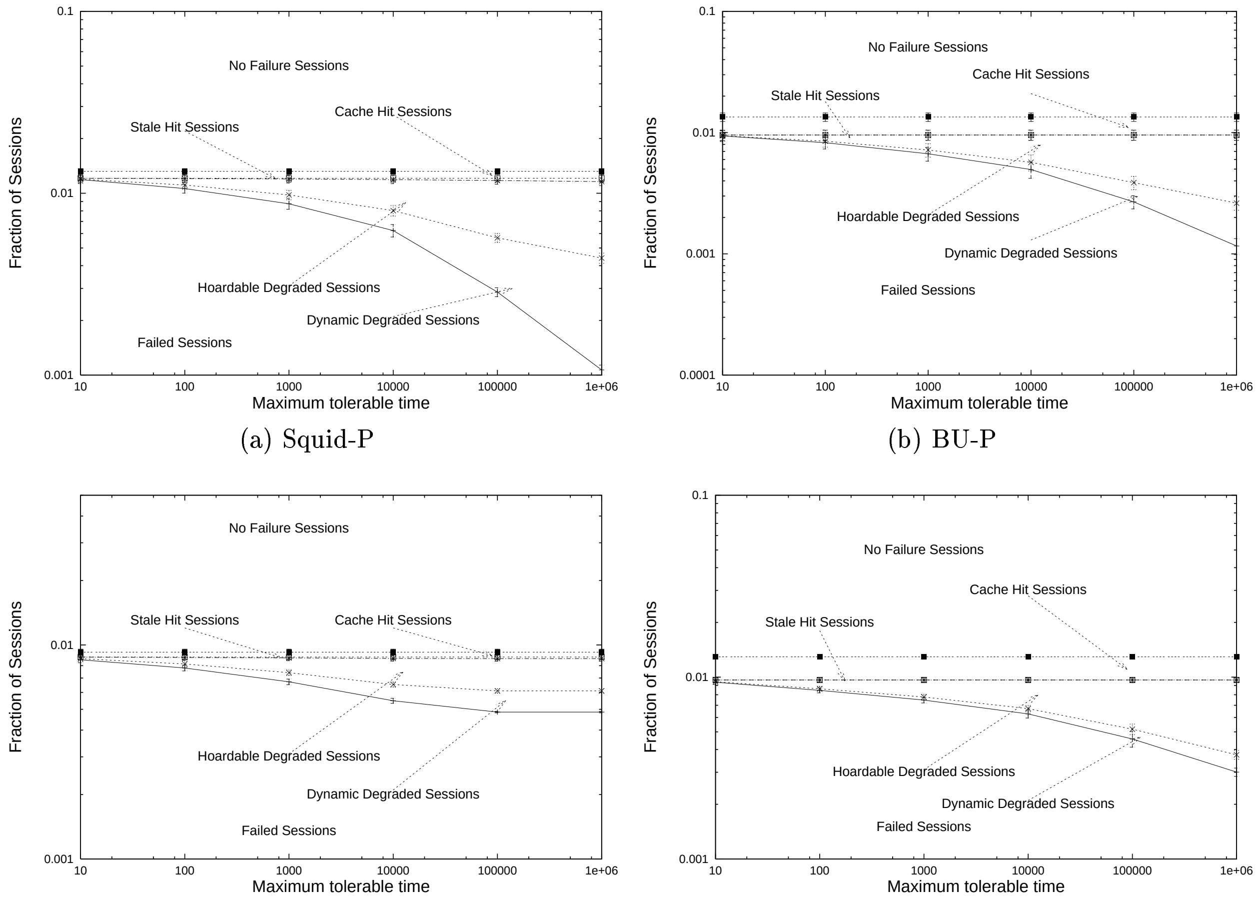
Figure 9: Availability improvement v. fraction of services.

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. Furthermore, some of the services may require large amounts of system resources to mask long failures. We examine the resource management challenges posed by such a workload in a separate study [5, 40].

