Detective Browsers: A Software Technique to Improve Web Access Performance and Security *

Songqing Chen and Xiaodong Zhang
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795
{sqchen, zhang}@cs.wm.edu

Abstract

The amount of dynamic Web contents and secured e-commerce transactions has been dramatically increasing in Internet where proxy servers between clients and Web servers are commonly used for the purpose of sharing commonly accessed data and reducing Internet traffic. A significant and unnecessary Web access delay is caused by the overhead in proxy servers to process two types of accesses, namely dynamic Web contents and secured transactions, not only increasing response time, but also raising some security concerns. Conducting experiments on Squid proxy 2.3STABLE4, we have quantified the unnecessary processing overhead to show their significant impact on increased client access response times. We have also analyzed the technical difficulties in eliminating or reducing the processing overhead and the security loopholes based on the existing proxy structure. In order to address these performance and security concerns, we propose a simple but effective technique from the client side that adds a detector interfacing with a browser. With this detector, a standard browser, such as the Netscape/Mozilla, will have simple detective and scheduling functions, called a *detective browser*. Upon an Internet request from a user, the detective browser can immediately determine whether the requested content is dynamic or secured. If so, the browser will bypass the proxy and forward the request directly to the Web server; otherwise, the request will be processed through the proxy. We implemented a detective browser prototype in Mozilla version 0.9.7, and tested its functionality and effectiveness. Since we simply move the necessary detective functions from a proxy server to a browser, the detective browser introduces little overhead to Internet accessing, and our idea can be implemented by patching existing browsers easily.

1 Introduction

Proxy servers are originally designed for caching static Web contents that are files stored in a Web server, and have been effectively used for this purpose.[1] Proxy servers also have to deal with other ever emerging types of Web contents. In this study, we examine the proxy's roles in processing dynamic Web contents and secured transactions, and present a software method to improve the Web access performance and security.

Dynamic Web contents are generated by programs executed at the requesting time. Although the response time to access a dynamic Web page is several orders of magnitude slower than that to access a static page, the amount of dynamic Web content services in commercial, government, and industrial applications has been dramatically increasing. Researchers have examined the percentage of dynamic contents in several highly popular Web sites including the Melissavirus site, the eBay site, the 1998 Olympic Winter Game site, and the Alexandria Digital Library site and others, and found the percentage ranges from 10% to 42% [10], [16], [17].

A number of methods had been devised to improve the performance of dynamic Web content services based on the current Web access infrastructure, focusing on effectively caching and processing dynamic web pages. One representative approach from the server side is to cache dynamic contents in the Web servers or in a dedicated storage close to the servers[10],[13],[14],[17],[19]. The approach from the proxy side is to restructure existing client side proxy servers to be capable of some Web server processing and caching functions for dynamic contents [8],[9],[15],[16]. Many studies have shown that caching dynamic contents at the server side is most effective and more appropriate [10],[12],[14],[17],[19]. In other words, dynamic contents are continuously changing and not suitable for client-side proxy caching, and thus it is not beneficial for a proxy to keep them. Although most proxy servers do not cache dynamic Web contents , a proxy has to make connections to Web servers and temporarily place a document while its dynamic nature is detected for the purpose of a replacement or deletion.

Internet E-commerce services have become popular and been provided by many company servers, and ever-increasing business transactions are completed online. E-commerce requires a secure channel to complete these transactions. Since the standard HTTP protocol is not sufficiently secure, the SSL was proposed [3], and commonly used for the secure data transmission, such as online shopping, on-line credit card payment. Since the secure transactions must not be intercepted or cached by any intermediary, a proxy has to tunnel the communication between the client and the server when such a content reaches the proxy. The involvement of the proxy can be a serious security concern besides the

*This work is supported in part by the National Science Foundation under grants CCR-9812187, EIA-9977030, and CCR-0098055.

[1] Proxy servers can also be used as firewalls for security reasons.

unnecessary processing overhead.

Instead of further investigating Web server caching or enhancing proxy caching ability for dynamic contents and the secured transactions, we have made our efforts to eliminate the client-side proxy processing overheads, and to provide a reliable way for secure transactions. This technique is also complementary to the server-side caching approach, further reducing response times to clients and removing unnecessary processing burdens on the proxy and unnecessary risks for secured transactions. In the rest of the paper, the term of "proxy" or a "proxy server" means a client-side proxy.

In this study, we will first show that this ignored proxy processing overhead is significant. We also look into the security risks of tunneling secured transactions. Conducting experiments on the proxy Squid2.3-STABLE4 [6], we have quantified this unnecessary processing overhead to show its significant impact on increased client access response times. We show that the average additional time spent on the Squid proxy to process a dynamic document is about 10%∼30% of the average response time of a direct access to the Web server with the caching ability for a dynamic document, and is 3∼10 times higher than that for accessing a static document in the Web server.

The performance results have led us to consider restructuring the organization of representative proxy systems, aiming to reduce or eliminate the processing overhead and the unnecessary risks in the proxy. For the dynamic contents, we discuss several possible techniques, and present technical difficulties that prevent us from achieving our goal. We conclude that it may not be possible to find an effective way to solve the overhead and the security problem by restructuring proxy servers.

In order to eliminate the overhead portion of the response time to access dynamic Web contents and unnecessary risks for secure transactions, we propose a simple but effective technique that enhances a standard browser, such as the Netscape/Mozilla, to be able to detect and schedule the outgoing requests, which we call a *detective browser*. Similar effort has been made to make clients more intelligent for the purpose of scalability in [18].

