# **EtE**: Passive End-to-End Internet Service Performance Monitoring *

Yun Fu†, Ludmila Cherkasova‡, Wenting Tang‡, and Amin Vahdat†

†Dept of Computer Science, Duke University
Durham, NC 27708, USA
email: {fu,vahdat}@cs.duke.edu

‡Hewlett-Packard Laboratories
1501 Page Mill Road,Palo Alto, CA 94303, USA
email: {lucy_cherkasova, wenting_tang}@hp.com

**Abstract.** *This paper presents, EtE monitor, a novel approach to measuring web site performance. Our system passively collects packet traces from a server site to determine service performance characteristics. We introduce a two-pass heuristic method and a statistical filtering mechanism to accurately reconstruct different client page accesses and to measure performance characteristics integrated across all client accesses. Relative to existing approaches, EtE monitor offers the following benefits: i) a breakdown between the network and server overhead of retrieving a web page, ii) longitudinal information for all client accesses, not just the subset probed by a third party, iii) characteristics of accesses that are aborted by clients, and iv) quantification of the benefits of network and browser caches on server performance. Our initial implementation and performance analysis across two sample sites confirm the utility of our approach.*

## 1 Introduction

Today, Internet services are delivering a large array of business, government, and personal services. Similarly, mission critical operations, related to scientific instrumentation, military operations, and health services, are making increasing use of the Internet for delivering information and distributed coordination. However, the best effort nature of Internet data delivery, changing client and network connectivity characteristics, and the highly complex architectures of modern Internet services make it very difficult to understand the performance characteristics of Internet services. In a competitive landscape, such understanding is critical to continually evolving and engineering Internet services to match changing demand levels and client populations.

*This work was originated and largely completed while Y. Fu worked at HPLabs during the summer 2001 and supported in part by research grant from HP. A. Vahdat and Y. Fu are supported in part by the National Science Foundation (EIA-9972879). A. Vahdat is also supported by an NSF CAREER award (CCR-9984328).

Currently, there are two popular techniques for benchmarking the performance of Internet services. The first approach, active probing [13, 17, 23, 19], uses machines from fixed points in the Internet to periodically request one or more URLs from a target web service, record end-to-end performance characteristics, and report a time-varying summary back to the web service. The second approach, web page instrumentation [8, 10, 2, 20], associates code (e.g., JavaScript) with target web pages. The code, after being downloaded into the client browser, tracks the download time for individual objects and reports performance characteristics back to the web site.

In this paper, we present a novel approach to measuring web site performance called EtE monitor. Our system passively collects network packet traces from the server site to enable either offline or online analysis of system performance characteristics. Using two-pass heuristics and statistical filtering mechanisms, we are able to accurately reconstruct individual page composition without parsing HTML files or obtaining out-of-band information about changing site characteristics. Relative to existing techniques, EtE monitor offers a number of benefits:

- Our system can determine the breakdown between the server and network overhead associated with retrieving a web page. This information is necessary to understand where performance optimizations should be directed, for instance to improve server-side performance or to leverage existing content distribution networks (CDNs) to improve network locality.

- EtE monitor tracks all accesses to web pages for a given service. Many existing techniques are typically restricted to a few probes per hour to URLs that are pre-determined to be popular. Our approach is much more agile to changing client access patterns. What real clients are accessing determines the performance that EtE monitor eval-

uates. Finally, given the Zipf popularity of service web pages [1], our approach is able to track the characteristics of the heavy tail that often makes up a large overall portion of web site accesses.

- Given information on all client accesses, clustering techniques [15] can be utilized to determine network performance characteristics by network region or autonomous system. System administrators can use this information to determine which content distribution networks to partner with (depending on their points of presence) or to determine multi-homing strategies with particular ISPs.

- EtE monitor captures information on page requests that are manually aborted by the client, either because of unsatisfactory web site performance or specific client browsing patterns (e.g., clicking on a link before a page has completed the download process). Existing techniques cannot model user interactions in the case of active probing or miss important aspects of web site performance such as TCP connection establishment in the case of web page instrumentation.

- Finally, EtE monitor is able to determine the actual benefits of both browser and network caches. By learning the likely composition of individual web pages, our system can determine when certain embedded objects of a web page are not requested and conclude that those objects were retrieved from some cache in the network.

This paper presents the architecture and implementation of our prototype EtE monitor. It also highlights the benefits of our approach through an evaluation of the performance of two sample network services using EtE monitor. Overall, we believe that detailed performance information will enable network services to dynamically react to changing access patterns and system characteristics to best match client QoS expectations. Depending on the architecture of the system, a front end "Layer-7" switch [18] could redirect requests for particular objects to a smaller or larger set of back-end machines based on observed performance summaries. Similarly, performance characteristics across multiple services being served from a single hosting center can be used to allocate resources to competing services to, for example, maximize aggregate throughput or to maintain higher-level service level agreements [4]. Sites may also use performance information to dynamically adjust system consistency [25] or content fidelity [3] with the goal of meeting target levels of performance.

The rest of this paper is organized as follows. In the next section, we survey existing techniques and products and discuss their merits and drawbacks. Section 3 outlines the EtE monitor architecture, with additional details in Sections 4-6. In Section 7, we present the results of two performance studies, which have been performed to test and validate EtE monitor and its approach. Section 8 presents two specially designed experiments to validate the accuracy of EtE monitor performance measurements and its page access reconstruction power. We discuss the limitations of the proposed technique in Section 9 and present our conclusions and future work in Section 10.

**Acknowledgments:** Both the tool and the study would not have been possible without generous help of our HP colleagues: Mike Rodriquez, Steve Yonkaitis, Guy Mathews, Annabelle Eseo, Peter Haddad, Bob Husted, Norm Follett, Don Reab, and Vincent Rabiller. Their help is highly appreciated. Our special thanks to Claude Villermain who helped to identify and to correct a subtle bug for dynamic page reconstruction. We would like to thank the anonymous referees for useful remarks and insightful questions, and our shepherd Jason Nieh for constructive suggestions to improve the content and presentation of the paper.

## 2 Related Work

A number of companies use active probing techniques to offer measurement and testing services today, including Keynote [13], NetMechanic [17], Software Research [23], and Porivo Technologies [19]. Their solutions are based on periodic polling of web services using a set of geographically distributed, synthetic clients. In general, only a few pages or operations can typically be tested, potentially reflecting only a fraction of all user's experience. Further, active probing techniques cannot typically capture the potential benefits of browser and network caches, in some sense reflecting "worst case" performance. From another perspective, active probes come from a different set of machines than those that actually access the service. Thus, there may not always be correlation in the performance/reliability reported by the service and that experienced by end users. Finally, it is more difficult to determine the breakdown between network and server-side performance using active probing, making it more difficult for customers to determine where best to place their optimization efforts.

Another popular approach is to embed instrumentation code with web pages to record access times and report statistics back to the server. For instance, WTO (Web Transaction Observer) from HP OpenView suite [8] uses JavaScript to implement this functionality. With additional web server instrumentation and cookie techniques, this product can record the server processing time for a request, enabling a breakdown between server and network processing time. A number of other products and proposals [10, 2, 20] employ similar techniques. Relative to our approach, web page instrumentation can also capture end-to-end performance information from real clients, except connection establishment times (po-

tentially an important aspect of overall performance). Further, this approach requires additional server-side instrumentation and dedicated resources to actively collect performance reports from clients.

There have been some earlier attempts to passively estimate the response time observed by clients from network level information. SPAND [21, 22] determines network characteristics by making shared, passive measurements from a collection of hosts and uses this information for server selection, i.e. for routing client requests to the server with the best observed response time in a geographically distributed web server cluster.

# 3 EtE Monitor Architecture

EtE monitor consists of four program modules shown in Figure 1:

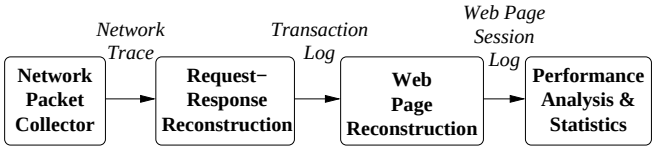



Figure 1: EtE Monitor Architecture.

1. The *Network Packet Collector* module collects the network packets using *tcpdump*[24] and records them to a *Network Trace*, enabling offline analysis.

2. In the *Request-Response Reconstruction* module, EtE monitor reconstructs all TCP connections from the *Network Trace* and extracts HTTP transactions (a request with the corresponding response) from the payload. EtE monitor does not consider encrypted connections whose content cannot be analyzed. After obtaining the HTTP transactions, the monitor stores some HTTP header lines and other related information in the *Transaction log* for future processing (excluding the HTTP payload). To rebuild HTTP transactions from TCP-level traces, we use a methodology proposed by Feldmann [7] and described in more detail and extended to work with persistent HTTP connections by Krishnamurthy and Rexford [14].

3. The *Web Page Reconstruction* module is responsible for grouping underlying physical object retrievals together into logical web pages and stores them in the *Web Page Session Log.*

4. Finally, the *Performance Analysis and Statistics* module summarizes a variety of performance characteristics integrated across all client accesses.