4.2 Network routing

In this section, we evaluate strategies that route around network failures. To simplify the analysis, we classify strategies into two broad categories: (1) network re-routing and (2) server replication and selection. The following discussion states the type of failures masked by each strategy, how we model the strategies in our experiments, and the service availability improvements that the strategies yield.

1. **Re-routing.** Techniques of this category still send requests to the service's origin server, but they may use alternate routes when failures occur. Examples of re-routing techniques include dynamic routing [21] and overlay networks [33, 1]. In the terminology of this paper, these techniques address in-middle failures, but will be ineffective against near-source and near-destination failures.



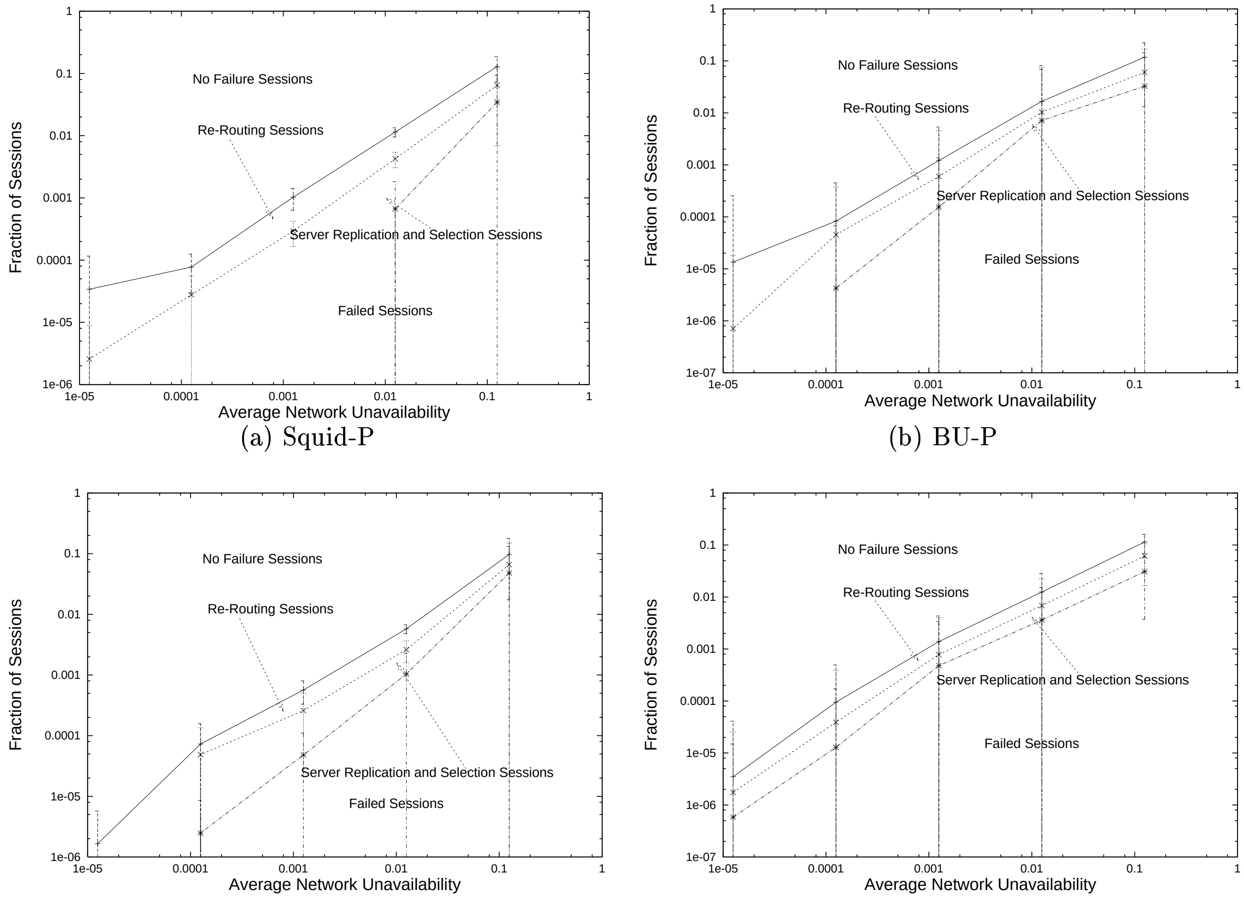
(a) Squid-P (b) BU-P

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

Figure 10: Session average unavailability v. maximum masking time required. Each region between two lines represents the fraction of sessions that can be handled by the specified technique.