Upon an Internet request from the user, the detective browser can immediately determine whether the requested content is dynamic, or it requires a secured channel. If so, the browser will bypass the proxy and send the request directly to the Web server instead of going through the proxy. Otherwise, the request will be sent to the proxy as usual. We have implemented a detective browser prototype and tested its functionality and effectiveness upon a text based browser. We have also implemented it in the Mozilla.0.9.7. Since we simply move the necessary detection function from a proxy to a browser, and the detection can be done by scanning the URL only once, the detective browser introduces little overhead to Internet accessing, and our implementation can be patched to any existing browser easily to provide an additional option for users.

2 Sheltering dynamic contents in proxy and the overhead

Before we discuss how a proxy processes dynamic contents, we briefly overview its basic procedure of processing requests. Upon a Web page delivery request, the proxy first checks if the page is available and valid. If so, the proxy will deliver the page to the requesting client. If the page is available but it is not valid, the proxy will send an IMS (If-Modified-Since) message to the server to check whether the contents have been changed. The server will either send an updated page to the proxy or inform the proxy that the page has not been changed. The proxy will then either send the updated page or the original page to the requesting client. If the requested page is not cached in the proxy, then the request will be forwarded to the server. Upon receiving a reply from the server, the proxy will store the page in the local memory/disk, as well as forward a copy of the page to the requesting client. As soon as the header of the page is received from the server, the proxy is able to decide if the page is cacheable or uncacheable[2]. The replacement policy will work to reclaim the space by deleting LRU or unusable pages at a certain frequency.

2.1 How are dynamic contents processed in the proxy?

A representative proxy is the Squid proxy. It uses the same procedure to process requests for dynamic contents that are uncacheable in the proxy as it uses for static ones. The time and space used to process the dynamic contents are true overhead because the contents will not be reused by other clients. Following is a sequence of steps in proxy squid2.3-STABLE4 to process requests for dynamic contents.

- Upon receiving a request from the client (using function *clientReadRequest()*), the proxy parses the request and processes the headers (using functions *parseHttpRequest() and urlParse()*, respectively). The access right of the request will be checked (using function *clientAccessCheck()*) after the redirection is done (using function *clientRedirectDone()*). Then the proxy will process the request (using function *clientProcessRequest()*) by filling in some known attributes in the data structure, and determining if the requested content is in the proxy. Since the request is for a dynamic content, it has not been cached in the proxy.
- The proxy forwards the request to the Web server (using functions *clientProcessMiss()* and *fwdStart()*) after finding no peer proxy (using function *peerSelect()*). A TCP connection will be started (using function *fwdConnectStart()*), if a persistent connection is not used, via which the request will be sent to the server (using function *httpSendComplete()*).
- When the server returns the generated dynamic content, the proxy allocates a block of memory to store the content (using function *httpReadReply()*). The proxy detects that the content is uncacheable after parsing the header (using function *httpProcessReplyHeader()*). The proxy makes the dynamic content be private (using function *httpMakePrivate()*). (In contrast, if the content is static, the proxy will

[2]The dynamic or static nature of the Web contents is determined by the Web server, while whether the content is cacheable or uncacheable may be suggested by the Web server by setting the reply headers, and decided by the proxy. Not all static contents are cacheable, but most dynamic contents are uncacheable.

use *httpMakePublic()* to make the file public.) Even if the dynamic content will expire and not be reused, the proxy will buffer/store it in the local memory/disk (using functions *storeAppend()* and *storeSwapOut()*), and sends a copy to the client (using function *storeClientCopy2()*).

- Since the stored dynamic content is not usable again, it will be put into the LRU list, where the document will be replaced to release the space (using function *storeMaintainSwapSpace()*).

In each step, corresponding data structures will be created and allocated for processing. Related operations for these data structures are nested. The space and processing time involved in these operations delay the response time to the clients accessing dynamic contents.

2.2 Technical difficulties in eliminating the overhead

"Can we eliminate or reduce the overhead by detecting the dynamic content as early as possible in the proxy?". We first raised this question, and have tried to provide solutions for it. The dynamic nature of a request can be detected if the proxy further parses the request immediately after the request is received.

The implementation of this early detection in the proxy is straightforward with little overhead. With this early detection ability, a proxy has the following three alternatives to deal with a dynamic content request.

- *Making the Web server contact the client directly.* After detecting the dynamic nature of a request, the proxy asks the Web server receiving the request for the dynamic content to contact the client directly, instead of sending the document to the proxy. The proxy processing overhead will be eliminated because the proxy will never receive dynamic contents from Web servers. Unfortunately, this proposal is not practically useful although it is technically possible. In current Internet infrastructure, the data communications for a request from a client and its reply from the proxy are fixed in a pair of ports. In order to make the server contact the client directly, each client must be capable of listening to multiple connections because the reply for a request may come from a site that is different from the targeted destination. In addition, the socket used by the client to send the request needs to be terminated if the reply does not come from the proxy. A new connection between the client and the Web server must be created after that. The additional cost and existing Internet infrastructure make it impossible for this proposal to be implemented.

- *Making the client contact the Web server directly.* After detecting the dynamic nature of a request, the proxy declines a dynamic content request by sending a message back to the client to ask it to contact the Web server directly. Thus, the proxy processing overhead will be eliminated. However, the overhead times spent on the client request and the proxy declination will significantly increase the response time of accessing dynamic contents.