EtE monitor can be deployed in several different ways. First, it can be installed on a web server as a *software component* to monitor web transactions on a particular server. However, our software would then compete with the web server for CPU cycles and I/O bandwidth (as quantified in Section 7). Another solution is to place EtE monitor as an independent *network appliance* at a point on the network where it can capture all HTTP transactions for a web server. If a web site consists of multiple web servers, EtE monitor should be placed at the common entrance and exit of all web servers. If a web site is supported by geographically distributed web servers, such a common point may not exist. Nevertheless, distributed web servers typically use "sticky connections", i.e., once the client has established a connection with a web server, the subsequent client requests are sent to the same server. In this case, EtE monitor can still be used to capture a flow of transactions to a particular geographic site.

# 4 Request-Response Reconstruction Module

As described above, the Request-Response Reconstruction module reconstructs all observed TCP connections. The TCP connections are rebuilt from the *Network Trace* using client IP addresses, client port numbers, and request (response) TCP sequence numbers. Within the payload of the rebuilt TCP connections, HTTP transactions can be delimited as defined by the HTTP protocol. Meanwhile, the timestamps, sequence numbers and acknowledged sequence numbers for HTTP requests can be recorded for later matching with the corresponding HTTP responses.

When a client clicks a hypertext link to retrieve a particular web page, the browser first establishes a TCP connection with the web server by sending a SYN packet. If the server is ready to process the request, it accepts the connection by sending back a second SYN packet acknowledging the client's SYN [1]. At this point, the client is ready to send HTTP requests to retrieve the HTML file and all embedded objects. For each request, we are concerned with the timestamps for the first byte and the last byte of the request since they delimit the request transfer time and the beginning of server processing. We are similarly concerned with the timestamps of the beginning and the end of the corresponding HTTP response.

EtE monitor detects aborted connections by observing either a RST packet sent by an HTTP client to explicitly indicate an aborted connection or by a FIN/ACK

[1] Whenever EtE monitor detects a SYN packet, it considers the packet as a new connection iff it cannot find a SYN packet with the same source port number from the same IP address. A retransmitted SYN packet is not considered as a newly established connection. However, if a SYN packet is dropped, e.g. by intermediate routers, there is no way to detect the dropped SYN packet on server side.

packet sent by the client where the acknowledged sequence number is less than the observed maximum sequence number sent from the server. After reconstructing the HTTP transactions (a request and the corresponding response), the monitor records the HTTP header lines in the *Transaction Log* and discards the actual body of the HTTP response.

Each entry in the log includes a number of fields: (1) a unique flow ID for the TCP connection, (2) the client's IP address, (3) the requested URL, (4) the content type, (5) the *referer* field, (6) the *via* field, (7) whether the request was aborted, (8)the number of packets resent during the connection (potentially an indication of the presence of network congestion), (9) the size and timestamps of the request and response. Some fields in the entry are used to rebuild web pages, while other fields can be used to measure end-to-end performance.

An alternative way to collect most of the fields of the *Transaction Log* entry is to extend web server functionality. Apache, Netscape and IIS all have appropriate APIs. Most of the fields in the *Transaction Log* can be extracted via server instrumentation. This approach has some merits: 1) since a web server deals directly with request-response processing, the reconstruction of TCP connections becomes unnecessary; 2) it can handle encrypted connections.

However, the primary drawback of this approach is that web servers must be modified in an application specific manner. Our approach is independent of any particular server technology. On the other hand, instrumentation solutions cannot obtain network level information, such as the connection setup time and the resent packets, which can be observed by EtE monitor.

# 5 Page Reconstruction Module

To measure the client perceived end-to-end response time for retrieving a web page, one needs to identify the objects that are embedded in a particular web page and to measure the response time for the client requests retrieving these embedded objects from the web server. Although we can determine some embedded objects of a web page by parsing the HTML for the "container object", some embedded objects cannot be easily discovered through static parsing. For example, JavaScript is used in web pages to retrieve additional objects. Without executing the JavaScript, it may be difficult to discover the identity of such objects.

Automatically, determining the content of a page requires a technique to delimit individual page accesses. One recent study [6] uses an estimate of client think time as the delimiter between two pages. While this method is simple and useful, it may be inaccurate in some important cases. For example, consider the case where a client opens two web pages from one server at the same time. Here, the requests for the two different web pages interleave each other without any think time between them. Another case is when the interval between the requests for objects within one page may be too long to be distinguishable from think time (perhaps because of the network conditions).

Different from previous work, our methodology uses heuristics to determine the objects composing a web page, and applies statistics to adjust the results. EtE uses the HTTP *referer* field as a major "clue" to group objects into a web page. The *referer* field specifies the URL from which the requested URL was obtained. Thus, all requests for the embedded objects in an HTML file are recommended to set the *referer* fields to the URL of the HTML file. However, since the *referer* fields are set by client browsers, not all browsers set the fields. To solve this, EtE monitor first builds a *Knowledge Base* from those requests with *referer* fields, and uses more aggressive heuristics to group the requests without *referer* fields based on the *Knowledge Base* information.

Subsection 5.1 outlines *Knowledge Base* construction of web page objects. Subsection 5.2 presents the algorithm and technique to group the requests in web page accesses using *Knowledge Base* information and a set of additional heuristics. Subsection 5.3 introduces a statistical analysis to identify valid page access patterns and to filter out incorrectly constructed accesses.

## 5.1 Building a Knowledge Base of Web Page Objects

The goal of this step is to reconstruct a special subset of web page accesses, which we use to build a *Knowledge Base* about web pages and the objects composing them. Before grouping HTTP transactions into web pages, EtE monitor first sorts all transactions from the *Transaction Log* using the timestamps for the beginning of the requests in increasing time order. Thus, the requests for the embedded objects of a web page must follow the request for the corresponding HTML file of the page. When grouping objects into web pages (here and in the next subsection), we consider only transactions with *successful* responses, i.e. with status code 200 in the responses.

The next step is to scan the sorted transaction log and group objects into web page accesses. Not all the transactions are useful for the *Knowledge Base* construction process. During this step, some of the *Transaction Log* entries are excluded from our current consideration:

- Content types that are known not to contain embedded objects are excluded from the knowledge base, e.g., *application/postscript*, *application/x-tar*, *application/pdf*, *application/zip* and *text/plain*. For the rest of the paper, we call them *independent, single page* objects.

- If the *referer* field of a transaction is not set and its

content type is not *text/html*, EtE monitor excludes it from further consideration.

To group the rest of the transactions into web page accesses, we use the following fields from the entries in the *Transaction Log*: the request URL, the request *referer* field, the response *content type*, and the client IP address. EtE monitor stores the web page access information into a hash table, the *Client Access Table* depicted in Figure 2, which maps a client's IP address to a *Web Page Table* containing the web pages accessed by the client. Each entry in the *Web Page Table* is a web page access, and composed of the URLs of HTML files and the embedded objects. Notice that EtE monitor makes no distinction between statically and dynamically generated HTML files. We consider embedded HTML pages, e.g. framed web pages, as separate web pages.

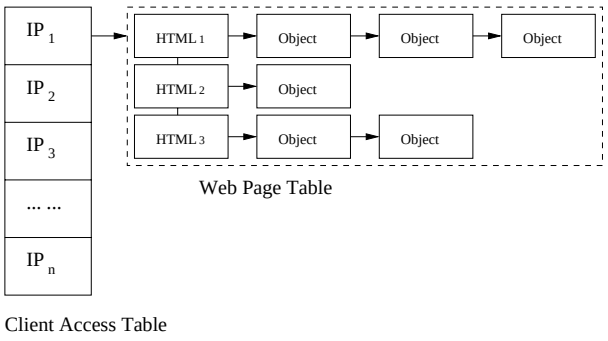



Figure 2: *Client Access Table*.

When processing an entry of the *Transaction Log*, EtE monitor first locates the *Web Page Table* for the client's IP in the *Client Access Table*. Then, EtE monitor handles the transaction according to its content type:

1. If the content type is *text/html*, EtE monitor treats it as the beginning of a web page and creates a new web page entry in the *Web Page Table*.

2. For other content types, EtE monitor attempts to insert the URL of the requested object into the web page that contains it according to its *referer* field. If the referred HTML file is already present in the *Web Page Table*, EtE monitor appends this object at the end of the entry. If the referred HTML file does not exist in the client's *Web Page Table*, it means that the client may have retrieved a cached copy of the object from somewhere else between the client and the web server. In this case, EtE monitor first creates a new web page entry in the *Web Page Table* for the referred HTML file. Then it appends the considered object to this page.

From the *Client Access Table*, EtE monitor determines the *content template* of any given web page as a combined set of all the objects that appear in all the access patterns for this web page. Thus, EtE monitor scans the *Client Access Table* and creates a new hash table, as shown in Figure 3, which is used as a *Knowledge Base* to group the accesses for the same web pages from other client's browsers that do not set the *referer* fields.

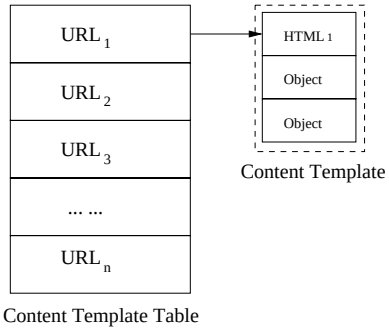



Figure 3: *Knowledge Base* of web pages.

