Where Have You Been?
A Comparison of Three Web Tracking Technologies

Onn Brandman, Hector Garcia-Molina, Andreas Paepcke
Stanford University
{onn, hector, paepcke}@cs.stanford.edu

Extended Abstract

"Web tracking" is the process of gathering Web access information. Such access information can then be used for improving future browsing and access, or as input for data mining processes that analyze patterns. We have implemented three Web tracking systems, and evaluated and compared their performance and characteristics. In the first system, rather than connecting directly to Web sites, a client issues URL requests to a proxy. The proxy connects to the remote server and returns the data to the client, keeping a log of all transactions. The second system uses packet "sniffers" to detect and log all HTTP traffic on a subnet. The third system periodically collects browser log files and sends them to a central repository for processing. Each of the systems differs in its advantages, pitfalls, and performance.

In the full paper we present a comprehensive comparison and discussion of the three tracking systems. The full papers includes:

- *Description.* A detailed description of each type of scheme, and the options for implementing the schemes.
- *Comparison of Features.* A comparison of the features, strengths and weaknesses of each type of system. The features considered include privacy vulnerabilities, ease of configuration, types of data that can be obtained, scalability, and impact on the end-user.
- *Implementation Issues.* A discussion of what existing software can be used to build these tracking systems, how difficult it is to run the system, and how output data can be collected.
- *Performance Comparisons.* A discussion of the key performance parameters, such as CPU overhead of proxies and sniffers, cache hit rates, and delays introduced by proxies. We summarize performance results that have been obtained elsewhere, and we complement them with some of our own experimental results focusing on the CPU consumption of a packet sniffer and a proxy under identical traffic loads.
- *Integration Issues.* A discussion of the issues that arise when integrating data generated via different tracking mechanisms.

In the rest of this extended abstract, we highlight some of the material from the full paper. A *first draft* of the full paper is available at http://diglib.stanford.edu/~onn/tracking.ps.gz.

1. Description

Very briefly, a *proxy* can be thought of as a Web server that uses the Web as its database rather than a local disk. Web browsers access the proxy, rather than connecting directly to the Web to download Web pages. The proxy forwards the request to its final destination, and returns the results to the client. The use of a proxy requires explicit client side configuration: the location of the proxy on the network must be specified. Proxies are often used to add value to the Web interaction [5]. For example, after fetching a page, a proxy might examine the outgoing links of that page, and prefetch the corresponding pages. This may speed up future page requests.

A *sniffer* is a process that examines all network traffic that reaches the machine it runs on. HTTP traffic generated by Web browsers is a subset of this traffic, and the sniffers must find and analyze the HTTP packets. For example, a sniffer must filter out packets concerned with email, or with Network File System (NFS) traffic. By examining the HTTP requests, the sniffer can obtain Web access information similar to that obtained by a proxy.

Browsers can generate local *log files* that record activities. For example, the Netscape Navigator browser log file records the frequencies with which all URLs are visited, and the times of the first and most recent download. Other systems have been implemented to record more complete data[14]. Log data is generated is a distributed fashion, so it must somehow be gathered into a more global access log.

2. Comparison of Features.

Table 1 below summarizes the characteristics of each type of tracking system. The full paper discusses these characteristics in more detail.

