Chapter 7

Evaluation

In the previous four chapters of this dissertation, we discussed two complementary approaches to Web prediction: history-based prediction and content-based prediction. The next five chapters are primarily concerned with the evaluation of prefetching techniques. While there are various procedures for evaluating prediction methods, they are likely to be insufficient as they perform measurements in isolation from the system in which they will be embedded.

7.1 Introduction

This chapter is particularly concerned with the general task of evaluation. Accurate evaluation of systems and ideas is central to scientific progress. Evaluation, however, is usually more than a binary result – not just success or failure. Typically, an evaluation of performance is really that of a measure suited for comparison with other systems or variations of the same system.

However, in all evaluations there is some measure of uncertainty in the results (especially for comparison), depending on what variables could be controlled. For example, system A might provide a large improvement in performance over system B for some workload. But if that workload is not sufficiently representative of the workloads in which A or B might be applied, then the experiment may have failed to truly determine the better system.

This is a common charge against the use of benchmarks. Conversely, benchmark proponents would cite the need for a standard test so that the results of many systems can be compared. However, even when a standard benchmark, such as SPECweb96 [Sys] is used, the test environment needs to be identical. Performance on a benchmark

can be affected by many variables, as pointed out by Iyengar and Challenger [IC97]. This can include different hardware, software (such as operating system and networking stack), configuration (parameter tuning, logging, disk assignments and partitioning, etc.). Thus, it is often difficult to fairly compare benchmark results when published by different sources.

A related problem arises when a standard benchmark is the primary mechanism against which a system is evaluated. To show performance improvements, system architects will often tune their system specifically to the benchmark in question (e.g., optimizing memory cache design [LW94a]). Unless the benchmark is particularly representative of real-world workloads, such optimizations result in even less confidence in the overall applicability of the evaluation results.

Thus, a valid question is whether a perfect evaluation methodology is possible. In an ideal evaluation methodology, the test would provide:

- accuracy — the evaluation correctly measures the performance aspect of interest.
- repeatability — the same test performed at some other time would provide the same results.
- comparability — evaluation results of one system could be fairly compared to the results of another.
- applicability — the results of evaluation fairly reflect performance of the system in practice.

Finally, most systems do not exist in isolation. Regardless of whether a system or program interacts with people or other systems, those interactions may change when system performance changes. This consideration raises new questions. Sometimes these effects can be managed, as when the other systems are also under experimental control, in which case the performance of the system as a whole is being evaluated as changes within one subsystem are made. However, when they are not under experimental control (such as a system that interacts with others across the Internet, or are used by people), the workload could change as performance changes. For example, if response

time improves, inter-request times may drop and more requests are made. Likewise, if response time gets worse, requests may be canceled or skipped entirely.

In the remainder of this section, we consider some of the issues raised above, but with a focus on the domain of Web and proxy cache evaluation.

7.1.1 Accurate workloads

Historically, the typical workload in Web caching research is that of a log of real usage, either from a Web server, or from a Web proxy, as in Figure 7.1. However, it is also possible to collect logs by packet sniffing (as in [WWB96, MDFK97, GB97]) or by appropriately modifying the browser to perform the logging (e.g., [TG97, CBC95, CP95]).

Those logs are then replayed within a simulation to measure cache performance. Unfortunately, in many cases, the representativeness of the logs is never questioned. Some of the most commonly used traces are logs of Web servers in the mid-1990s. While indeed those logs were once representative of real-world activity, they are not likely to be so today. While Web traffic and the number of deployed Web servers and proxies have exploded, the public availability of real-world traces has apparently decreased because of privacy concerns.