## 5.2 Reconstruction of Web Page Accesses

With the help of the *Knowledge Base*, EtE monitor processes the entire *Transaction Log* again. This time, EtE monitor does not exclude the entries without *referer* fields. Using data structures similar to those introduced in Section 5.1, EtE monitor scans the sorted *Transaction Log* and creates a new *Client Access Table* to store all accesses as depicted in Figure 2. For each transaction, EtE monitor locates the *Web Page Table* for the client's IP in the *Client Access Table*. Then, EtE monitor handles the transaction depending on the content type:

1. If the content type is *text/html*, EtE monitor creates a new web page entry in the *Web Page Table*.

2. If a transaction is an independent, single page object, EtE monitor marks it as individual page without any embedded objects and allocates a new web page entry in the *Web Page Table*.

3. For other content types that can be embedded in a web page, EtE monitor attempts to insert it into the web page that contains it.

- If the *referer* field is set for this transaction, EtE monitor attempts to locate the referred page in the following way. If the referred HTML file is in an existing page entry in the *Web Page Table*, EtE monitor appends the object at the end of the entry. If the referred HTML file does not exist in the client's *Web Page Table*, EtE monitor first creates a new web page entry in the table for the referred page and marks it as *nonexistent*. Then it appends the object to this page. If the *referer* field is not set for this transaction, EtE monitor uses the following policies. With the help of the *Knowledge Base*, EtE monitor checks each page entry in the *Web Page Table* from the latest to earliest. If the *Knowledge Base* contains the *content template* for the checked

page and the considered object does not belong to it, EtE monitor skips the entry and checks the next one until a page containing the object is found. If such an entry is found, EtE monitor appends the object to the end of the web page.

- If none of the web page entries in the *Web Page Table* contains the object based on the *Knowledge Base*, EtE monitor searches in the client's *Web Page Table* for a web page accessed via the same flow ID as this object. If there is such a web page, EtE monitor appends the object to the page.

- Otherwise, if there are any accessed web pages in the table, EtE monitor appends the object to the latest accessed one.

If none of the above policies can be applied, EtE monitor drops the request. Obviously, the above heuristics may introduce some mistakes. Thus, EtE monitor also adopts a *configurable think time threshold* to delimit web pages. If the time gap between the object and the tail of the web page that it tries to append to is larger than the threshold, EtE monitor skips the considered object. In this paper, we adopt a configurable think time threshold of 4 sec.

### 5.3 Identifying Valid Accesses Using Statistical Analysis of Access Patterns

Although the above two-pass process can effectively provide accurate web page access reconstruction in most cases, there could still be some accesses grouped incorrectly. To filter out such accesses, we must better approximate the actual content of a web page.

All the accesses to a web page usually exhibit a set of different access patterns. For example, an access pattern can contain all the objects of a web page, while other patterns may contain a subset of them (e.g., because some objects were retrieved from a browser or network caches). We assume the same access patterns of those incorrectly grouped accesses should rarely appear repeatedly. Thus, we can use the following statistical analysis on access patterns to determine the actual content of web pages and exclude the incorrectly grouped accesses.

First, from the *Client Access Table* created in Subsection 5.2, EtE monitor collects all possible access patterns for a given web page and identifies the *probable content template* of the web page as the combined set of all objects that appear in all the accesses for this page. Table 1 shows an example of a *probable content template*. EtE monitor assigns an index for each object. The column *URL* lists the URLs of the objects that appear in the access patterns for the web page. The column *Frequency* shows the frequency of an object in the set of all web page accesses. In Table 1, the indices are sorted by the occurrence frequencies of the objects. The column *Ratio* is the percentage of the object's accesses in the total accesses for the page.

| Index | URL | Frequency | Ratio (%) |
|---|---|---|---|
| 1 | /index.html | 2937 | 95.51 |
| 2 | /img1.gif | 689 | 22.41 |
| 3 | /img2.gif | 641 | 20.85 |
| 4 | /log1.gif | 1 | 0.03 |
| 5 | /log2.gif | 1 | 0.03 |

Table 1: Web page *probable content template*. There are 3075 accesses for this page.

Sometimes, a web page may be pointed to by several URLs. For example, *http://www.hpl.hp.com* and *http://www.hpl.hp.com/index.html* both point to the same page. Before computing the statistics of the access patterns, EtE monitor attempts to merge the accesses for the same web page with different URL expressions. EtE monitor uses the *probable content* templates of these URLs to determine whether they indicate the same web page. If the *probable content* templates of two pages only differ due to the objects with small percentage of accesses (less than 1%, which means these objects might have been grouped by mistake), then EtE monitor ignores this difference and merges the URLs.

Based on the *probable content template* of a web page, EtE monitor uses the indices of objects in the table to describe the access patterns for the web page. Table 2 demonstrates a set of different access patterns for the web page in Table 1. Each row in the table is an access pattern. The column *Object Indices* shows the indices of the objects accessed in a pattern. The columns *Frequency* and *Ratio* are the number of accesses and the proportion of the pattern in the total number of all the accesses for the web page. For example, pattern 1 is a pattern in which only the object *index.html* is accessed. It is the most popular access pattern for this web page: 2271 accesses out of the total 3075 accesses represent this pattern. In pattern 2, the objects *index.html*, *img1.gif* and *img2.gif* are accessed.

| Pattern | Object Indices | Frequency | Ratio (%) |
|---|---|---|---|
| 1 | 1 | 2271 | 73.85 |
| 2 | 1,2,3 | 475 | 15.45 |
| 3 | 1,2 | 113 | 3.67 |
| 4 | 1,3 | 76 | 2.47 |
| 5 | 2,3 | 51 | 1.66 |
| 6 | 2 | 49 | 1.59 |
| 7 | 3 | 38 | 1.24 |
| 8 | 1,2,4 | 1 | 0.03 |
| 9 | 1,3,5 | 1 | 0.03 |

Table 2: Web page access patterns.

With the statistics of access patterns, EtE monitor further attempts to estimate the *true content template* of web pages, which excludes the mistakenly grouped access patterns. Intuitively, the proportion of these invalid

access patterns cannot be high. Thus, EtE monitor uses a configurable ratio threshold to exclude the invalid patterns (in this paper, we use 1% as a configurable ratio threshold). If the ratio of a pattern is below the threshold, EtE does not consider it as a valid pattern. In the above example, patterns 8 and 9 are not considered as valid access patterns. Only the objects found in the valid access patterns are considered as the embedded objects in a given web page. Objects 1, 2, and 3 define the *true content template* of the web page shown in Table 3. Based on the *true content templates*, EtE monitor filters out all the invalid accesses in a *Client Access Table*, and records the correctly constructed page accesses in the *Web Page Session Log*, which can be used to evaluate the end-to-end response performance.

| Index | URL |
|---|---|
| 1 | /index.html |
| 2 | /img1.gif |
| 3 | /img2.gif |

Table 3: Web page *true content template*.

# 6 Metrics to Measure Web Service Performance

In this section, we introduce a set of metrics and the ways to compute them in order to measure a web service efficiency. These metrics can be categorized as:

- metrics approximating the end-to-end response time observed by the client for a web page download. Additionally, we provide a means to calculate the breakdown between server processing and networking portions of overall response time.
- metrics evaluating the caching efficiency for a given web page by computing the server file hit ratio and server byte hit ratio for the web page.
- metrics relating the end-to-end performance of aborted web pages to the QoS.

## 6.1 Response Time Metrics

We use the following functions to denote the critical timestamps for connection *conn* and request $r$:

- $t_{syn}(conn)$: time when the first SYN packet from the client is received for establishing the connection *conn*;
- $t_{req}^{start}(r)$: time when the first byte of the request $r$ is received ;
- $t_{req}^{end}(r)$: time when the last byte of the request $r$ is received;
- $t_{resp}^{start}(r)$: time when the first byte of the response for $r$ is sent;
- $t_{resp}^{end}(r)$: time when the last byte of the response for $r$ is sent;
- $t_{resp}^{ack}(r)$: time when the ACK for the last byte of the response for $r$ is received.

Additionally, for a web page $P$, we have the following variables:

- $N$ - the number of distinct connections used to retrieve the objects in the web page $P$;
- $r_1^k, ...r_{n_k}^k$ - the requests for the objects retrieved through the connection $conn_k$ $(k = 1, ..., N)$, and ordered accordingly to the time when these requests were received, i.e.,

$$t_{req}^{end}(r_1^k) \leq t_{req}^{end}(r_2^k) \leq ... \leq t_{req}^{end}(r_{n_k}^k).$$

The extended version of HTTP 1.0 and later version HTTP 1.1 [9] introduce the concepts of *persistent connections* and *pipelining*. Persistent connections enable reuse of a single TCP connection for multiple object retrievals from the same IP address. Pipelining allows a client to make a series of requests on a persistent connection without waiting for the previous response to complete (the server must, however, return the responses in the same order as the requests are sent).

We consider the requests $r_i^k, ..., r_n^k$ to belong to the same *pipelining group* (denoted as $PipeGr = \{r_i^k, ..., r_n^k\}$) if for any j such that $i \leq j-1 < j \leq n$, $t_{req}^{start}(r_j^k) \leq t_{resp}^{end}(r_{j-1}^k)$.