Table 1: Summary of System Characteristics

	Proxy	**Sniffer**	**Browser log file**
Major Complications in employing system	Users must configure web browsers	Sniffers must run as root. Nework topology must be carefully studied.	Users must download client side program and run it all the time Log file must be deciphered
Privacy/Security concerns	Low: proxy can only observe http requests for participating users	High: sniffer can observe passwords, email, HTTP traffic, etc, for *all* users	High: Interrupting log activity is complicated. Client side process can be a Trojan horse; can introduce viruses and push any client-side information onto the network
User can bypass system	Easy	Impossible	Hard
Scaleablity: Computational cost	Medium: proxy only needs to work when HTTP requests are made	Variable: sniffer must read all network traffic, regardless of type or origin	Low: process sleeps most of the time. Size of information pushed from client to server is proportional to relevant information
Scaleability: ability to handle many users	High: can employ multiple proxies	Depends on network topology: may require a large number of bulky processes to sniff all users	High
Client side configuration	Web browser configuration	None	A client process must be initiated, restarted whenever it crashes or is killed
User browsing experience	Altered; some delays, possible speed up if Web cache is used	Unchanged	Unchanged
Failure consequences (other than loss of tracking data)	User looses Internet connectivity	None	Could crash or slow down user machine
Data flow	Realtime	Realtime	Batches
Sensitivity to network topology	None	Heavy	None
Potential of malicious user to fabricate bad data	None	None	Heavy

3. Implementation Issues.

To illustrate the types of issues discussed in the full paper, here we focus on one such issue, the deployment of sniffers on a network. In particular, two complications arise. First, the network topology must be taken in account when choosing how many sniffers are needed and where they should run. Second, special privileges are needed to run the sniffers, and in some cases to configure the network equipment.

For a sniffer to track a user, the sniffer must either

1. Run on a machine residing on the same hub as the user.
2. Run on a machine residing on a hub located on the path between the user machine and the subnet's root hub. The root hub is the hub connected to the Internet.

These two possibilities are illustrated in Figure A. The leftmost configuration corresponds to bullet one above and the other to bullet two. Each configuration tracks four users.

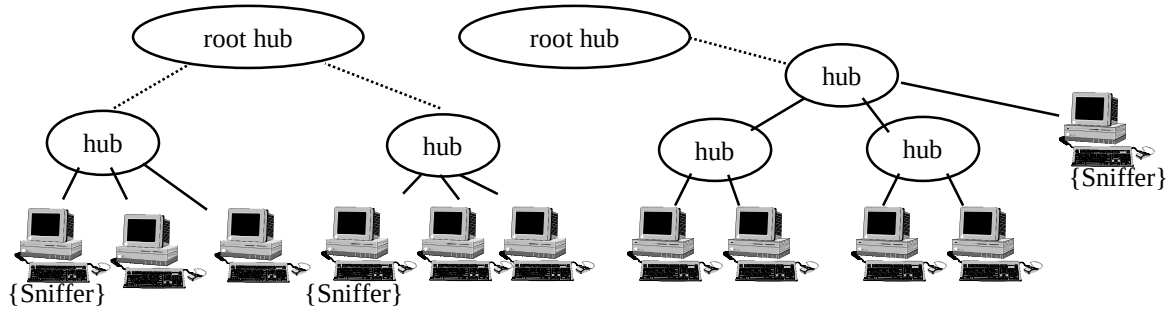


Figure A. Two sniffer configurations

Both solutions require the sniffer's hub to broadcast all traffic to the sniffer. The first solution requires an array of sniffers to be deployed if the users reside on different hubs. The second solution requires fewer sniffers, but inter-hub traffic on hubs without resident sniffer processes may be lost. It is also clear that a single sniffer process running at the root hub could monitor all users on the subnet. However, this would require that the root hub broadcast all traffic. It is usually the case that the root hub does not broadcast traffic precisely to prevent a sniffer from monitoring network traffic. This precaution is taken because sniffers could record passwords, email, and other kinds of packets, in addition to HTTP traffic. Thus, to monitor a subnet using a single sniffer may require special configuration of network hardware; that is, the root hub must be configured to broadcast all traffic to the sniffer.

Additional complications arise when a sniffer and proxy run on the same subnet. The sniffer tracks Web transactions by selecting packets sent to port 80 of a remote host. If the sniffer resides between the proxy and the outside network then the sniffer detects all traffic generated by the proxy. Note that in this case the individual users are hidden from the sniffer. On the other hand, if the sniffer lies between the user machines and the proxy then the sniffer configuration requires special attention. Since users are connecting with the proxy rather than the remote Web servers, the sniffer will not detect web traffic. To catch the Web traffic, the sniffer must be configured to monitor the proxy's port rather than the standard HTTP port (port 80).