- *Making the proxy not shelter the dynamic contents.* After detecting the dynamic nature of a request, the proxy processes the request as the existing proxy does. However, the received dynamic content will not be cached. In other words, the content will be flushed out of the memory soon after it is forwarded to the client. This approach can certainly save the proxy space, but the processing time overhead and proxy load burden remain, because the space maintenance (using function *storeMaintainSwapSpace()* in Squid) is overlapped with the proxy operations on the clients' requests and servers' replies. This is the best that current proxy can do.

Discussing several alternatives based on existing proxy structures, we have presented the technical difficulties in eliminating the processing overhead even if the proxy detects the dynamic nature of a client request in the earliest stage.

3 Tunneling HTTP communications between clients and servers through proxy

Before we look into the details about how the proxy tunnels the HTTP communications, we have a brief overview at the SSL, which is atop of TCP/IP for the secured data transmission. SSL is an open, non-proprietary protocol proposed by Netscape Inc. [3], which has become the most common way to provide encrypted data transmission between Web browsers and Web servers in Internet. Built upon private key encryption technology, SSL provides data encryption, server authentication, message integrity, and client authentication for any TCP/IP connection. Most commercial Web sites provide secured services to the clients based on the SSL.

To tunnel the communication between the client and the server, a CONNECT method is used (instead of the normal GET). The CONNECT method is a way to notify the proxy to tunnel the arrived contents. The SSL session is established between the client who sends the request, and the Web server who generates the reply; the proxy between the two parties merely tunnels the encrypted data, simply passing bytes back and forth between the client and the server without knowing the meaning of the content.

3.1 How is the tunneling done in the proxy?

In Squid 2.3, the tunneling is done for both the client and the server as follows upon an SSL session request.

- Upon receiving a request for secured transactions from the client by function *clientReadRequest()*, the proxy parses the request and processes the headers by functions *parseHttpRequest() and urlParse()*. The access right of the request will be checked by function *clientAccessCheck()*, after the redirection is done by using function *clientRedirectDone()*. Then the proxy will process the request using function *clientProcessRequest()*, in which the CONNECT method will be identified and function sslStart() will be called to start the tunneling.

- The proxy uses function *sslReadClient()* to read the client request and queues it for writing to the server. Functions

sslSetSelect() and *sslWriteServer()* are then called to write data from the client buffer to the server side.

- When the proxy gets a reply from the Web server, it calls function *sslReadServer()* to read from the server and queue it for writing to the client. Functions *sslSetSelect()* and *sslWriteClient()* are then called to write data from the server buffer to the client side.

In Squid 2.5, the program becomes more complicated since Squid can encrypt or decrypt the connections, with the help of the OpenSSL[4]. However, the tunneling principle is the same.

3.2 The potential security problems of the proxy tunneling function

Besides the additional proxy overhead to tunnel the communications between the client and the server (we will present our measurement results later in the paper), the tunneling can be a potential source to cause security problems.

- *Bogus transactions.* IRCache group has reported their experiences on processing SSL requests in proxy [2] with following quotes. "Hackers have abused our service in the past by routing SSL requests through our caches". "We used to accept SSL requests, but some dishonest people abused our service by relaying bogus transactions through our caches. Because of these transactions, we received many complaints about credit card fraud and threats of FBI involvement. Thus, we now must deny all SSL requests". "Currently, the only way anyone could make a credit card purchase through proxies is if the origin server accepts such transactions over insecure, unencrypted connections".
- *Tunneling port can be a target.* The number of ports used for communications is limited. The port used for tunneling can be targeted for attacks even it is not in a reserved one. For example, it has been reported that a HTTP client CONNECTing to port 25 could relay spam via SMTP.
- *Implementation Bugs.* It is impossible to guarantee that an implementation of tunneling is bugs-free. Any small program bugs in tunneling can open security loopholes. This is only a potential problem.

We strongly argue that the proxy should not be involved in any secured transactions.

4 Analysis and measurement of the overhead for sheltering and tunneling

As discussed in Section 2.1, a dynamic document request will be processed by a sequence of four steps, the same as the static contents to be requested for the first time, in the proxy although the obtained dynamic document will not be used. In each step, processing time and/or storage space are consumed. In step 1, overhead operations involved are receiving, parsing, checking, redirecting the request, and a final request miss in the proxy. In step 2, the proxy will make a peer-selection and a socket connection to the target server. The major delay in this step comes from a TCP connection and slow start. In step 3, overhead operations involved are receiving, reading, parsing and storing the requested dynamic document. A memory block must be allocated to receive the document, and disk space is allocated to store the document. Finally in step 4, a replacement operation is used to delete the obtained document.

The time and space involved for the above operations are true processing overhead. In this section, we will quantify the overhead by measurement.

4.1 Processing overhead measurement for Sheltering

Figure 1 presents a basic measurement structure of the processing overhead for dynamic contents in proxy. There are two sets of measurements: (1) the average response time from a client to directly request a set of dynamic documents (represented by the solid arrow lines in Figure 1), and (2) the average response time from the same client to request the same set of dynamic documents through a proxy (represented by the dotted arrow lines in Figure 1). The difference between (2) and (1) is the average processing overhead in the proxy ideally. The Squid proxy is used in our experiments. The "client" we used in the experiments is not a normal browser, but is a text-based program. Its functions are to send out the requests, and wait for the reply from a Web server or from a proxy server. After the requested document is received, the client completes its job. It does not involve the normal browser functions of displays and other services (defined in Netscape/Mozilla), since they may cause more unstable factors.