Assuming the availability of a representative trace of activity, that trace will need to be applied to a simulation. If the concerns of side-effects of replaying Web requests (which we will address in Chapter 12) are ignored, the possibility exists to replay the Web requests across a live network. In either case, the chances are that the environment in which the trace is being re-played will not exactly match the environment under which the trace was captured. For example, network properties (such as the bottleneck bandwidth to the rest of the Internet) may have changed (perhaps even intentionally). This suggests, at minimum, that the performance of the re-enacted system cannot be compared to the system under which the trace was captured. It also brings up the potential (as discussed above) that the trace may not be representative of the new environment, or of the new system. If the performance as seen by the user changes, the user may alter the behavior from that which might have otherwise occurred.

```
893252015.307 14 <client-ip> TCP_HIT/200 227 GET
  http://images.go2net.com/metacrawler/images/transparent.gif - NONE/- image/gif
893252015.312 23 <client-ip> TCP_HIT/200 4170 GET
  http://images.go2net.com/metacrawler/images/head.gif - NONE/- image/gif
893252015.318 38 <client-ip> TCP_HIT/200 406 GET
  http://images.go2net.com/metacrawler/images/bg2.gif - NONE/- image/gif
893252015.636 800 <client-ip> TCP_REFRESH_MISS/200 8872 GET
  http://www.metacrawler.com/ - DIRECT/www.metacrawler.com text/html
893252015.728 355 <client-ip> TCP_HIT/200 5691 GET
  http://images.go2net.com/metacrawler/images/market2.gif - NONE/- image/gif
893252016.138 465 <client-ip> TCP_HIT/200 219 GET
  http://images.go2net.com/metacrawler/templates/tips/../../images/pixel.gif -
  NONE/- image/gif
893252016.430 757 <client-ip> TCP_REFRESH_HIT/200 2106 GET
  http://images.go2net.com/metacrawler/templates/tips/../../images/ultimate.jpg -
  DIRECT/images.go2net.com image/jpeg
```

Figure 7.1: Excerpt of a proxy log generated by Squid 1.1 which records the timestamp, elapsed-time, client, code/status, bytes, method, URL, client-username, peerstatus/peerhost and objecttype for each request. It is also an example of how requests can be logged in an order inappropriate for replaying in later experiments.

There are additionally some non-user actions that will change. Requests for embedded resources (images, sounds, animations) are made at the time that the browser has parsed that portion of the encompassing page, and so when page retrieval performance changes, the request time for embedded resources will also change.

Even if the environment and system were identical to the original, the characteristics of playback can cause evaluation concerns. At one level, a Web log represents just a sequence of requests. If the only goal is to simulate replacement policies on a finite cache of unchanging objects, that sequence could be replayed at any rate and still produce identical results. Playback at maximum speed can be found in the research literature. However, it would not be realistic in that Web objects in the real world do change and have particular expiration times. This implies that a slower playback might have a higher probability of requesting objects that have expired, and that a faster playback would generate more cache hits than are warranted. Changes in playback speeds also imply changes in browser performance, connectivity, and user think times between requests, and in general may result in overall performance differences because of higher or lower demands for resources such as bandwidth.

Similarly, request inter-arrival rates need to be maintained. While a fixed request

rate might be simpler, the original request arrival times provide a strong contribution to the variation in network and server loads experienced. In addition, interactive prefetching systems rely on the existence of such "thinking time" — the time between page requests — to provide time in which to prefetch the objects likely to be requested next.

The above discussion of workloads has concentrated on the use of captured workloads. One alternative is the use of artificial workloads. Such workloads are designed to capture the relevant properties of real-world workloads, such as request inter-arrival and recurrence rates. Past research [CP95, CBC95] suggests a heavy tailed distribution of thinking times with a mean of 30 seconds, which can be replicated in carefully created workloads. Artificial workloads provide the advantage (and disadvantage) of being able to generate workloads that do not arise in practice. This is helpful when stress-testing a system for anticipated workloads, but also opens questions for evaluation. For example, even advanced artificial workloads (e.g., those in Web Polygraph [Rou02]) are unable to replicate the Web page relationships that the hypertext links of the Web provide for real datasets.

7.1.2 Accurate evaluation

As mentioned previously, accurate evaluation requires that the evaluation mechanism correctly measures the performance aspect of interest. This is sometimes overlooked in favor of evaluating metrics that are available. For example, early caching research often favored reports of increasing object hit rates. However, hit rates were never the real aspect of interest. Instead, they were a substitute metric for the real interests — reductions in bandwidth usage and improvements in user-perceived response times. In this case, byte hit rates (or better, a measure of all bytes received and generated by a system) would provide a real metric of byte savings. Similarly, user-perceived latencies and response times require significantly more careful evaluation of a more detailed system.

Finally, since Web systems are typically deployed for use by people, a human study

may be required. For example, if the goal is strictly to measure reduced bandwidth usage, the change in user-perceived latency associated with caching usage might prevent accurate measurement. Likewise, measures such as user-perceived response times are still one step removed — user satisfaction might be the ultimate goal, whose measurement would require a user study.