The privacy and security risks of the sniffing approach are exacerbated because sniffers must run as root user. This is because sniffers control the computer's network hardware. If the sniffers are run by a single organization with few machines that all traffic passes through, then the requirement for special privileges may not pose too much of a problem. However, if the users are distributed across a set of different subnets, it is likely impossible to deploy the needed sniffers and monitor all subnets. This retards the ability of a sniffer based system to scale to an arbitrary number of users.

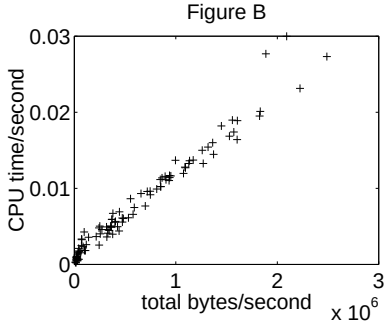
For our experiments, we used the *tcpdump*[18] packet sniffer. A variety of packet sniffers are freely available on the net. However, most are built on top of tcpdump. The results returned by tcpdump require some processing before they can be fed to a tracking system. In particular, the packets are displayed in hexadecimal format rather than *ascii* characters. Also, tcpdump captures a fixed number of bytes from each packet. Capturing more bytes from each packet reduces the number of packets that fit in the buffer and can result in lost packets when traffic is heavy. In our experiment, we found that capturing the first 100 bytes of each packet was sufficient since this allowed us to record both the IP addresses and 40 characters from the URL of each HTTP GET request.

4. Performance Comparisons.

In this extended abstract we summarize some of our experimental comparisons; others are presented in the full paper. The comparison we focus on was done to determine under what circumstances a proxy might be more efficient than a sniffer. We conducted an experiment to compare the CPU consumption of the *SQUID*[6] proxy and the tcpdump packet sniffer. The tracking systems were tested repeatedly under different ratios of noise and HTTP traffic. The noise traffic consisted of email, NFS, FTP, DNS lookup, and other traffic on ports other than 80. Noise traffic was not specifically generated for the experiment but was part of the normal daily traffic on our network. To generate the HTTP traffic, we automatically downloaded Web data (URL's of images, Web pages, etc.) at a fixed rate. The rate ranged between 1 to 130 pages per minute depending on the test run. The URL's came from previously collected trace data consisting of 70,000 accesses by members of the Stanford Database Group. The sniffer and proxy ran concurrently on the same hardware so that both monitored the same load under the same conditions. The hardware used was a Sun[16] Sun Ultra 2 containing two 200 MHz processors and 256 MB memory.

CPU time was measured by using the *time* command. Figure B shows the CPU consumption rate of the sniffer vs. the total number of bytes per second (HTTP plus noise traffic) on the network. The X-axis displays the total traffic in bytes per second while the Y-axis shows the sniffer's CPU consumption rate as the fraction of available CPU time devoted to sniffing. Each point represents one test run of the sniffer.

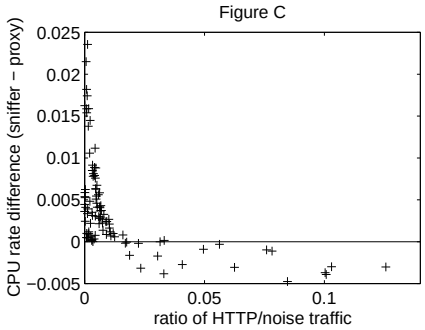
Figure B



The sniffer's CPU consumption appears linear with respect to the total traffic on the line. This implies that most of the sniffer's work consists of filtering HTTP packets out from large volumes of other traffic. In addition to filtering, a sniffer logs selected information about HTTP packets. However, we found the cost of filtering overshadows the cost of logging.