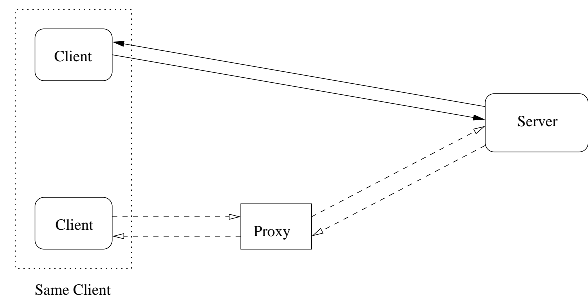

Figure 1: Basic measurement structure of the processing overhead for dynamic contents in proxy.

Intuitively, the overhead measurement should involve necessary instrumentation in the proxy and the use of workloads of dynamic Web contents. We did the instrumentation, but found that the results are very unstable with high intrusive effect. Examining the experiments, we realized that there is a potential disadvantage for using workloads of dynamic contents in our measurements. Dynamic contents often need timely services from the Web server, which may take up to multiple seconds for running service programs. Such dynamic changes and long delays may significantly disturb the server's load, and thus the measurement accuracy. For example, when we access the cgi-bin Web pages, the server needs to fork a process/thread to execute the program, and then sends the result back.

Considering the nature of the processing overhead in the proxy, we have found that the overhead is independent of dynamic contents. Because the Squid proxy processes a dynamic document exactly the same as it does a static document, if it is the first time to be requested, except that the proxy marks the a dynamic document as "private" for a future replacement (see Section 2.1). Thus, the processing overhead can be accurately measured by using static workloads.

In our experiments, the response time of static Web contents is much more stable and short (in the order of 0.1 seconds), and the Web server was not involved after the content is delivered. In addition, instrumentation in the proxy for measurements can generate additional overhead, possibly disturbing the measurement accuracy.

In our new experiments, the "client" program periodically sends requests to front pages of a set of selected and representative Web sites that are listed in Table 1. The selection is based on the following considerations. First, the selected Web sites are frequently accessed, it is likely that their front pages are always cached in the servers' memories, and the servers are sufficiently powerful to react to a huge amount of accesses. Thus, periodically sending requests to each Web site, we are able to obtain relatively stable response time. Second, four popular Web site types are covered in our experiments: ".com", ".edu", ".gov", and ".org". Finally, considering the distance, we selected Web servers on the east coast (www.ets.org, www.ieee.org, www.mit.edu, and www.whitehouse.gov), on the west coast (www.hp.com, www.intel.com, www.microsoft.com, and www.stanford.edu), and a local site (www.wm.edu).

The experiments are set up by running the Squid proxy (version squid2.3-STATBLE4) and client programs on a Pentium 3 Intel 1GHz processor machine with Redhat 7.1 Linux. The machine is dedicated to the experiments. We have minimized possible system intrusion when we measure the processing overhead for two reasons. First, a proxy is normally shared by multiple clients with context switching overheads. In contrast, our proxy serves only one client, minimizing the effect of unrelated overhead in the measurement. Second, in our experiments, the client and the proxy are co-located on the same machine, eliminating the networking transfer time between a client and the proxy. In practice, this networking time can potentially disturb the measurement of the processing overhead.

4.2 Quantifying the processing overhead for Sheltering

In order to cover the entire time period of a day, we conduct the measurement every hour 24 times a day. Besides the differences of type and distance, the front pages of the selected Web sites have different content lengths. We have repeated measurements 100 times for each site to calculate the average Squid proxy processing overhead. In our calculation, we discarded extremely large values that are not possible, and discarded the measured values when some of the web sites is temporally unavailable. Table 1 presents the content length, average processing overhead time, its variance, and the standard deviation of the measurements.

The measured processing overhead of each site is quite consistent, ranging from 0.1 seconds to 0.3 seconds. The average overhead time is 0.2 second. The quantum of the time overhead accounts for 10% to 30% of response time for a direct access to the Web server for a dynamic document, and 3 to 10 times higher than that for accessing a static document [19]. In order to verify that the measured result is machine independent, we ran the same experiments on an Intel Pentium 4 with 1.7 GHz processor, where the other configurations are exactly the same as in the Pentium 3 machine on which we did experiments. We obtained almost identical results.

In addition, space overhead is also involved because memory and disk are used to temporally store the dynamic contents. Besides the required space for the contents, related data structures will be allocated, which can be complicated. For example, the structure of *StoreEntry* is used, which includes other complex structures, such as *MemObject*, *HttpReply*, and *HttpHeader, HttpBody*, in turn.

4.3 Processing overhead for tunneling

It is difficult to get secured transaction workloads for experiments. Thus, we are not able to provide the measured overhead at the moment in this paper. Comparing operation differences, the tunneling overhead should be slightly lower than the sheltering overhead but at a comparable level, where no further request/reply header parsing is necessary. Our major concern of tunneling is not the performance overhead, but the security problem.

5 The design and implementation of detective browsers

Figure 2 presents the position of the detective browser in the Internet, which consists of an unmodified browser and its attached detector. Upon a client request, the detective browser first checks if the request is for a dynamic document. If so, the request will be directed to the targeted server, bypassing the proxy. Otherwise, the request will be routinely sent to the proxy. In Figure 2, the proxy is set on the client side (client-side proxy or proxy), and/or the server side (server-side proxy or reverse proxy). We try to eliminate the client-side proxy overhead.