Thus for all the requests on the same connection $conn_k$: $r_1^k, ..., r_{n_k}^k$, we define the maximum pipelining groups in such a way that they do not intersect, e.g.,

$$\underbrace{r_1^k, ..., r_i^k}_{PipeGr_1}, \underbrace{r_{i+1}^k}_{PipeGr_2}, ..., \underbrace{r_{n_k}^k}_{PipeGr_l}.$$

For each of the pipelining groups, we define three portions of response time: total response time ($Total$), network-related portion ($Network$), and lower-bound estimate of the server processing time ($Server$).

Let us consider the following example. For convenience, let us denote $PipeGr_1 = \{r_1^k, ..., r_i^k\}$.

Then

$$Total(PipeGr_1) = t_{resp}^{end}(r_i^k) - t_{req}^{start}(r_1^k),$$

$$Network(PipeGr_1) = \sum_{j=1}^{i} (t_{resp}^{end}(r_j^k) - t_{resp}^{start}(r_j^k)),$$

$$Server(PipeGr_1) = Total(PipeGr_1) - Network(PipeGr_1).$$

If no pipelining exists, a pipelining group only consists of one request. In this case, the computed server time represents precisely the server processing time for a given

request-response pair. If a connection adopts pipelining, the "real" server processing time might be larger than the computed server time because it can partially overlap the network transfer time, and it is difficult to estimate the exact server processing time from the packet-level information. However, we are still interested to estimate the "non-overlapping" server processing time as this is the portion of the server time on a critical path of overall end-to-end response time. Thus, we use as an estimate the lower-bound server processing time, which is explicitly exposed in the overall end-to-end response.

If connection $conn_k$ is a newly established connection to retrieve a web page, we observe additional connection setup time:

$$Setup(conn_k) = t_{req}^{start}(r_1^k) - t_{syn}(conn_k)\ ^2,$$

otherwise the setup time is 0. Additionally, we define $t^{start}(conn_k) = t_{syn}(conn_k)$ for a newly established connection, otherwise, $t^{start}(conn_k) = t_{req}^{start}(r_1^k)$.

Similarly, we define the breakdown for a given connection $conn_k$:

$$Total(conn_k) = Setup(conn_k) + t_{resp}^{end}(r_{n_k}^k) - t_{req}^{start}(r_1^k),$$

$$Network(conn_k) = Setup(conn_k) + \sum_{j=1}^{l} Network(PipeGr_j),$$

$$Server(conn_k) = \sum_{j=1}^{l} Server(PipeGr_j).$$

Now, we define similar latencies for a given page $P$:

$$Total(P) = \max_{j \le N} t_{resp}^{end}(r_{n_j}^j) - \min_{j \le N} t^{start}(conn_j),$$

$$CumNetwork(P) = \sum_{j=1}^{N} Network(conn_j),$$

$$CumServer(P) = \sum_{j=1}^{N} Server(conn_j).$$

For the rest of the paper, we will use the term *EtE time* interchangeably with $Total(P)$ time.

All the above formulae use $t_{resp}^{end}(r)$ to calculate response time. An alternative way is to use as the end of a transaction the time $t_{resp}^{ack}(r)$ when the ACK for the last byte of the response is received by a server. Figure 4 shows an example of a simplified scenario where a 1-object page is downloaded by the client: it shows the communication protocol for connection setup between the client and the server as well as the set of major timestamps collected by the EtE monitor on the server side. The connection setup time measured on the server side is the time between the client SYN packet and the first byte of the client request. This represents a close approximation for the original client setup time (we present more detail on this point in Section 8 when reporting our validation experiments).

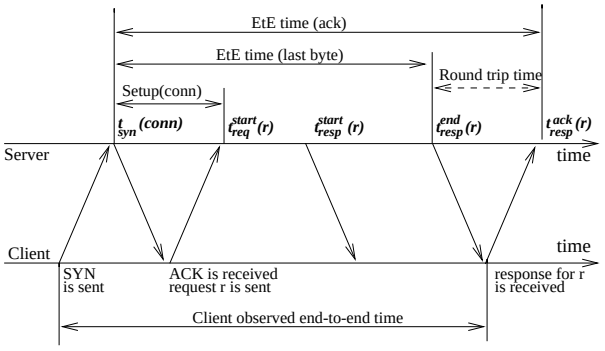



Figure 4: An example of a 1-object page download by the client: major timestamps collected by the EtE monitor on the server side.

If the ACK for the last byte of the client response is not delayed or lost, $t_{resp}^{ack}(r)$ is a more accurate approximation of the end-to-end response time observed by the client: it "compensates" for the latency of the first client SYN packet that is not measured on the server side. The difference between the two methods, i.e. *EtE time (last byte)* and *EtE time (ack)*, is only a round trip time, which is on the scale of milliseconds. Since the overall response time is on the scale of seconds, we consider this deviation an acceptably close approximation. However, to avoid the problems with delayed or lost ACKs, EtE monitor determines the end of a transaction as the time when the last byte of a response is sent by a server.

Metrics introduced in this section account for packet retransmission. However, if the retransmission happens on connection establishment (i.e. due to dropped SYNs), EtE monitor cannot account for this.

The functions $CumNetwork(P)$ and $CumServer(P)$ give the sum of all the network-related and server processing portions of the response time over all connections used to retrieve the web page. However, the connections can be opened concurrently by the browser. To evaluate the concurrency impact, we introduce the page concurrency coefficient *ConcurrencyCoef(P)*:

$$ConcurrencyCoef(P) = \frac{\sum_{j=1}^{N} Total(conn_j)}{Total(P)}.$$

Using page concurrency coefficient, we finally compute the network-related and the service-related portions of response time for a particular page $P$:

$$Network(P) = CumNetwork(P)/ConcurrencyCoef(P),$$

$$Server(P) = CumServer(P)/ConcurrencyCoef(P).$$

EtE monitor can distinguish the requests sent to a web server from clients behind proxies by checking the

[2] The connection setup time as measured by EtE monitor does not include dropped SYNs, as discussed earlier in Section 4.

HTTP *via* fields. If a client page access is handled via the same proxy (which is typically the case, especially when persistent connections are used), EtE monitor provides correct measurements for end-to-end response time and other metrics, as well as provides interesting statistics on the percentage of client requests coming from proxies. Clearly, this percentage is an approximation, since not all the proxies set the *via* fields in their requests. Finally, EtE monitor can only measure the response time to a proxy instead of the actual client behind it.

### 6.2 Metrics Evaluating the Web Service Caching Efficiency

Real clients of a web service may benefit from the presence of network and browser caches, which can significantly reduce their perceived response time. However, none of the existing performance measurement techniques provide any information on the impact of caches on web services: what percentage of the files and bytes are delivered from the server comparing with the total files and bytes required for delivering the web service. This impact can only be partially evaluated from web server logs by checking response status code 304, whose corresponding requests are sent by the network caches to validate whether the cached object has been modified. If the status code 304 is set, the cached object is not expired and need not be retrieved again.

To evaluate the caching efficiency of a web service, we introduce two metrics: *server file hit ratio* and *server byte hit ratio* for each web page.

For a web page $P$, assume the objects composing the page are $O_1, ..., O_n$. Let $Size(O_i)$ denote the size of object $O_i$ in bytes. Then we define $NumFiles(P) = n$ and $Size(P) = \sum_{j=1}^{n} Size(O_j)$.

Additionally, for each access $P^i_{access}$ of the page $P$, assume the objects retrieved in the access are $O^i_1, ..., O^i_{k_i}$, we define $NumFiles(P^i_{access}) = k_i$ and $Size(P^i_{access}) = \sum_{j=1}^{k_i} Size(O^i_j)$. First, we define *file hit ratio* and *byte hit ratio* for each page access in the following way:

$$FileHitRatio(P^i_{access}) = NumFiles(P^i_{access})/NumFiles(P),$$

$$ByteHitRatio(P^i_{access}) = Size(P^i_{access})/Size(P).$$

Let $P^1_{access}, ..., P^N_{access}$ be all the accesses to the page $P$ during the observed time interval. Then

$$ServerFileHitRatio(P) = \frac{1}{N} \sum_{k \le N} FileHitRatio(P^k_{access}),$$

$$ServerByteHitRatio(P) = \frac{1}{N} \sum_{k \le N} ByteHitRatio(P^k_{access}).$$

The lower numbers for *server file hit ratio* and *server byte hit ratio* indicate the higher caching efficiency for the web service, i.e., more files and bytes are served from network and client browser caches.