7.1.3 Accurate simulation

A prerequisite of accurate evaluation is accurate simulation. In this case, the simulation must not only be correct (i.e., without errors) but appropriately model various factors that contribute to desired performance. In Web caching research, desired performance includes the three major claimed benefits to caching on the Web: reduced bandwidth utilization, reduced server loads, and reduced client-side response times. Web bandwidth utilization can be modeled effectively with correct simulation, but the latter two may require extra care. To accurately model server loads, it may be necessary to model internal operating system and hardware characteristics (such as connection queue lengths and disk seek times). For accurate client-side response times, it may be necessary to model transport layer characteristics such as TCP slow start effects and user behaviors such as request cancellations (e.g., [FCD+99, CB98]). In both cases, accurate modeling of connection characteristics (multiple parallel connections, persistent connections) will be essential. These effects will need to be integrated into what might otherwise have been caching-only simulators, if accurate simulation of response times are to be achieved. Likewise, for prefetching simulators, the costs of prefetching must be included, such as time and costs for receiving server hints.

7.2 Cache Evaluation Methodologies

Since proxy caches are increasingly used around the world to reduce bandwidth requirements and alleviate delays associated with the World-Wide Web, this section describes a space of proxy cache evaluation methodologies and places a sampling of research within that space. The primary contributions of this section are threefold: 1) definition and

	Workload Source	
Algorithm Implementation	artificial	captured logs
simulated systems/network	A1	A2
real systems/isolated network	B1	B2

Table 7.1: A space of traditional evaluation methodologies for Web systems.

description of the space of evaluation techniques; 2) appraisal of the different methods within that space; and 3) a survey of cache evaluation techniques from the research literature.

It is useful to be able to assess the performance of proxy caches, both for consumers selecting the appropriate system for a particular situation and also for developers working on alternative proxy mechanisms. By evaluating performance across all measures of interest, it is possible to recognize drawbacks with particular implementations. For example, one can imagine a proxy with a very large disk cache that provides a high hit rate, but because of poor disk management it noticeably increases the client-perceived response times. One can also imagine a proxy that achieves a high hit rate by 1) caching images only for a very short time, which frees storage for use by other objects, and 2) prefetching the inline (embedded) images for the page currently requested. This would allow it to report high request hit rates, but not perform well in terms of bandwidth savings. Finally, it is possible to select a proxy based on its performance on a workload of requests generated by dialup users and have it perform unsatisfactorily as a parent proxy for a large corporation with other proxies as clients. These examples illustrate the importance of appropriate evaluation mechanisms for proxy systems.

7.2.1 The space of cache evaluation methods

As mentioned earlier, the most commonly used cache evaluation method is that of simulation on a benchmark log of object requests. In such a system, the byte and page hit rate savings can be calculated as well as estimates of latency improvements. In a few cases, an artificial dataset with the necessary characteristics (appropriate average and median object sizes, or similar long-tail distributions of object sizes and object repetitions) is used. More commonly, actual client request logs are used since they

	Workload Source		
Algorithm Implementation	artificial	captured logs	current requests
simulated systems/network	A1	A2	A3
real systems/isolated network	B1	B2	B3
real systems/real network	C1	C2	C3

Table 7.2: An expanded space of evaluation methodologies for Web systems.

arguably better represent likely request patterns and include exact information about object sizes and retrieval times.

We characterize Web/proxy evaluation architectures along two dimensions: the source of the workload used and form of algorithm implementation. Table 7.1 shows the traditional space with artificial versus captured logs and simulations versus implemented systems. With the introduction of prefetching proxies, the space needs to be expanded to include implemented systems on a real network connection, and in fact, evaluation of an implemented proxy in the field would necessitate another column, that of live requests. Thus, the expanded view of the space of evaluation methodologies can be found in Table 7.2 and shows that there are at least three categories of data sources for testing purposes, and that there are at least three forms of algorithm evaluation. While it may be possible to make finer distinctions within each area, such as the representativeness of the data workload or the fidelity of simulations, the broader definitions used in the expanded table form a simple yet useful characterization of the space of evaluation methodologies.