2. **Server replication and selection.** This category of techniques directs requests to replicas of the origin servers when the origin servers are unreachable. Several file systems [32] and databases [25] provide replicated servers to handle failures in distributed environments. In the context of the Web, mirror sites with "manual failover", as well as replicated servers with anycast [3, 12, 46] can support server replication. This class of techniques can resolve near-destination and in-middle failures but is ineffective against near-source failures.

As in our analysis of client independence techniques, we abstract implementation details of routing-based techniques and focus on bounding improvements that they may provide. Several factors may limit these improvements in practice. For re-routing strategies, overheads include the failure detection time and route switching time. For server replication and selection, there are costs to maintain extra replicas and overheads to select alternative servers. These overheads vary for different implementations and may vary for different services (e.g. depending on failures, consistency, and semantics). Therefore, as with client-independence techniques, clients may experience sessions handled by re-routing or server replication as "degraded" with the significance of the deterioration varying on a service-by-service and implementation-by-implementation basis.



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

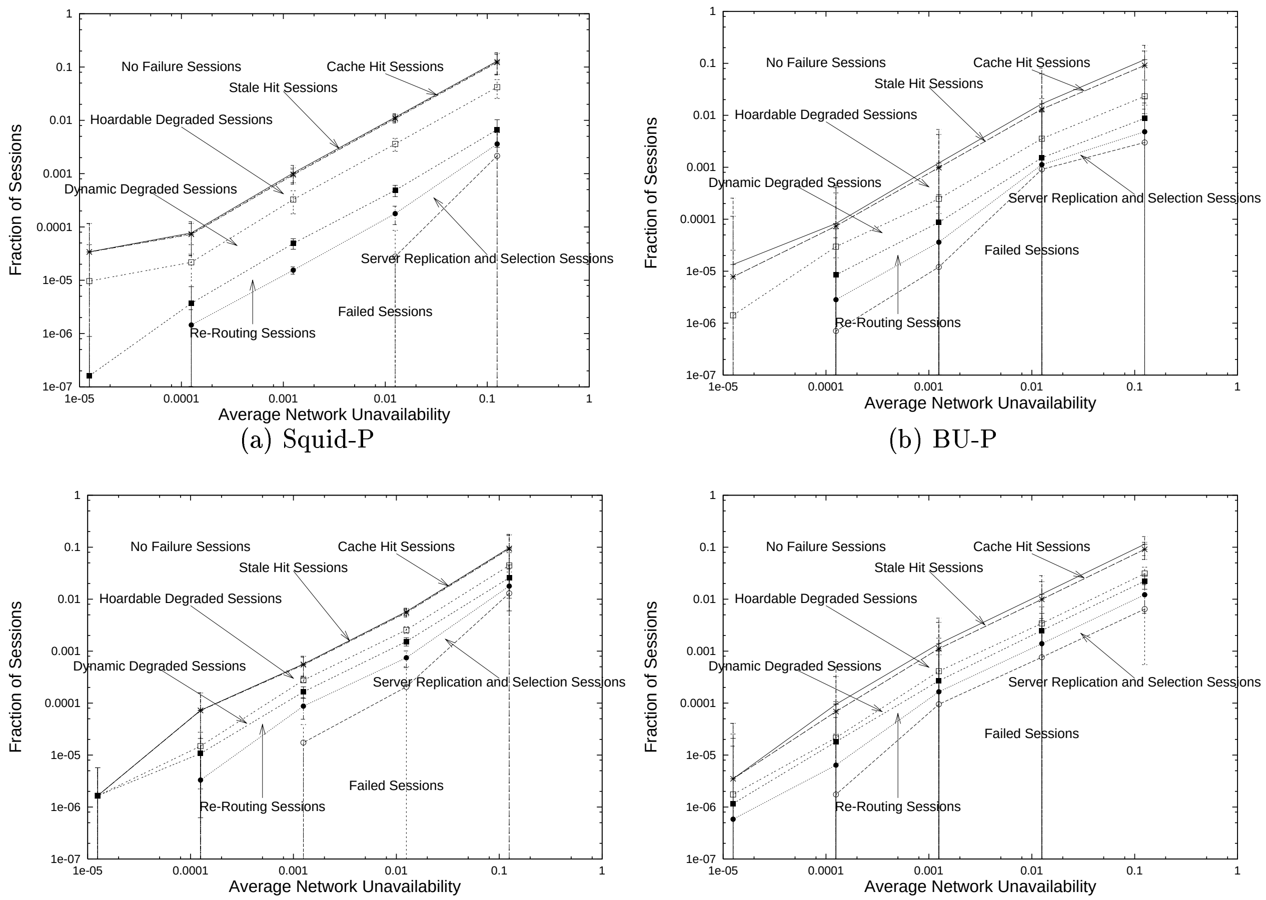
(c) Squid-C (d) BU-C

Figure 11: Session average unavailability v. network average unavailability. Each region between two lines represents the fraction of sessions that can be handled by the specified technique.

Workload and methodology. We use the same workloads and similar methodology as for the client-independence experiments. We group requests into sessions and classify each network disconnection by its location: near-source, in-middle, or near-destination. We run each experiment 25 times and plot the mean with 90% confidence intervals.

Results. In our first set of experiments, we vary the fraction of disconnections in each location category. These graphs are omitted due to space limits. Across a wide range of ratios, the findings are as expected: the fraction of unavailability events that each class of techniques can mask varies in proportion to the fraction of disconnections assigned to a particular location category. For example, when in-middle disconnections account for 50% of all disconnections, techniques that mask in-middle disconnections but not others can improve average unavailability by about a factor of two.