### 6.3 Aborted Pages and QoS

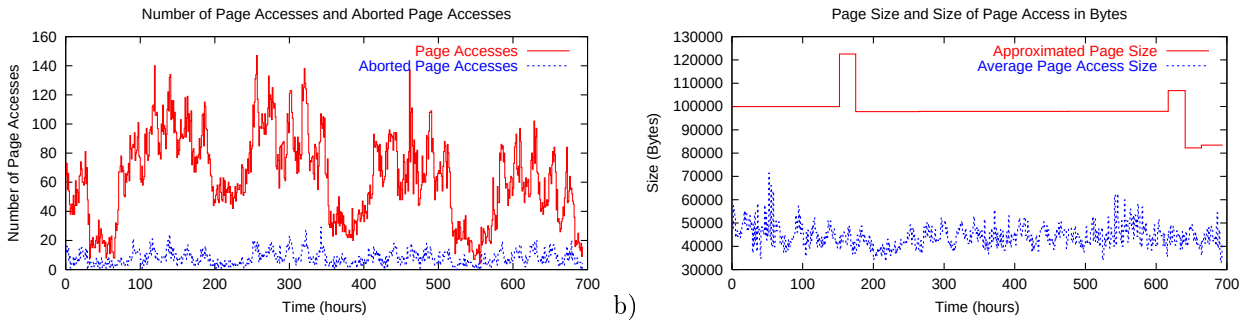
User-perceived QoS is another important metric to consider in EtE monitor. One way to measure the QoS of a web service is to measure the frequency of aborted connections. However, such simplistic interpretation of aborted connections and web server QoS has several drawbacks. First, a client can interrupt HTTP transactions by clicking the browser's "stop" or "reload" button while a web page is downloading, or clicking a displayed link before the page is completely downloaded. Thus, only a subset of aborted connections are relevant to poor web site QoS or poor networking conditions, while other aborted connections are caused by client-specific browsing patterns. On the other hand, a web page can be retrieved through multiple connections. A client's browser-level interruption can cause all the currently open connections to be aborted. Thus, the number of aborted page accesses more accurately reflects client satisfaction than the number of aborted connections.

For aborted pages, we distinguish the subset of pages $\Pi_{bad}$ with the response time higher than the given threshold $X_{EtE}$ (in our case, $X_{EtE} = 6\ sec$). Only these pages might be reflective of the bad quality downloads. While a simple deterministic cut off point cannot truly capture a particular client's expectation for site performance, the current industrial *ad hoc* quality goal is to deliver pages within 6 sec [12]. We thus attribute aborted pages that have not crossed the 6 sec threshold to individual client browsing patterns. The next step is to distinguish the reasons leading to poor response time: whether it is due to network or server-related performance problems, or both.

## 7 Case Studies

In this section, we present two simple case studies to illustrate the benefits of EtE monitor in assessing web site performance. The first site is the HP Labs external site (*HPL Site*), *http://www.hpl.hp.com*. Static web pages comprise most of this site's content. We measured performance of this site for a month, from July 12, 2001 to August 11, 2001. The second site is a support site for a popular HP product family, which we call *Support Site*. It uses JavaServer Pages [11] technology for dynamic page generation. The architecture of this site is based on a geographically distributed web server cluster with Cisco Distributed Director [5] for load balancing, using "sticky connections". We measure the site performance for 2 weeks, from October 11, 2001 to October 25, 2001.

Table 4 summarizes the two site's performance *at-a-glance* during the measured period using the two most frequently accessed pages at each site. The average end-to-end response time of client accesses to these pages reflects good overall performance. However in the case of




a) b)

Figure 5: HPL site during a month: a) Number of all and aborted accesses to *index.html* ; b) Approximated page size and average access size to *index.html*.

| **Metrics** | HPL *url1* | HPL *url2* | Support *url1* | Support *url2* |
|---|---|---|---|---|
| EtE time | 3.5 sec | 3.9 sec | 2.6 sec | 3.3 sec |
| % of accesses above 6 sec | 8.2% | 8.3% | 1.8% | 2.2% |
| % of aborted accesses above 6 sec | 1.3% | 2.8% | 0.1% | 0.2% |
| % of accesses from clients-proxies | 16.8% | 19.8% | 11.2% | 11.7% |
| EtE time from clients-proxies | 4.2 sec | 3 sec | 4.5 sec | 3 sec |
| Network-vs-Server ratio in EtE time | 99.6% | 99.7% | 96.3% | 93.5% |
| Page size | 99 KB | 60.9 KB | 127 KB | 100 KB |
| Server file hit ratio | 38.5% | 58% | 22.9% | 28.6% |
| Server byte hit ratio | 44.5% | 63.2% | 52.8% | 44.6% |
| Number of objects | 4 | 2 | 32 | 32 |
| Number of connections | 1.6 | 1 | 6.5 | 9.1 |

Table 4: *At-a-Glance* statistics for *www.hpl.hp.com* and *support* site during the measured period.

HPL, a sizeable percentage of accesses take more than 6 sec to complete (8.2%-8.3%), with a portion leading to aborted accesses (1.3%-2.8%). The Support site had better overall response time with a much smaller percentage of accesses above 6 sec (1.8%-2.2%), and a correspondingly smaller percentage of accesses aborted due to high response time (0.1%-0.2%). Overall, the pages from both sites are comparable in size. However, the two pages from the HPL site have a small number of objects per page (4 and 2 correspondingly), while the Support site pages are composed of 32 different objects. Page composition influences the number of client connections required to retrieve the page content. Additionally, statistics show that network and browser caches help to deliver a significant amount of page objects: in the case of the Support site, only 22.9%-28.6% of the 32 objects are retrieved from the server, accounting for 44.6%-52.8% of the bytes in the requested pages. As discussed earlier, the Support site content is generated using dynamic pages, which could potentially lead to a higher ratio of server processing time in the overall response time. But in general, the network transfer time dominates the performance for both sites, ranging from 93.5% for the Support site to 99.7% for the HPL site.

Given the above summary, we now present more detailed information from our site measurements. For the HPL site, the two most popular pages during the observed period were *index.html* and a page in the news section describing the Itanium chip (we call it *itanium.html*).

Figure 5 a) shows the number of page accesses to *index.html*, as well as the number of aborted page accesses during the measured period. The graph clearly reflects weekly access patterns to the site.

Figure 5 b) reflects the *approximate page size*, as reconstructed by EtE monitor. We use this data to additionally validate the page reconstruction process. While debugging the tool, we manually compare the content of the 20 most frequently accessed pages reconstructed by EtE monitor against the actual web pages: the EtE monitor page reconstruction accuracy for popular pages is very high, practically 100%. Figure 5 b) allows us to "see" the results of this reconstruction process over the period of the study. In the beginning, it is a straight line exactly coinciding with the actual page size. At hour mark 153, it jumps and returns to a next straight line interval at the 175 hour mark. As we verified, the page has been partially modified during this time interval. The EtE monitor "picked" both the old and the modified page images, since they both occurred during the same day interval and represented a significant fraction of accesses. However, the next day, the *Knowledge Base* was "renewed" and had only the modified page information. The second "jump" of this line corresponds to the next modification of the page. The gap can be tightened, depending on the time interval EtE monitor is set to process. The other line in Figure 5 b) shows the average page access size, reflecting the server byte hit ratio of approximately 44%.

To characterize the reasons leading to the aborted web pages, we present analysis of the aborted accesses to *index.html* page for 3 days in August (since the monthly graph looks very "busy" on an hourly scale). Figure 6 a)

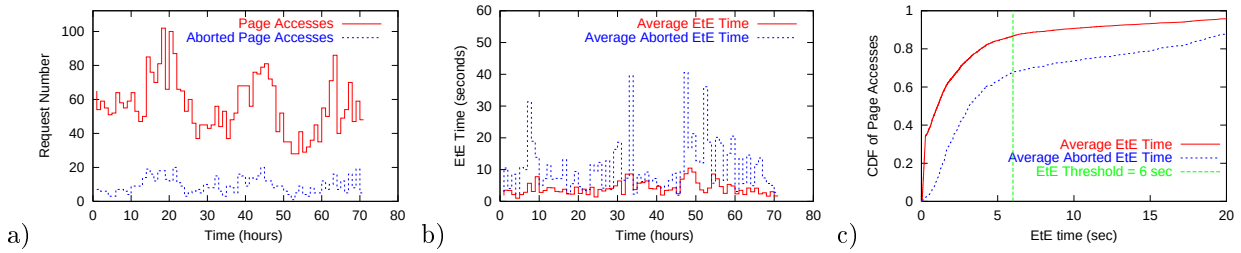



Figure 6: HPL site during 3 days: a) Number of all and aborted accesses to *index.html*; b) End-to-end response times for accesses to *index.html*; c) CDF of all and aborted accesses to *index.html* sorted by the response time in increasing order.

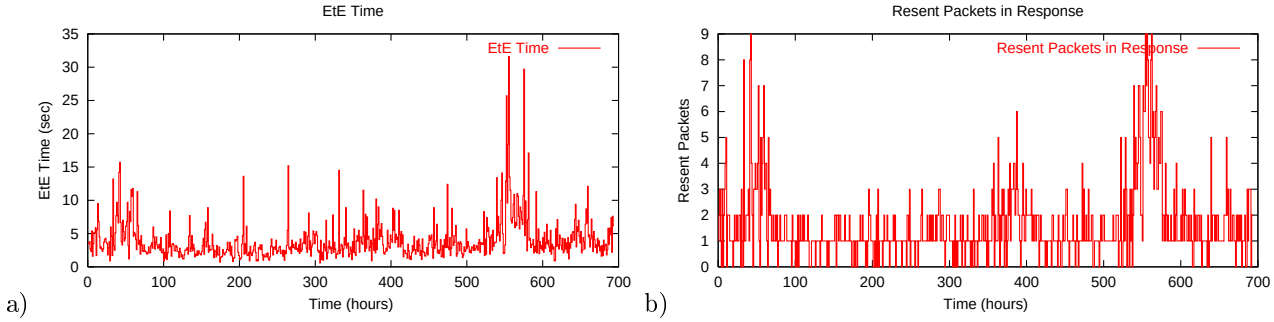



Figure 7: HPL site during a month: a) end-to-end response times for accesses to *index.html*; b) number of resent packets in response.