For the systems represented in each area of the space, we find it worthwhile to consider how the desired qualities of Web caching (reduced bandwidth requirements, server loads and improved response times) are measured. As will be seen below, each of the areas of the space represents a trade-off of desirable and undesirable features. In the rest of this section we point out the characteristic qualities of systems in each area of the space and list work in Web research that can be so categorized.

7.2.2 Methodological appraisal

In general, both realism and complexity increase as one moves diagonally downward and to the right in the evaluation methodology space. Note that only methods in areas A1,

A2, B1 and B2 are truly replicable, since live request stream samples change over time, as do the availability and access characteristics of hosts via a live network connection.

Simulated systems. Simulation is the simplest mechanism for evaluation as it does not require full implementation. However, simulating the caching algorithm requires detailed knowledge of the algorithms which is not always possible (especially for commercial implementations). Even then, typical simulation cannot accurately assess document retrieval delays [WA97].

It is also impossible to accurately simulate caching mechanisms that prefetch on the basis of the contents of the pages being served (termed *content-based prefetching*), such as those in CacheFlow [Cac02b], Wcol [CY97], and WebCompanion [Kle99] since they need at least to have access to the links within Web pages — something that is not available from server logs. Even if page contents were logged (as in [CBC95, MDFK97, Dav02a]), caches that perform prefetching may prefetch objects that are not on the user request logs and thus have unknown characteristics such as size and Web server response times.

Real systems/isolated networks. In order to combat the difficulties associated with a live network connection, measurement techniques often eliminate the variability of the Internet by using local servers and isolated networks, which may generate unrealistic expectations of performance. In addition to not taking into consideration current Web server and network conditions, isolated networks do not allow for retrieval of current content (updated Web pages).

Real systems and networks. Because the state of the Internet changes continuously (i.e., web servers and intermediate network connections may be responsive at one time, and sluggish the next), tests on a live network are generally not replicable. Additionally, to perform such tests the experiment requires reliable hardware, robust software, and a robust network connection to handle the workload applied. Finally, connection to a real network requires compatibility with, and no abuse of, existing systems (e.g., one cannot easily run experiments that require changes to standard httpd servers or experimental extensions to TCP/IP).

Artificial workloads. Synthetic traces are often used to generate workloads that

have characteristics that do not currently exist to help answer questions such as whether the system can handle twice as many requests per second, or whether caching of non-Web objects is feasible. However, artificial workloads often make significant assumptions (e.g., that all objects are cacheable, or that requests follow a particular distribution) which are necessary for testing, but not necessarily known to be true.

Captured logs. Using actual client request logs can be more realistic than artificial workloads, but since on average the lifetime of a Web page is short (less than two months [Wor94, GS96, Kah97, DFKM97]), any captured log loses its value quickly as more references within it are no longer valid, either by becoming inaccessible or by changing content (i.e., looking at a page a short time later may not give the same content). In addition, unless the logs are recorded and processed carefully, it is possible for the logs to reflect an inaccurate request ordering as the sample Squid logs show in Figure 7.1. Note that the request for the main page follows requests for three subitems of that main page. This is because entries in the log are recorded when the request is completed, and the timestamp records the time at which the socket is closed. Finally, proxy cache trace logs are inaccurate when they return stale objects since they may not have the same characteristics as current objects. Note that logs generated from non-caching proxies or from HTTP packet sniffing may not have this drawback. In other work [Dav99c], we further examine the drawbacks of using standard trace logs and investigate what can be learned from more complete logs that include additional information such as page content.

Current request stream workloads. Using a live request stream produces experiments that are not reproducible (especially when paired with live hardware/networks). Additionally, the test hardware and software may have difficulties handling a high real load. On the other hand, if the samples are large enough and have similar characteristics, an argument for comparability might be made.

7.2.3 Sampling of work in methodological space

In this section we place work from the research literature into the evaluation space described above. Note that inclusion/citation in a particular area does not necessarily

indicate the main thrust of the paper mentioned.

Area A1: Simulations using artificial workloads. It can be difficult to characterize a real workload sufficiently to be able to generate a credible artificial workload. This, and the wide availability of a number of proxy traces, means that relatively few researchers attempt this kind of research:

- Jiang and Kleinrock [JK97, JK98] evaluate prefetching mechanisms theoretically (area A1) but also on a limited trace set (area A2).
- Manley, Seltzer and Courage [MSC98] propose a Web benchmark which generates realistic loads to focus on measuring response times. This tool would be used to perform experiments somewhere between areas A1 and A2 as it uses captured logs to build particular loads on request.
- Tewari *et al.* [TVDS98] use synthetic traces for their simulations of caching continuous media traffic.