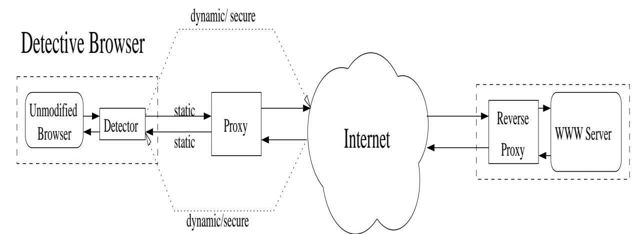

Figure 2: The detective browser model.

Site Names	Length (Bytes)	Overhead (s)	Variance	Standard Deviation	Locations
MIT.EDU	6919	0.094	0.025	0.158	MA
STANFORD.EDU	10197	0.118	0.010	0.102	CA
ETS.ORG	18903	0.131	0.009	0.093	NJ
WM.EDU	19160	0.117	0.001	0.033	VA
MICROSOFT.COM	23167	0.265	0.0003	0.005	WA
IEEE.ORG	26839	0.260	0.060	0.240	NJ
WHITEHOUSE.GOV	27655	0.271	0.11	0.33	DC
INTEL.COM	36831	0.273	0.003	0.055	CA
HP.COM	46180	0.299	0.078	0.279	CA

Table 1: The selected Web sites and measured average overheads for processing dynamic contents in the proxy.

5.1 The types of dynamic contents and secured transactions to be detected

Generally, dynamic Web contents have following features (1) documents are changed upon each access (e.g. cgi binaries [1], asp [7], fast-cgi, ColdFusion,etc.), (2) documents are the results of queries (e.g. the google search engine), and (3) documents embody client-specific information (e.g. cookies [11] or the SSIs. Generally speaking, these documents are the following types of dynamic contents: queries, SSI(Server Side Includes), and scripts.

- scripts: There are scripts written and executed in different ways. Generally, they could be in following formats:
  - cgi (Common Gateway Interface[1]): CGI is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that does not change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information. Generally, it can be used to connect a Web server with a wide range of applications. It could be written in different languages, as long as they are executable. Such as, the script written by Perl is always named with "pl" as its extension.
  - asp (Active Server Page[7]): The operations on asp page is done at the Web server. After the ASP codes are executed, all the asp code is stripped out of the page. A pure HTML page is all that is left and will be sent to the browser.
  - PHP (PHP: Hypertext Preprocessing[5]): PHP is a general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. Like the asp, the code is executed on the server and the client would receive the results of running that script.
- queries: The contents in all the search engines belong to this category. Users normally interact with the server by inputting some information into the form (for example, use "google" to search something by inputting the key word). Normally the server is connected to some background databases, so that the query could be executed and the result could be sent back to the user via the server. Queries could be implemented by forms, CGI, ASP, PHP, JSP, etc. No matter how the queries are implemented, they have the commons that a "?" appears in the URL when a client sends the request.
- SSI (Server Side Includes): SSI applies to an HTML document, provides for interactive real-time features such as echoing current time, conditional execution based on logical comparisons, and others. An SSI consists of a special sequence of tokens on an HTML page. As the page is sent from the HTTP server to the requesting client, the page is scanned by the server for these special tokens. When a token is found the server interprets the data in the token and performs an action based on the token data. The pages with the "shtml" as their name extensions are the SSIs, but some do not have a "shtml" name extension.

The detective browser is also able to detect the following requests for secured transactions.

- Secure ports HTTP requests: When the port 443 or 563 is given in the request following the host, then it is clearly a request for secured service from the server. 443 is for secured http, 563 is for snews [3]
- HTTPS requests: All netscape versions support the https requests, which is a secured http request, and is done on the SSL. Whenever you go to the American Express, Discover, or whatever to pay your bill online, it automatically leads you to the https.

The detective browser detects each type of dynamic contents and secured transactions as follows:

- Regarding scripts, there are the following.
  - For cgi, the URL must include the "cgi-bin", and the script ends with name extension of ".cgi" or ".pl". It will include a symbol of "?" when it is used for queries. The detective browser can easily determine the type by parsing the unique symbols.

[3]The TSL is working to make the secured and insecured services to share a common port, such as 80.

- For asp, all asp pages must have the extensions of ".asp", which is easy to check for in the URL. Also, when it is used for queries, the "?" must appear in the URL.
- For PHP, all PHP pages must have the extensions of ".php", which is also very easy to check for in the URL. Same as asp, when it is used for queries, the "?" must appear in the URL.

- Regarding queries, one or more keywords are always associated with each query. No matter how they are implemented, there must be a "?" in the URL followed by some keywords, so that we can simply check for this symbol in the URL. This can be also combined with searching for "cgi-bin".
- Regarding the SSI, we only process the pages with ".shtml". Since they all have the name extension of ".shtml", so it is easy to detect them in the URL.
- Regarding secured transactions:
  - For the HTTPS request, the https will be easily checked out on the URL since "https" will appear.
  - For the requests to ports 443 or 563, the port number must appear after the URL's host. So it is easily to check it out in the URL.