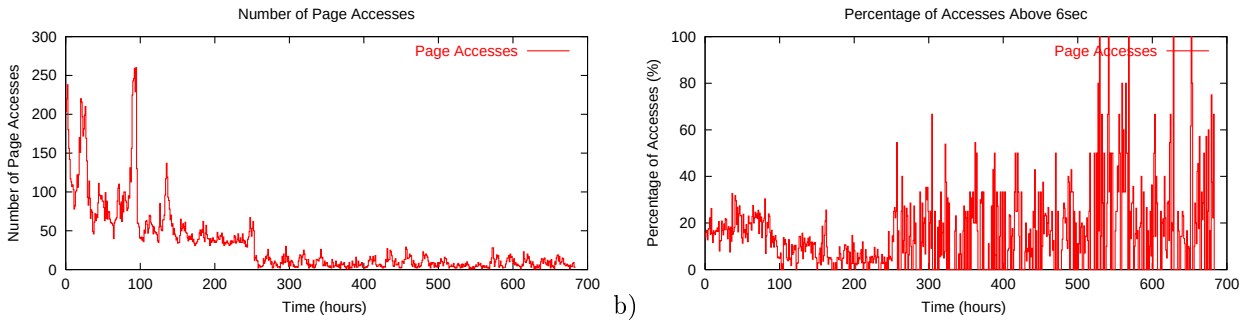
shows the number of all the requests and the aborted requests to *index.html* page during this interval. The number of aborted accesses (662) accounts for 16.4% of the total number of requests (4028).

Figure 6 b) shows the average end-to-end response time measured by EtE monitor for *index.html* and the average end-to-end response time for the aborted accesses to *index.html* on an hourly scale. The end-to-end response time for *index.html* page, averaged across all the page accesses, is 3.978 sec, while the average end-to-end response time of the aborted page accesses is 9.21 sec.

Figure 6 c) shows a cumulative distribution of all accesses and aborted accesses to *index.html* sorted by the end-to-end response time in increasing order. The vertical line on the graph shows the threshold of 6 sec that corresponds to an acceptable end-to-end response time. Figure 6 c) shows that 68% of the aborted accesses demonstrate end-to-end response times below 6 sec. This means that only 32% of all the aborted accesses, which in turn account for 5% of all accesses to the page, observe high end-to-end response time. The next step is to distinguish the reasons leading to a poor response time: whether it is due to network or server performance problems, or both. For all the aborted pages with high response time, the network portion of the response time dominates the overall response time (98%-99% of the total). Thus, we can conclude that any performance problems are likely not server-related but rather due to congestion in the network (though it is unclear whether the congestion is at the edge or the core of the network).

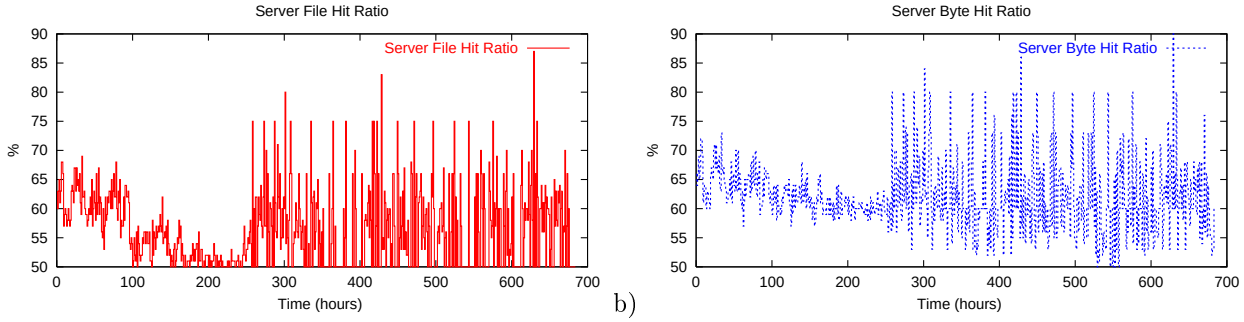
Figure 7 a) shows the end-to-end response time for accesses to *index.html* on an hourly scale during a month. In spite of good average response time reported in at-a-glance table, hourly averages reflect significant variation in response times. This graph helps to stress the advantages of EtE monitor and reflects the shortcomings of active probing techniques that measure page performance only a few times per hour: the collected test numbers could vary significantly from a site's instantaneous performance characteristics.

Figure 7 b) shows the number of resent packets in the response stream to clients. There are three pronounced "humps" with an increased number of resent packets. Typically, resent packets reflect network congestion or the existence of some network-related bottlenecks. Interestingly enough, such periods correspond to weekends when the overall traffic is one order of magnitude lower than weekdays (as reflected in Figure 5 a)). The explanation for this phenomenon is that during weekends the client population of the site "changes" significantly: most of the clients access the site from home using modems or other low-bandwidth connections. This leads to a higher observed end-to-end response time and an increase in the number of resent packets (i.e., TCP is likely to cause drops more often when probing for the appropriate congestion window over a low-bandwidth link). These results again stress the unique capabilities of EtE monitor to extract appropriate information from network packets, and reflect another shortcoming of active probing techniques that use a fixed number of artificial clients with rather good network connections to the Internet. For site designers, it is important to




a) b)

Figure 8: HPL site during a month: a) number of all accesses to *itanium.html*; b) percentage of accesses with end-to-end response time above 6 sec.




a) b)

Figure 9: HPL site: a) server file hit ratio for *itanium.html*; b) server byte hit ratio for *itanium.html*.

understand the actual client population and their end-to-end response time and the "quality" of the response. For instance, when large population of clients have limited bandwidth parameters, the site designers should consider making the pages and their objects "lighter weight".

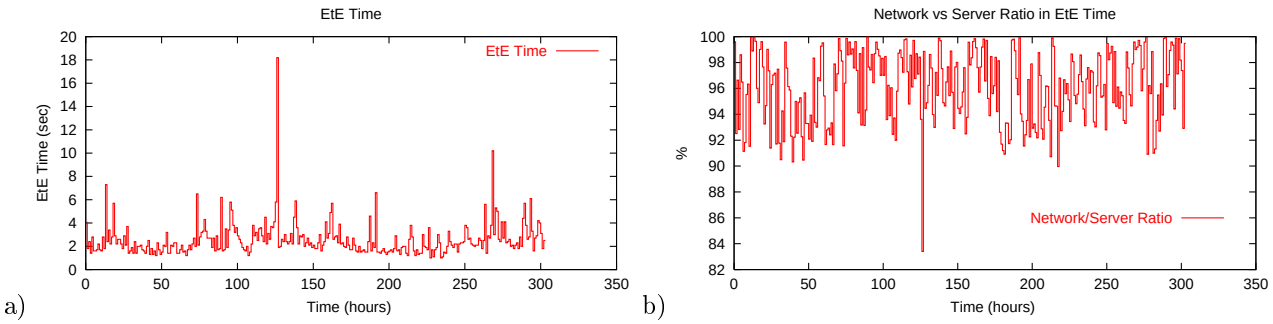
Figure 8 a) shows the number of page accesses to *itanium.html*. When we started our measurement of the HPL site, the *itanium.html* page was the most popular page, "beating" the popularity of the main *index.html* page. However, ten days later, this news article started to get "colder", and the page got to the seventh place by popularity.

Figure 8 b) shows the percentage of accesses with end-to-end response time above 6 sec. The percentage of high response time jumps significantly when the page becomes "colder". The reason behind this phenomenon is shown in Figure 9, which plots the server file hit and byte hit ratio. When the page became less popular, the number of objects and the corresponding bytes retrieved from the server increased significantly. This reflects that fewer network caches store the objects as the page becomes less popular, forcing clients to retrieve them from the origin server.

Figure 8 b) and Figure 9 explicitly demonstrate the network caching impact on end-to-end response time. When the caching efficiency of a page is higher (i.e., more page objects are cached by network and browser caches), the response time measured by EtE monitor is lower. Again, active probing techniques cannot measure (or account for) the page caching efficiency to reflect the "true" end-to-end response time observed by the actual clients.