Given that the traces exhibit significant fractions of failures at each location, Amdahl's Law limits improvements from routing based strategies that do not address failures in all three locations. More study is needed to quantify the prevalence of near-source disconnections precisely, but the preliminary result of our study suggests that near-source failures account for at least 10%-20% of disconnections, probably limiting routing-based techniques to less than an order of magnitude improvement. As noted above, our methodology is likely to underestimate near-source failures.



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

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

Figure 12: Session average unavailability v. network average unavailability (combined techniques). Each region between two lines represents the fraction of sessions that can be handled by the specified technique.

Figure 11 shows the sensitivity of these results as we vary average network unavailability. As with the client-independence strategies, the relative improvements to average unavailability provided by these techniques remains stable over a wide range of underlying average network availabilities.

4.3 Combined Techniques

Client-independence techniques are limited by compulsory misses and installation time, and re-routing techniques are limited by near-source failures. Because these techniques fail in different circumstances, they may be combined to reduce average system unavailability.

For example, Figure 12 shows session average unavailability under a combined scheme in which network unavailability is masked by caching, prefetching, and active objects and in which prefetching and installation of active objects use anycast to access replicated servers. This combined approach thus masks all failures except near-source failures during prefetching or active object installation time. Figure 6 suggests that these results will be relatively insensitive to increases in *install_time*. Overall improvements in BU-P for this combined scheme are factors of 117, 100, 18.2, and 24.5 for average network unavailability probabilities of 0.0125%, 0.125%, 1.25%, and 12.5%, respectively. This relatively wide improvement range appears to be due to experimental variation magnified by the small number of unavailability events observed in the simulations. The graphs show qualitatively similar results for the other workloads.

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.

In this paper, we develop a network unavailability model and an evaluation strategy 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 [5].

We speculate that for most services, the path to high end-to-end availability 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.