Area A2: Simulations using captured logs. Simulating proxy performance is much more popular than one might expect from the list of research in area A1. In fact, the most common mechanism for evaluating an algorithm's performance is simulation over one or more captured traces.

- Cunha *et al.* [CBC95], Partl [Par96], Williams *et al.* [WAS+96], Gwertzman and Seltzer [GS96], Cao and Irani [CI97], Bolot and Hoschka [BH96], Gribble and Brewer [GB97], Duska, Marwood and Feeley [DMF97], Caceres *et al.* [CDF+98], Niclausse, Liu and Nain [NLN98], Kurcewicz, Sylwestrzak and Wierzbicki [KSW98], Scheuermann, Shim and Vingralek [SSV97, SSV98, SSV99], and Zhang *et al.* [ZIRO99] all utilize trace-based simulation to evaluate different cache management algorithms.
- Trace-based simulation is also used in evaluating approaches to prefetching, such as Bestavros's server-side speculation [Bes95], Padmanabhan and Mogul's persistent HTTP protocol along with prefetching [PM96], Kroeger, Long and Mogul's

calculations on limits to response time improvement from caching and prefetching [KLM97], Markatos and Chronaki's object popularity-based method [MC98], Fan, Cao and Jacobson's response time reduction to low-bandwidth clients via prefetching [FJCL99] and Crovella and Barford's prefetching with simulated network effects [CB98].

- Markatos [Mar96] simulates performance of a Web server augmented with a main memory cache on a number of public Web server traces.

- Mogul *et al.* [MDFK97] use two large, full content traces to evaluate delta-encoding and compression methods for HTTP via calculated savings (area A2) but also performed some experiments to include computational costs on real hardware with representative request samples (something between areas B1 and B2).

Area A3: Simulation based on current requests. We are aware of no published research that could be categorized in this area. The algorithms examined above in areas A1 and A2 do not need the characteristics of live request streams (such as contents rather than just headers of HTTP requests and replies). Those researchers who use live request streams all use real systems of some sort (as will be seen below).

Area B1: Real systems on an isolated network using an artificial dataset. A number of researchers have built tools to generate workloads for Web systems in a closed environment. Such systems make it possible to generate workloads that are uncommon in practice (or impossible to capture) to illuminate implementation problems. These include:

- Almeida, Almeida and Yates [AAY97a] propose a Web server measurement tool (Webmonitor), and describe experiments driven by an HTTP load generator.

- Banga, Douglis and Rabinovich [BDR97] use an artificial workload with custom client and server software to test the use of transmitting only page changes from a server proxy to a client proxy over a slow link.

- Almeida and Cao's Wisconsin Proxy Benchmark [AC98b, AC98a] uses a combination of Web client and Web server processes on an isolated network to evaluate

proxy performance.

- While originally designed to exercise Web servers, both Barford and Crovella's SURGE [BC98a] and Mosberger and Jin's httperf [MJ98] generate particular workloads useful for server and proxy evaluation.

- The CacheFlow [Cac00] measurement tool was designed specifically for areas C1 and C2, but could also be used on an isolated network with an artificial dataset.

Area B2: Real systems on an isolated network using captured trace logs. Some of the research projects listed in area B1 (using artificial logs) may be capable of using captured trace logs. In addition, we place the following here:

- In evaluating their Crispy Squid, Gadde, Chase and Rabinovich [GCR98] describe the tools and libraries called Proxycizer. These tools provide a trace-driven client and a characterized Web server that surround the proxy under evaluation, much like the Wisconsin Proxy Benchmark.

Area B3: Real systems on an isolated network using current requests. Like area A3 which used simulation, we found no research applicable to this area. Since live requests can attempt to fetch objects from around the globe, it is unlikely to be useful for testing forward proxies within an isolated network. However, we suggest that reverse proxies could be tested internally under this model.

Area C1: Real systems on a live network using an artificial dataset. Some of the research projects in area B1 (e.g., CacheFlow's measurement tool) may be designed for the use of a live network connection. The primary restriction is the use of real, valid URLs that fetch over the Internet rather than particular files on a local server.