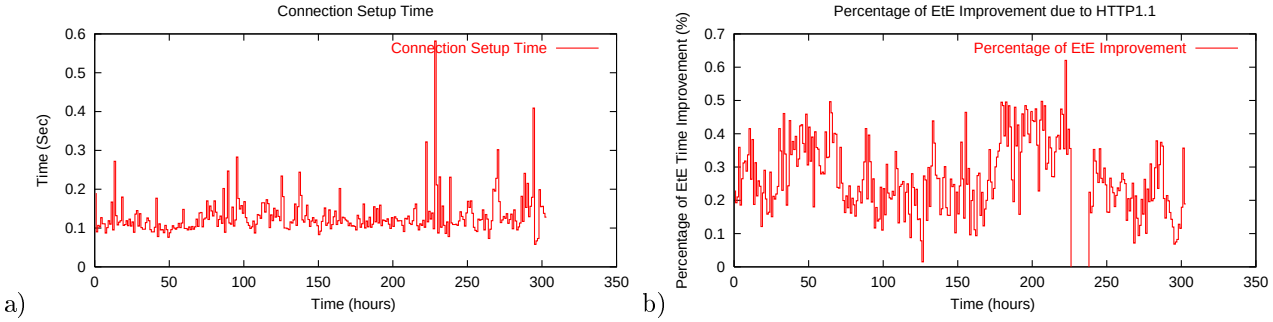
We now switch to the analysis of the Support site. We will only highlight some new observations specific to this site. Figure 10 a) shows the average end-to-end response time as measured by EtE monitor when downloading the site main page. This site uses JavaServer Pages technology for dynamic generation of the content. Since dynamic pages are typically more "compute intensive," it has a corresponding reflection in higher server-side processing fraction in overall response time. Figure 10 b) shows the network-server time ratio in the overall response time. It is higher compared to the network-server ratio for static pages from the HPL site. One interesting detail is that the response time spike around the 127 hour mark has a corresponding spike in increased server processing time, indicating some server-side problems at this point. The combination of data provided by EtE monitor can help service providers to better understand site-related performance problems.

The Support site pages are composed of a large number of embedded images. Two most popular site pages, which account for almost 50% of all the page accesses, consist of 32 objects. The caching efficiency for the site is very high: only 8-9 objects are typically retrieved from the server, while the other objects are served from network and browser caches. The site server is running



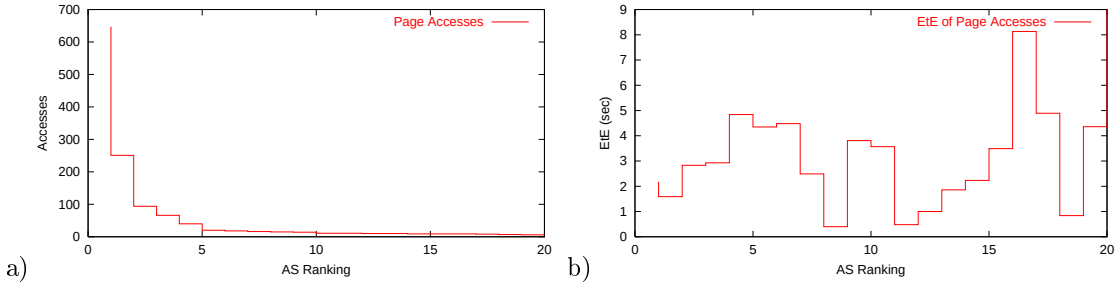

a) b)

Figure 10: Support site during 2 weeks: a) end-to-end response time for accesses to a main page; b) network-server time ratio for the main page.




a) b)

Figure 11: Support site during 2 weeks: a) connection setup time for the main page; b) an estimated percentage of end-to-end response time improvement if the server runs HTTP1.1.




a) b)

Figure 12: Support site: daily analysis of 20 ASes with largest client clusters: a) number of different clients accessing the main page; b) corresponding end-to-end response time per AS.

HTTP 1.0 server. Thus typical clients used 7-9 connections to retrieve 8-9 objects. The *ConcurrencyCoef* (see Section 6), which reflects the overlap portion of the latency between different connections for this page, was very low, around 1.038 (in fact, this is true for the site pages in general). This indicates that the efficiency of most of these connections is almost equal to sequential retrievals through a single persistent connection.

Figure 11 a) shows the connection setup time measured by EtE monitor. We perform a simple computation: how much of the end-to-end response time observed by current clients can be improved if the site server would run an HTTP 1.1 server, allowing clients to use just two persistent connections to retrieve the corresponding objects from the site? In other words, how much of the response time can be improved by eliminating unnecessary connection setup time?

Figure 11 b) shows the estimated percentage of end-to-end response time improvement available from running an HTTP 1.1 server. On average, during the observed interval, the response time improvement for *url1* is around 20% (2.6 sec is decreased to 2.1 sec), and for *url2* is around 32% (3.3 sec is decreased to 2.2 sec).

Figure 11 b) reveals an unexpected "gap" between 230-240 hour marks, when there was "no improvement" due to HTTP 1.1. More careful analysis shows that during this period, all the accesses retrieved only a basic HTML page using 1 connection, without consequent image retrievals. The other pages during the same interval have a similar pattern. It looks like the image directory was not accessible on the server. Thus, EtE monitor, by exposing the abnormal access patterns, can help service providers get additional insight into service related problems.

EtE monitor also provides the information about the client clustering by associating them with various ASes (Autonomous Systems). Figure 12 a) shows the 20 largest client clusters by ASes. Figure 12 b) reflects the corresponding average end-to-end response time per AS. The information provides a useful quantitative view on response times to the major client clusters. It can be used for efficient site design when the geographically distributed web cluster is needed to improve site performance. Similarly, such information can be used to make appropriate decisions on specific content distribution networks and wide-area replication strategies given a particular service's client population.

The ability of EtE monitor to reflect a site performance for different ASes (and groups of IP addresses) happens to be a very attractive feature for service providers. When service providers have special SLA-contracts with certain groups of customers, EtE monitor provides a unique ability to measure the response time observed by those clients and validate QoS for those contracts.

Finally, we present a few performance numbers to reflect the execution time of EtE monitor when processing data for the HPL and Support sites. The tests are run on a 550Mhz HP C3600 workstation with 512 MB of RAM. Table 5 presents the amount of data and the execution time for processing $10,000,000$ TCP Packets.

| **Duration, Size, and Execution Time** | HPL site | Support site |
|---|---|---|
| Duration of data collection | 3 days | 1 day |
| Collected data size | 7.2 GB | 8.94 GB |
| *Transaction Log* size | 35 MB | 9.6 MB |
| Entries in *Transaction Log* | 616,663 | 157,200 |
| Reconstructed page accesses | 90,569 | 8,642 |
| Reconstructed pages | 5,821 | 845 |
| EtE Execution Time | 12 min 44 sec | 17 min 41 sec |

Table 5: EtE monitor performance measurements.

The performance of reconstruction module performance depends on the complexity of the web page composition. For example, the Support site has a much higher percentage of embedded objects per page than the HPLabs pages. This "higher complexity" of the reconstruction process is reflected by the higher EtE monitor processing time for the Support site (17 min 41 sec) compared to the processing time for the HPLabs site (12 min 44 sec). The amount of incoming and outgoing packets of a web server farm that an EtE monitor can handle also depends on the rate at which *tcpdump* can capture packets and the traffic of the web site.

# 8 Validation Experiments

We performed two groups of experiments to validate the accuracy of EtE monitor performance measurements and its page access reconstruction power.

In the first experiment, we used two remote clients residing at Duke University and Michigan State University to issue a sequence of 40 requests to retrieve a designated web page from HPLabs external web site, which consists of an HTML file and 7 embedded images. The total page size is 175 Kbytes. To issue these requests, we use *httperf*[16], a tool which measures the connection setup time and the end-to-end time observed by the client for a full page download. At the same time, an EtE monitor measures the performance of HPLabs external web site. From EtE monitor measurements, we filter the statistics about the designated client accesses. Additionally, in EtE monitor, we compute the end-to-end time using two slightly different approaches from those discussed in Section 6.1:

- EtE time (*last byte*): where the *end* of a transaction is the time when the last byte of the response is sent by a server;
- EtE time (*ACK*): where the *end* of a transaction is the time when the ACK for the last byte of the response is received.

Table 6 summarizes the results of this experiment (the measurements are given in *sec*):

| **Client** | *httperf* | | *EtE monitor* | | |
|---|---|---|---|---|---|
| | Conn Setup | Resp. time | Conn Setup | EtE time (*last byte*) | EtE time (*ACK*) |
| Michigan | 0.074 | 1.027 | 0.088 | 0.953 | 1.026 |
| Duke | 0.102 | 1.38 | 0.117 | 1.28 | 1.38 |

Table 6: Experimental results validating the accuracy of EtE monitor performance measurements.

The connection setup time reported by EtE monitor is slightly higher (14-15 ms) than the actual setup time measured by *httperf*, since it includes the time to not only establish a TCP connection but also receive the first byte of a request. The EtE time (*ACK*) coincides with the actual measured response time observed by the client. The EtE time (*last byte*) is slightly lower than the actual response time by exactly a round trip delay (the connection setup time measured by *httperf* represents the round trip time for each client, accounting for 74-102 ms). These measurements correctly reflect our expectations for EtE monitor accuracy (see Section 6.1). Thus, we have some confidence that EtE monitor accurately approximates the actual response time observed by the client.

The second experiment was performed to evaluate the reconstruction power of EtE monitor. The EtE monitor with its two-pass heuristic method actively uses the *referer* field to reconstruct the page composition and to build a *Knowledge Base* about the web pages and objects composing them. This information is used during the second pass to more accurately group the requests into page accesses. The question to answer is: how dependent are the reconstruction results on the existence

of *referer* field information. If the *referer* field is not set in most of the requests, how is the EtE monitor reconstruction process affected? How is the reconstruction process affected by accesses generated by proxies?