5.2 The software structure of the detector

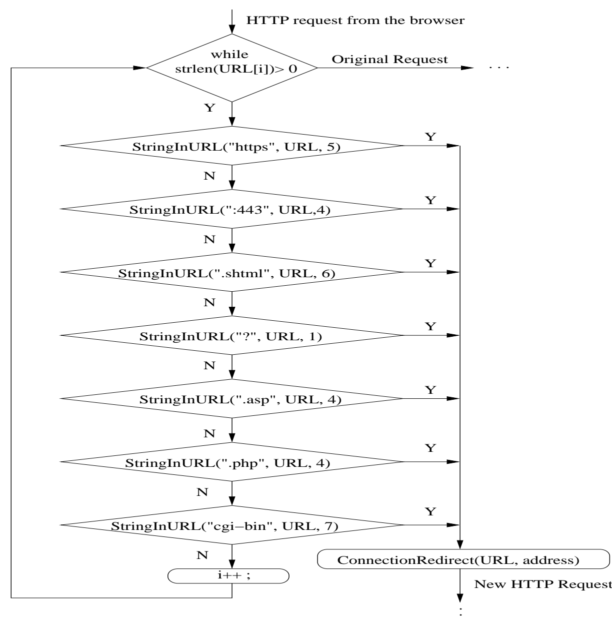

Figure 3: The operation flow diagram of the detector.

Figure 3 gives a high-level overview of how the detector is attached to an unmodified browser to construct a detective browser. The detector intercepts the HTTP requests before it is sent out, and then analyzes the request. A major component of the detector is the StringSearch function for searching the specific symbols representing dynamic contents in the URL or header. If such symbols are detected, the request will bypass the proxy. Another component is the ConnectionRedirect function for bypassing the proxy.

We have implemented the detector associated with a text-based browser for the convenience to measure its overhead. We had also implemented it on the Mozilla.0.9.7.(Currently it works on Linux Redhat 7.1.) It is very easy to patch the current standard browser(Netscape/Mozilla) so that it is capable of performing the detection function. We are making the detector as an user option of the browser.

5.3 Detector overhead measurements

The detector adds some processing time to each request although the URL is only scanned once. This overhead must be very small so that the detective browser is viable in practice. The quantum of the overhead must be trivial compared to the proxy processing overhead we have eliminated by the detective browser.

We measured the detector overhead in two ways. One way is to run the same set of requests with both the unmodified browser and the detective browser programs. The measured time difference is the detector overhead, where the system clock is used. Another way is to measure the number of clock cycles for executing the detector. Both measurements are the time interval between when a request is sent and when the reply is received completely. We obtained very consistent results from the two measurement alternatives. Table 2 presents average measured detector overhead results. Our measurements show that the detective browser only consumes 5 to 6 microseconds for each client access, which is trivial compared with the browser's performance gain, and insignificant from a client point view.

6 Detective Browser Performance Analysis

If there are not many dynamic requests, or secured transactions, why should we be bothered to make the patch on the browser? To quantitatively determine how effective the detective browser is, we analyzed access traces from NLANR [2]. The time period ranges from February 25 to March 4, 2002. Among the 9 different proxy sites from NLANR, we chose three covering the east coast, the Rocky Mountain area and the west coast of USA. Traces of the east is from proxy site "pb.us.ircache.net" located in Pittsburgh, Pennsylvania, (simplified as PB). Traces of the Rocky mountain area is from proxy site "bo.us.ircache.net" located in Boulder, Colorado, (simplified as BO). Traces of the west is from proxy site "sj.us.ircache.net" located in San Jose, California, (simplified as SJ).

6.1 The analyzed results from the traces

Table 3 gives breakdowns of different types of requests to the PB Squid proxy. We put SSI and Scripts together here, since we will give their detailed breakdowns below. The table shows that the sum of the queries, SSI and scripts occupies a high percentage

Site Names	Length (Bytes)	Original Access(s)	Detective Access(s)	Difference (s)	Overhead (μs)
MIT.EDU	6919	0.067	0.068	0.001	6
STANFORD.EDU	10197	0.245	0.245	0	5
ETS.ORG	18903	0.091	0.088	-0.003	5
WM.EDU	19160	0.250	0.249	-0.001	5
MICROSOFT.COM	23167	0.161	0.162	0.001	6
IEEE.ORG	26839	0.151	0.151	0	5
WHITEHOUSE.GOV	27655	0.060	0.060	0	5
INTEL.COM	36831	0.173	0.173	0	6
HP.COM	46180	0.297	0.297	0	5

Table 2: Measured detector overhead.

Date	Total	# Queries	Queries (%)	# SSI+Scripts	SSI+Scripts (%)	# Security	Security (%)
Feb. 25	1,286,520	221,232	17.20	48,628	3.78	9,114	0.71
Feb. 26	1,421,559	245,162	17.25	51,620	3.63	10,271	0.72
Feb. 27	1,299,109	241,427	18.58	53,631	4.13	9,732	0.75
Feb. 28	1,182,899	175,237	14.81	38,456	3.25	6,738	0.57
Mar. 1	998,905	101,228	10.13	25,220	2.52	6,306	0.63
Mar. 2	592,992	51,231	8.64	15,001	2.53	3,418	0.58
Mar. 3	615,945	50,544	8.21	16,196	2.63	3,751	0.61
Mar. 4	1,026,297	113,478	11.06	32,607	3.18	9,263	0.90

Table 3: The breakdowns of requests from PB

of the total requests, ranging from 11% to 23%, which can be bypassed from the proxy. Table 3 also shows that the number of requests for secured transactions is small. The main reason for this is that since 1998, the IRCACHE has stopped accepting the SSL requests. Those recorded by the access.log of squid is only those requests with 443 port. This has been further verified by our trace analysis on denied requests in the corresponding access logs and store logs. The total number of the detectable requests should be much higher than the number we have reported here.