Area C2: Real systems on a live network using captured logs. With captured logs, the systems being evaluated are as realistically operated as possible without involving real users as clients. In addition to those listed below, some of the tools from area B1 may be usable in this type of experiment.

- Wooster and Abrams [Woo96, WA97] report on evaluating multiple cache replacement algorithms in parallel within the Harvest cache, both using URL traces and online (grid areas C2 and C3, respectively) but the multiple replacement algorithms are within a single proxy. Wooster also describes experiments in which a client replayed logs to multiple separate proxies running on a single multiprocessor machine.

- Maltzahn and Richardson [MR97] evaluate proxies with the goal of finding enterprise-class systems. They test real systems with a real network connection and used carefully selected high load-generating trace logs.

- Liu *et al.* test the effect of network connection speed and proxy caching on response time using public traces [LAJF98] on what appears to be a live network connection.

- Lee *et al.* [LHC+98] evaluate different cache-to-cache relationships using trace logs on a real network connection.

Area C3: Real systems on a live network using the current request stream. In many respects, this area represents the strongest of all evaluation methodologies. However, most real installations are not used for the comparison of alternate systems or configurations, and so we report on only a few research efforts here:

- Chinen and Yamaguchi [CY97] described and evaluated the performance of the Wcol proxy cache on a live network using live data, but do not compare it to any other caching systems.

- Rousskov and Soloviev [RS98] evaluated performance of seven different proxies in seven different real-world request streams.

- Cormack [Cor98] described performance of different configurations at different times of a live Web cache on a live network.

- Later in Chapter 10 we will describe the Simultaneous Proxy Evaluation (SPE) architecture that compares multiple proxies on the same request stream. While

originally designed for this area with a live request stream and live network connection, it can also be used for replaying captured or artificial logs on isolated or live networks (areas B1, B2, C1, and C2).

7.3 Recommendations for Future Evaluations

While almost every researcher uses some kind of evaluation in their work, there are a number of pitfalls that may be encountered. The better papers will point out the drawbacks of their methodology, but not all do. In this section, we consider many of the typical mistakes that can be found in existing caching research.

Using an inappropriate trace. Some publicly available traces are more than a few years old[1] and thus may no longer be representative of current network traffic. In some cases, however, these are the only ones available with the desired characteristics. Ideally traces would be large, representative of the desired traffic, and be publicly accessible for others to examine and possibly reconstruct tests.

Hiding effects of changes to subsystem. When comparing an artificial or otherwise modified environment to a real-world trace, there are a multitude of factors for which accounting must be made. An example of this is the use of a proxy trace in replaying through a Web browser. Unless specially configured, the browser may use its built-in cache and no longer generate as many external requests as found in the original trace.

Ignoring the effect of connection caching. Connection caching [CKZ99] can be a significant effect of the use of persistent connections in proxies. When a proxy is introduced to an environment, the effects of persistent connections and the caching of connections by the proxy should be considered.

Ignoring the effect of differences between HTTP/1.0 and HTTP/1.1. HTTP/1.1 [FGM+99] introduced a number of improvements over HTTP/1.0 [KMK99], including persistent connections as default behavior and increased support for caching.

[1] Unfortunately, many of the traces used in this dissertation have also gotten old. We encourage ISPs, content providers, and hosting providers to make traces of Web usage publicly available for analysis (after suitable anonymization).

The use of a trace containing one kind of traffic for replaying in another environment should not be made blindly. HTTP traffic measurements today are likely to see both protocols in widespread use.

Ignoring TCP effects on HTTP performance. TCP introduces particular effects for the relatively short-lived HTTP connections, including costs associated with slow start and connection establishment [PK98, SCJO01].

Ignoring canceled requests. While relatively few public traces include information about canceled requests, some studies (e.g., [FCD+99]) have revealed the relatively high presence of such activity. Inappropriate use of such requests in logs (such as fully retrieving them or assuming the sizes reflect a new, smaller object) may distort actual performance of a system.

Ignoring effects of replaying trace. When a trace is replayed, it really does not faithfully replicate what users would do or see in a new environment. For example, the network and destination hosts may have changed content or accessibility. More importantly, the user is likely to have acted differently — when responses are made faster, the user might make requests faster. Or vice versa — if a response was much slower, the user might give up and cancel the request and move on to something else.