To answer these questions, we performed the following experiment. To reduce the incorrectness introduced by proxies, we first filtered the requests with *via* fields, which are issued by proxies, from the original *Transaction Log*s for the both sites. These requests constitute 24% of total requests for the HPL site and 1.1% of total requests for the Support site. We call these logs *filtered logs*. Further, we mask the *referer* fields of all transactions in the *filtered logs* to study the correctness of reconstruction. We call these modified logs *masked logs*, which do not contain any *referer* fields. We notice that the requests with *referer* fields constitute 56% of the total requests for the HPL site and 69% for the Support site in the *filtered logs*. Then, EtE monitor processes the *filtered logs* and *masked logs*. Table 7 summarizes the results of this experiment.

| **Metrics** | HPL *url1* | HPL *url2* | Support *url1* | Support *url2* |
|---|---|---|---|---|
| Reconstructed page accesses (*filtered logs*) | 36,402 | 17,562 | 17,601 | 11,310 |
| EtE time (*filtered logs*) | 3.3 sec | 4.1 sec | 2.4 sec | 3.3 sec |
| Reconstructed page accesses (*masked logs*) | 33,735 | 14,727 | 15,401 | 8,890 |
| EtE time (*masked logs*) | 3.2 sec | 4.1 sec | 2.3 sec | 3.6 sec |

Table 7: Experimental results validating the accuracy of EtE monitor reconstruction process for HPL and Support sites.

The results of *masked logs* in Table 7 show that EtE monitor does a good job of page access reconstruction even when the requests do not have any *referer* fields. However, with the knowledge introduced by the *referer fields* in the *filtered logs*, the number of reconstructed page accesses increases by 9-21% for the considered URLs in Table 7. Additionally, we also find that the number of reconstructed accesses increases by 11.2-19.8% for all the considered URLs if EtE monitor processes the original logs without filtering either the *via* fields or the *referer* fields. The difference of EtE time between the two kinds of logs in Table 7 can be explained by the difference of the number of reconstructed accesses. Intuitively, more reconstructed page accesses lead to higher accuracy of estimation. This observation also challenges the accuracy of active probing techniques considering their relatively small sampling sets.

# 9 Limitations

There are a number of limitations to our EtE monitor architecture. Since EtE monitor extracts HTTP transactions by reconstructing TCP connections from captured network packets, it is unable to obtain HTTP information from encrypted connections. Thus, EtE monitor is not appropriate for sites that encrypt much of their data (e.g., via SSL).

In principle, EtE monitor must capture all traffic entering and exiting a particular site. Thus, our software must typically run on a single web server or a web server cluster with a single entry/exit point where EtE monitor can capture all traffic for this site. If the site "outsources" most of its popular content to CDN-based solutions then EtE monitor can only provide the measurement information about the "rest" of the content, which is delivered from the original site. For sites using CDN-based solutions, the active probing or page instrumentation techniques are more appropriate solutions to measure the site performance. A similar limitation applies to pages with "mixed" content: if a portion of a page (e.g., an embedded image) is served from a remote site, then EtE monitor cannot identify this portion of the page and cannot provide corresponding measurements. In this case, EtE monitor consistently identifies the portion of the page that is stored at the local site, and provides the corresponding measurements and statistics. In many cases, such information is still useful for understanding the performance characteristics of the local site.

The EtE monitor does not capture DNS lookup times. Only active probing techniques are capable of measuring this portion of the response times. Further, for clients behind proxies, EtE monitor can only measure the response times to the proxies instead of to the actual clients.

As discussed in Section 3, the heuristic we use to reconstruct page content may determine incorrect page composition. Although the statistics of access patterns can filter invalid accesses, it works best when the sample size is large enough.

Dynamically generated web pages introduce another issue with our statistical methods. In some cases, there is no consistent *content template* for a dynamic web page if each access consists of different embedded objects (for example, some pages use a rotated set of images or are personalized for client profiles). In this case, there is a danger that metrics such as the *server file hit ratio* and the *server byte hit ratio* introduced in Section 6 may be inaccurate. However, the end-to-end time will be computed correctly for such accesses.

There is an additional problem (typical for server access log analysis of e-commerce sites) about how to aggregate and report the measurement results for dynamic sites where most page accesses are determined by *URL*s with client customized parameters. For example, an e-commerce site could add some client specific parameters to the end of a common URL path. Thus, each access to this logically same URL has a different URL expression. However, service providers may be able to provide the policy to generate these URLs. With the help of the policy description, EtE monitor is still able to aggregate these URLs and measure server performance.

## 10 Conclusion and Future Work

Today, understanding the performance characteristics of Internet services is critical to evolving and engineering Internet services to match changing demand levels, client populations, and global network characteristics. Existing tools for evaluating web service performance typically rely on active probing to a fixed set of URLs or on web page instrumentation that monitors download performance to a client and transmits a summary back to a server. This paper presents, EtE monitor, a novel approach to measuring web site performance. Our system passively collects packet traces from the server site to determine service performance characteristics. We introduce a two-pass heuristic method and a statistical filtering mechanism to accurately reconstruct composition of individual page and performance characteristics integrated across all client accesses.

Relative to existing approaches, EtE monitor offers the following benefits: i) a breakdown between the network and server overhead of retrieving a web page, ii) longitudinal information for all client accesses, not just the subset probed by a third party, iii) characteristics of accesses that are aborted by clients, and iv) quantification of the benefits of network and browser caches on server performance. Our initial implementation and performance analysis across two sample sites confirm the utility of our approach. We are currently investigating the use of our tool to understand the client performance on a per-network region. This analysis can aid in the placement of wide-area replicas or in the choice of an appropriate content distribution network. Finally, our architecture is general to analyzing the performance of multi-tiered web services. For example, application-specific log processing can be used to reconstruct the breakdown of latency across tiers for communication between a load balancing switch and a front end web server, or communication between a web server and the storage tier/database system.

## References


[1] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching, and Zipf-like Distributions: Evidence, and Implications, In Proceedings of IEEE INFOCOM, March, 1999.

[2] Candle Corporation: eBusiness Assurance. `http://www.candle.com/`.

[3] S. Chandra, C. Schlatter Ellis and A. Vahdat. Differentiated Multimedia Web Services Using Quality Aware Transcoding. In Proceedings of IEEE INFOCOM 2000, Tel Aviv, March 2000.

[4] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing Energy and Server Resources in Hosting Centers. Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP), October, 2001.

[5] Cisco DistributedDirector., `http://www.cisco.com/`.

[6] F.D. Smith, F.H. Campos, K. Jeffay, and D. Ott. What TCP/IP Protocol Headers Can Tell Us About the Web. In Proceedings of ACM SIGMETRICS, Cambridge, May, 2001.

[7] A. Feldmann. BLT: Bi-Layer Tracing of HTTP and TCP/IP. Proceedings of WWW-9, May 2000.

[8] HP Corporation. OpenView Products: Web Transaction Observer. `http://www.openview.hp.com/`.

[9] R. Fielding, J. Gettys, J. Mogul, H. Nielsen, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. RFC 2616 , IETF, June 2001. http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[10] IBM Corporation. Tivoli Web Management Solutions, `http://www.tivoli.com/products/demos/twsm.html`.

[11] JavaServer Pages. http://java.sun.com/products/jsp/technical.html.

[12] T. Keeley. Thin, High Performance Computing over the Internet. Invited talk at Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'2000).

[13] Keynote Systems, Inc. `http://www.keynote.com`.

[14] B. Krishnamurthy and J. Rexford. Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement, pp.511-522, Addison Wesley, 2001.

[15] B. Krishnamurthy and J.Wang, On Network-Aware Clustering of Web Clients. Proceedings of ACM SIGCOMM 2000, August 2000.

[16] D. Mosberger and T. Jin. Httperf—A Tool for Measuring Web Server Performance. J. of Performance Evaluation Review, Volume 26, Number 3, December 1998.

[17] NetMechanic, Inc. `http://www.netmechanics.com`.

[18] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. 8th International Conference on Architectural Support for Programming Languages and Operating Systems, 1998.

[19] Porivo Technologies, Inc. `http://www.porivo.com`.

[20] R. Rajamony, M. Elnozahy. Measuring Client-Perceived Response Times on the WWW. Proceedings of the Third USENIX Symposium on Internet Technologies and Systems (USITS), March 2001, San Francisco.

[21] S. Seshan, M. Stemm and R. Katz. SPAND: Shared Passive Network Performance Discovery USENIX Symposium on Internet Technologies and Systems, 1997.

[22] Mark Stemm, Randy Katz, Srinivasan Seshan. A Network Measurement Architecture for Adaptive Applications. Proc. of IEEE INFOCOM, 2000.

[23] Software Research Inc. `http://www.soft.com`.

[24] `http://www.tcpdump.org`.

[25] H. Yu and A. Vahdat. Design and Evaluation of a Continuous Consistency Model for Replicated Services. Proceedings of Operating Systems Design and Implementation (OSDI), October 2000.