Acknowledgments

We thank the anonymous *Transactions on Networking* reviewers for their helpful comments, particularly concerning the discussion of the availability models.

We thank the National Laboratory for Applied Network Research (NLANR) for making the Squid access logs available under National Science Foundation grants NCR-9616602 and NCR-9521745, the Oceans research group for making the BU traces available, the University of Washington Detour research group for making the uw traces available, and Vern Paxson for making the Paxson traces available.

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 Cisco, IBM, Novell, and Tivoli. Dahlin was also supported by an NSF CAREER award (CCR-9733842) and an Alfred P. Sloan Research Fellowship.

This article is an extended version of a paper by the same title, which appeared at the *Second Usenix Symposium on Internet Technologies and Systems*, March 2001.

References

[1] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proceedings of the Eightteenth ACM Symposium on Operating Systems Principles*, 2001.

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

[3] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, N. Shah, and Z. Fei. Application Layer Anycasting. In *Proc. IEEE INFOCOM'97*, 1997.

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

[5] B. Chandra, M. Dahlin, L. Gao, A. Khoja, A. Nayate, A. Razzaq, and A. Sewani. Resource management for scalable disconnected access to web services. In *10th International World Wide Web Conference*, May 2001.

[6] B. Chandra, M. Dahlin, L. Gao, and A. Nayate. End-to-end WAN Service Availability. In *Proc. of the Third USENIX Symposium on Internet Technologies and Systems*, 2001.

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

[8] R. D'Agostino and M. Stephens, editors. *Goodness-of-Fit Techniques.* Marcel Dekker, Inc., 1986.

[9] M. Dahlin, B. Chandra, L. Gao, A. Khoja, A. Nayate, A. Razzaq, and A. Sewani. Using Mobile Extensions to Support Disconnected Services. Technical Report TR-2000-20, University of Texas at Austin Department of Computer Sciences, June 2000.

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

[11] 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.

[12] Z. Fei, S. Bhattacharjee, E. Zegura, and M. Ammar. A Novel Server Selection Technique for Improving the Response Time of a Replicated Service. In *Proc. of IEEE Infocom*, March 1998.

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

[14] 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.

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

[16] 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.

[17] R. Jain. *The Art of Computer Systems Performance Analysis*, chapter 13, pages 179–200. Wiley, 1991.

[18] 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.

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

[20] M. Korupolu and M. Dahlin. Coordinated Placement and Replacement for Large-Scale Distributed Caches. In *Proc. of the 1999 IEEE Workshop on Internet Applications*, June 1999.

[21] K. R. Krishnan, R. Doverspike, and C. Pack. Improved Survivability with MultiLayer Dynamic Routing. *IEEE Communications Magazine*, 33(7), July 1995.

[22] 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.

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

[24] 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.

[25] A. Moissis. SYBASE replication server: A practical architechture for distributing and sharing corporate information. Technical report, SYBASE Inc, March 1994.

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

[27] National Institute of Standards and Technology. *Dataplot Reference Manual. NIST Handbook Number 148*, July 2001.

[28] 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.

[29] 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.

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

[31] 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.

[32] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere. Coda: A highly Available File System for a Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4), April 1990.

[33] 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.

[34] G. Snedecor and W. Cochran. *Statistical Methods.* Iowa State University Press, seventh edition, 1980.

[35] 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.

[36] 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.

[37] 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.

[38] K. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* Wiley, second edition, 2002.

[39] 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.

[40] A. Venkataramani, R. Kokku, and M. Dahlin. TCP-Nice: A mechanism for background transfers. In *OSDI02*, December 2002.

[41] A. Venkataramani, P. Yalagandula, R. Kokku, S. Sharif, and M. Dahlin. "potential costs and benefits of long-term prefetching for content-distribution". In *Proceedings of the 2001 Web Caching and Content Distribution Workshop*, June 2001.

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

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

[44] 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.

[45] 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.

[46] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using Smart Clients to Build Scalable Services. In *Proc. of the 1997 USENIX Technical Conf.*, January 1997.

[47] 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.