Table 4 gives breakdowns of different types of requests to the BO Squid proxy. The table shows that the sum of the queries, SSI and scripts occupies a high percentage of the total requests, ranging from about 15% to 98%. The percentage of queries on March 2 and March 3 were very high. In two other periods, we had a similar observation. Looking into the traces, we learned that most of the quaries were from "www.yahoo.com". These are the proxy burdens that can be eliminated.

Table 5 gives breakdowns of different types of requests to the SJ Squid proxy. It shows a similar trend as that in Table 3. This table shows that the sum of the queries, SSI and scripts occupies a high percentage of the total requests, ranging from about 10% to 24%. These are also the proxy burdens that can be eliminated.

As an very important portion of all the traces, the queries are further analyzed to see different ways of their implementations. For the brevity, we gave the breakdowns of the queries to the BO Squid proxy as a representative case.

Table 6 shows that ASP is used more frequently than CGI, PHP, PL in implementing queries. Since SSIs and different kinds of scripts may be intertangled together, Table 7 shows us that CGI and ASP are used more than others.

Furthermore, we find some data from publications, which confirms our analysis. The Melissa virus online forum traces and results can be used as references for estimating the effects of the detective browser to dynamic contents of the ASP type. Based on the data published in [19], if the normal client accesses are going through a client-side proxy, the detective browser is able to reduce the average response time by 12.7%. If reverse-proxy caching is also used, then the reduction of the average response time to clients will be 33.3%. Also the proxy's load burden will be reduced at least 10%, since requests for dynamic contents bypass the proxy.

Regarding CGI, the ADL(Alexandria Digital Library) traces and results can be used as a reference [13]. Since among 69337 requests, 28663 are for dynamic contents, then with our detective browser, the proxy's load burden can be reduced at least 41.3% if the client accesses always go through the client-side proxies. The reduction of the average response time to the clients will be 11.1%.

The AT&T internal recruiting database is considered as a reference for evaluating the detective browser's effects to queries [12]. If the detective browser is used by the client, then the average response time can be reduced by 18.2%.

Date	Total	# Queries	Queries (%)	# SSI+Scripts	SSI+Scripts (%)	# Security	Security (%)
Feb. 25	197,332	25,254	12.80	7,203	3.65	1,264	0.64
Feb. 26	328,435	51,005	15.53	12,610	3.84	3,135	0.95
Feb. 27	324,658	44,200	13.61	11,505	3.54	2,519	0.78
Feb. 28	323,736	45,005	13.90	11,748	3.63	2,336	0.72
Mar. 1	470,783	251,796	53.48	9,871	2.10	3,517	0.75
Mar. 2	1,893,541	1,834,187	96.87	5,662	0.30	12,073	0.64
Mar. 3	1,947,952	1,895,764	97.32	5,803	0.30	14,301	0.73
Mar. 4	384,462	173,174	45.04	8,838	2.30	2,430	0.63

Table 4: The breakdowns of requests from BO

Date	Total	# Queries	Queries (%)	# SSI+Scripts	SSI+Scripts (%)	# Security	Security (%)
Feb. 25	390,915	73,687	18.85	18,462	4.72	2,251	0.58
Feb. 26	201,212	9,398	4.67	9,031	4.49	1,371	0.68
Feb. 27	202,377	12,930	6.39	9,517	4.70	1,376	0.68
Feb. 28	240,133	18,564	7.73	9,090	3.79	1,592	0.66
Mar. 1	159,721	16,193	10.14	6,012	3.76	1,071	0.67
Mar. 2	161,702	12,469	7.71	4,582	2.83	1055	0.65
Mar. 3	115,392	11,354	9.84	4,170	3.61	844	0.73
Mar. 4	141,240	9,450	6.69	4,895	3.47	1,014	0.72

Table 5: The breakdowns of requests from SJ

6.2 What is the detective browser not able to detect?

Besides the four types of common dynamic contents (cgi, queries, asp, and cookies), the detective browser can also detect following two dynamic content types: (1) Method (the request method other than “GET” and “HEAD”), and (2) Auth (a request with an authorization header). However, the detective browser is not able to process the following uncacheable Web contents, since they are only designated by the Web servers’ response:

- Pragma: the response is explicitly marked uncacheable with a “Pragma:no-cache” header.
- Cache-control: the response is explicitly marked uncacheable with t he HTTP 1.1 cache-control header.
- Response-status: the server response code does not allow the proxy to cache the response.
- Push-content: the content type “multipart/x-mixed-replace” is used by some servers to specify dynamic content.
- Vary: the vary is specified in the header.

The usage of the above dynamic content types is low. We believe there may be some other rare requests that are not well filtered by the current version of the detective browser. The detective functions will be upgraded as the formats of dynamic contents and secured transactions are updated.

7 Conclusion

We have identified and quantified two overhead sources in the proxy for processing dynamic Web contents and secured transactions. We have also shown that this overhead source could not be easily eliminated from the proxy, and security concerns can be serious for proxy to tunnel secured transactions. Avoiding the delay caused by proxy processing overhead for accessing dynamic contents, and addressing the security concerns, our detective browser actively determines if a request should go directly to the Web server bypassing the proxy, or go through the proxy. We have shown the effectiveness of this approach, and its low overhead in implementations.