Ignoring real timing issues. For those researchers interested in latency and response time effects, it is not sufficient to measure performance in terms of round trip times. As mentioned above, real networking effects may be needed to get accurate estimates of performance.

Obscuring pertinent details of test infrastructure. For example, the use of old proxy cache software (e.g., the CERN proxy [LNBL96]) often implies a lack of support for HTTP/1.1 and possibly persistent connections under HTTP/1.0 as well. Therefore, such details are important to disclose.

Ignoring concerns of freshness in a trace. Since most Web usage traces do not include information containing a response's expiration, a common approach is to test for changes in response size to determine a change in the underlying object. While this is not ideal, it is better than simpler (but wrong) approaches that only check for recurrence of the object request. Likewise, simulations that hold data in a cache forever

are not realistic — many objects should not be cached at all, or only for a short period.

Ignoring other cacheability information. In addition to the concerns mentioned above, there are many factors involved in determining cacheability. For example, under HTTP/1.0, the presence of a cookie can make a response uncacheable. Likewise, many researchers mistakenly assume that uncacheable responses can be determined from the format of the URL (e.g., the inclusion of a '?') when under HTTP/1.1, no such *a priori* relationship exists.

Ignoring DNS issues when building a prototype. The domain name system can be a significant source of slow response times [CK00, CK01a, STA01]. Thus, DNS effects must be considered; for example, prefetching DNS can provide real savings.

Hiding or ignoring bad data. Some researchers choose to drop requests believed to be dynamic from their traces or to eliminate error responses. As long as it is well documented, the practice may be acceptable, but a better, more general approach would be to incorporate all requests into the model.

Vague or missing latency calculations. Since caching and prefetching can provide significant benefits to user-perceived response times, it is imperative that latency and response time calculations be explicitly described.

The concerns presented above are not meant to be exhaustive. Instead, they simply represent many of those raised by the author while reviewing publications cited in this dissertation.

7.4 Summary

Evaluation is a significant concern, both for consumers and for researchers. Objective measurements are essential, as are comparability of measurements from system to system. Furthermore, it is important to eliminate variables that can affect evaluation.

Selection of an appropriate evaluation methodology depends on the technology being tested and the desired level of evaluation. Some technologies, such as protocol extensions, are often impossible to test over a live network because other systems do not support it. Similarly, if the goal is to find bottlenecks in an implementation, one

may want to stress the system with very high synthetic loads since such loads are unlikely to be available in captured request traces. In fact, this reveals another instance in which a live network and request stream should not be used — when the goal of the test is to drive the system into a failure mode to find its limits.

In general, we argue for the increased believability of methodologies that are placed further down and to the right in the evaluation space when objectively testing the successful performance of a Web caching system. If the tested systems make decisions based on content, a method from the bottom row is likely to be required. When looking for failure modes, it will be more useful to consider methodologies near the upper left of the evaluation space.

This survey provides a sampling of some of published Web caching research and presents one of potentially many spaces of evaluation methodologies. In particular, we haven't really considered aspects of testing cache hierarchies and inter-cache communication [CDN+96, Nat02, Dan98, CC95, CC96, ZFJ97, GRC97, GCR97, FCAB98, RCG98, WC97b, WC97a, RW98, VR98, VW00], cache consistency, or low-level protocol issues such as connection caching which are significant in practice [CDF+98].

In this chapter we have described many concerns for accurate, repeatable, comparable, and applicable evaluation methodologies. We have additionally described a space of evaluation methodologies and shown where a sample of research efforts fall within it. By considering the space of appropriate evaluation methodologies, one can select the best trade-off of information provided, implementation costs, comparability, and relevance to the target environment.

In Chapter 8 we will describe a simulator that can be used to develop and evaluate some types of caching and prefetching approaches. Subsequently, in Chapter 9 we will use that simulator to combine predictions from multiple sources and report on the efficacy of the approach. As we described in this chapter, simulation has limitations as well, and so in Chapters 10 and 11, we propose and develop a proxy cache evaluation architecture and test it with real systems. Finally, Chapter 12 concludes the dissertation by looking ahead to the next steps that will be necessary to make wide-spread prefetching possible on the Web.