We also measured the CPU time of the proxy. The supporting graphs are shown in the full paper. The proxy's CPU consumption is linear with respect to HTTP traffic. Noise network traffic has no effect. Although both the proxy and sniffer are linear, the proxy's plot follows a steeper slope. The proxy's slope is roughly 10^{-6} CPU seconds per byte while the sniffer's slope is approximately 10^{-8} CPU seconds per byte. It follows that the per HTTP request cost of the sniffer is 100 times lower than that of the proxy. This is because the sniffer merely reads data while the proxy must fork a new process, set up an HTTP connection, and stream data back to the client. Other then updating the trace database, reading HTTP traffic is a only subset of a proxy's functionality, while it is the entirety of the sniffer's task.

Figure C shows that the ratio of HTTP to noise traffic determines whether the sniffer or proxy consumes less CPU cycles. The Y-axis shows the difference in CPU consumption rate between the sniffer and the proxy. When the difference is zero, the proxy and sniffer consume the same CPU resources. A positive difference indicates that the proxy is more efficient, while a negative difference indicates the superior sniffer performance. The X-axis displays the ratio of HTTP traffic to noise traffic.



The difference becomes negative when the traffic ratio is greater than 0.02. That is, when the network is dominated by noise traffic, the sniffer becomes more computationally expensive than the proxy. However, as the network becomes more and more dominated by HTTP traffic, the sniffer becomes a more efficient solution. While 0.02 seems a small number, we found in practice that was surprisingly high. It took high Web data download rates (130 per minute) and low noise traffic to surpass this ratio. We hypothesize that the majority of locations on the network will have a HTTP to noise ration smaller than 0.02. Consequently, a we believe that a proxy will be more efficient in most cases.

5. Integration Issues.

To address the topic of scaleablity, we introduce two distributed scenarios. Common to both scenarios is a distributed set of Web tracking processes acting as the source of all trace information. Also, common to both scenarios is the goal of building a single complete trace database. In the first scenario, the Web tracking processes communicate directly with the system that manages the complete trace database. Updates can be done in batches or in realtime update/insert commands issued to the DBMS. The second, more distributed, solution involves maintaining a distributed set of trace databases, each comprising some subset of the total trace information. These databases are then used to feed a common complete trace database. The schema of the databases may differ based on differing schemas from different Web tracking systems. Challenges in these integration scenarios include integrating different schemas, dealing with duplicate information, whether to push or pull the information from the sources, and how to cope with high data rates. More discussion of these issues can be found in the full paper.

[1] http://squid.nlanr.net/Squid/, Squid Internet Object Cache. 1999.
[2] http://www.netscape.com, Netscape. 1999.
[3] http://www.microsoft.com, Microsoft Corporation. 1999.
[4] http://www.cs.wpi.edu/~sigmet98/ Paul Barford, Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In ACM SIGMETRICS Performance Evaluation Review Vol. 26, No. 1 (June 1998), Pages 151-160, June 26 1998
[5] http://www.w3.org/Conferences/WWW4/Papers/155/ Marc Abrams,Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, Edward A. Fox. Caching Proxies: Limitations and Potentials. In WWW4, 1995
[6] Bob Barr, Sung Yoo, Tom Cheatham, Network Monitoring System Design, In Proceedings of the ACM SIGCSE, 1998
[7] http://www.cs.vt.edu/~chitra/pubs.html Marc Abrams and Stephen Williams, Complementing Surveying and Demographics with Automated Network Monitoring, In World Wide Web Journal June1996, volume 1, no. 3
[8] Cunha, C.R., A. Bestavros, and M.E. Crovella, Characteristics of WWW Client-based Traces, In TR: BU-CS-95-010, Boston University July 1995
[9] http://www.sun.com Sun Microsystems Inc.
[10] http://www.cs.ndsu.nodak.edu/~rousskov/research/cache/squid/profiling/papers/ Alex Rousskov, Valery Soloviev, On Performance of Caching Proxies in WWW Journal

[11] ftp://ftp.ee.lbl.gov/tcpdupm.tar.Z Lawrence Berkeley Laboratory network Research Group, tcpdump