Acknowledgment: The work is a part of an independent research project sponsored by the National Science Foundation for author Xiaodong Zhang who serves as the NSF Program Director of Advanced Computational Research. The comments from the anonymous referees are helpful and constructive.

References

[1] http://hoohoo.ncsa.uiuc.edu/cgi/overview.html

[2] http://www.ircache.net/

[3] http://www.netscape.com/eng/ssl3/

[4] http://www.openssl.org/

[5] http://www.php.net/

[6] http://www.squid-cache.org/

[7] http://www.takempis.com/asp1.asp

[8] K. Seluk Candan, Wen-Syan Li, Qiong Luo, Wang-Pin Hsiung, and Divyakant Agrawal, “Enabling Dynamic Content Caching for Database-Driven Web Sites”, in *SIGMOD*, 2001

Date	Total	# CGI-Q	CGI-Q (%)	# ASP-Q	ASP-Q (%)	# PHP-Q	PHP-Q (%)	# PL-Q	PL-Q (%)	OTHERS (%)
Feb. 25	25,254	1,000	3.96	2,893	11.46	767	3.04	288	1.14	80.41
Feb. 26	51,005	2,745	5.38	5,676	11.13	2,943	5.77	649	1.27	76.45
Feb. 27	44,200	2,322	5.25	5,039	11.40	1,990	4.50	756	1.71	77.13
Feb. 28	45,005	1,401	3.11	4,243	9.43	1,132	2.52	341	0.76	84.19
Mar. 1	251,796	1,356	0.54	3,854	1.53	1,029	0.41	377	0.15	97.37
Mar. 2	1,834,187	609	0.03	771	0.04	553	0.03	87	0.00	99.89
Mar. 3	1,895,764	284	0.01	753	0.04	142	0.01	41	0.00	99.94
Mar. 4	173,174	1014	0.59	3,751	2.17	912	0.53	218	0.13	96.60

Table 6: The breakdowns of queries from BO

Date	Total	# SHTML	SHTML (%)	# CGI	CGI (%)	# ASP	ASP (%)	# PHP	PHP (%)	# PL	PL (%)
Feb. 25	7,203	597	8.29	1,401	19.45	3,343	46.41	746	10.36	1,116	15.49
Feb. 26	12,610	1,601	12.70	2,807	22.26	5,638	44.71	971	7.70	1,593	12.63
Feb. 27	11,505	1,311	11.40	1,981	17.22	5,473	47.57	1,126	9.79	1,614	14.03
Feb. 28	11,748	1,741	14.82	1,738	14.79	4,907	41.77	1,086	9.24	2,276	19.37
Mar. 1	9,871	1,019	10.32	1,783	18.06	3,421	34.66	1,378	13.96	2,270	23.00
Mar. 2	5,662	336	5.93	3,052	53.90	690	12.19	344	6.08	1,240	21.90
Mar. 3	5,803	204	3.52	2,843	48.99	996	17.16	192	3.31	1,568	27.02
Mar. 4	8,838	1,415	16.01	1,312	14.84	3711	41.99	1,032	11.68	1,368	15.48

Table 7: The breakdowns of the SSI and Scripts from BO

[9] Pei Cao, Jin Zhang, and Kevin Beach, "Active Cache: Caching Dynamic Contents on the Web", in *Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing(Middleware '98)*, Mar. 1998.

[10] Jim Challenger, Arun Iyengar, and Paul Dantzig, "A Scalable System for Consistently Caching Dynamic Web Data", in *Proceedings of the IEEE INFOCOM '99*, Mar. 1999.

[11] http://home.netscape.com/newsref/std/cookie_spec.html

[12] Fred Douglis, Antonio Haro, and Michael Rabinovich, "HPP: HTML macropreprocessing to support dynamic document caching", in *Proceedings of USENIX Symposium on Internet Technologies and Systems*, 1997.

[13] Vegard Holmedahl, Ben Smith, and Tao Yang, "Cooperative Caching of Dynamic Content on a Distributed Web Server", in *Proceedings of the Seventh IEEE Intl. Symposium on High Performance Distributed Computing*, July 1998.

[14] Arun Iyengar and Jim Challenger, "Improving web server performance by caching dynamic data", In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 49–60, December 1997.

[15] Qiong Luo, Rajasekar Krishnamurthy, Yunrui Li, Pei Cao, Jeffrey F. Naughton, "Active Query Caching for Database Web Servers", In *the 3rd International Workshop on the Web and Databases (WebDB'2000)* in conjunction with the *ACM SIGMOD Conference*, May 2000.

[16] Ben Smith, Anurag Acharya, Tao Yang and Huican Zhu, "Exploiting Result Equivalence in Caching Dynamic Web Content",in *Proceedings of Second USENIX Symposium on Internet Technologies and Systems(USITS99)*, Oct. 1999

[17] Jian Yin, Lorenzo Alvisi, Mike Dahlin, Arun Iyengar, "Engineering server-driven consistency for large scale dynamic web services", *WWW10*, May 2001.

[18] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler, "Using smart clients to build scalable services", *Proceedings of 1997 USENIX Annual Technical Conference*, Anahein, California, January 6-10, 1997.

[19] Huican Zhu and Tao Yang, "Class-based Cache Management for Dynamic Web Content", in *Proceedings of the IEEE INFOCOM '01*, May 2001.