

MOT: Memory Online Tracing of Web Information System¹

Yun Mao, Kang Chen, Dongsheng Wang, Weimin Zheng

Dept. of Computer Science and Technology

Tsinghua University

Beijing 100084, P.R. China

Xiaotie Deng

Department of Computer Science

City University of Hong Kong

Hong Kong

Abstract

With advances in World-Wide Web applications and technologies, research on measurement and modeling of Internet and Web-based information systems has become increasingly important. This paper focuses on continuously monitoring Web traffic by packet sniffing on high-speed links, which is the foundation of analyzing theoretical models of Web characteristics and evolution. The author presents MOT, a memory online tracing system of Web traffic. MOT parses all packets in memory directly without involving unnecessary I/O operations with magnetic disks to enhance the system performance. Event-driven design pattern and several other techniques are adopted in MOT to overcome the difficulties of buffering huge volume of traffic. MOT provides a fast, accurate and safe way to obtain the source data for many Web-related studies.

Keywords

Web Measurement, Information Modeling, Packet Sniffing, Event-driven Architecture

1. Introduction

Web applications become more and more dominant on today's Internet. For constructing high performance Web-related applications and tracking Web characteristic variations, people need detailed investigation of the underlying protocols (including TCP/IP, HTTP, etc.) and user access patterns. In fact, the size, growth, complexity and diversity of the Internet make it impossible to understand without measurement and analysis.

However, sustained, rapid growth, increased e-commerce services, and proliferation of new applications have combined to change the characteristic of the Internet in recent years. The sheer volume of the traffic and the high capacity of the trunks have rendered Web traffic measurement a more challenging endeavor.

Among all measurement methods, the passive packet sniffing has more advantages than others. It provides all useful information, such as full HTTP headers, TCP events and precise timestamps, and does not affect the Internet performance, and remains transparent to end-users. Existing passive measurement systems, such as BLT^[1] and HTTPDump^[2] are not well suited for monitoring on high-speed links. Frequent I/O operations of magnetic disks become the bottleneck of system performance. Although disk arrays, i.e. RAID, provide higher transfer rate, it is expensive to use and not scalable.

In this paper we present the design and implementation of *MOT*, a Memory Online Tracing system by packet sniffing. Instead of storing all the sniffed packets in hard disks before parsing, *MOT* parses these packets in memory directly. Therefore, a significant amount of I/O operations are reduced and the overall performance is improved. Meanwhile, sensitive information in HTTP contents is not saved in our system during the parsing process, which prevents some potential privacy problems. We adopt the event-driven architecture^[3] to implement *MOT*, which is proved to be scalable and suitable for highly concurrent applications.

The rest of this paper is organized as follows. Section 2 introduces related work. In section 3, the design and implementation of memory online tracing system are described in detail. Section 4 outlines the studies enabled by *MOT*. Experimental results are illustrated in section 5. Finally, section 6 concludes this paper.

¹ This work is supported in part by National Natural Science Foundation of China (grant No.F020303) and Development Plan of the State Key Fundamental Research. G1999032702

2. Related Work

Some universities and research institutes have used packet level data to extract Web data. A group at Virginia Tech University has developed the HTTPDump^[2] to extract HTTP headers from Tcpdump^[4] traces. The performance of HTTPDump is not capable of sustaining continuous tracing of 10Mbps Ethernet. AT&T Research Lab has the BLT (Bi-Layer Trace)^[1] project, in which the challenges and advantages of packet monitoring are discussed in detail. The log file format is designed carefully and precisely to meet many analysis requirements. BLT has provided the foundation of several Web performance studies^[5, 6]. Some IBM researchers have studied their Web server during Olympics with partial packet level data combined with server log files^[7] and analyzed the TCP behavior in Web server. A group at Berkeley has developed a HTTP tracing module based on ISPE (Internet Protocol Scanning Engine) to monitor the HomeIP network (A modem dial-up network center with 10Mbps bandwidth) continuously. Client's activities are traced and analyzed.^[8]

A more generic measurement platform, called Windmill, is described in [9]. This platform is meant to run continuously providing the means to perform several "experiments" without ever terminating it. Since people in different experiments might be interested in different packet streams, the platform has the ability to dynamically modify the packet-filter expression. No special treatment, however, is given to HTTP protocol.

3. Design and Implementation of *MOT*

3.1 Feasibility of *MOT*

Many monitoring system choose hard disks as their temporary buffers to store the incoming packets. The more bandwidth they monitor, the more I/O operations they involve and then the more overall performance is degraded. Furthermore, all the browsing contents of users, including passwords, credit card numbers and private letters, are temporarily logged into disks; this could lead to privacy and security problems. *MOT*, Memory online tracing will not induce these problems. Although the memory resources are limited, we believe that memory online reconstruction is feasible for the following reasons:

(1) Not all Internet traffic is Web data. The useless non-Web data include: (a) Non-TCP packets, such as

UDP, ICMP and IGMP. Due to the popularity of streaming playback of music and news, as well as real-time voice telephoning and conferencing, the number of UDP packets is increasing rapidly. (b) Traffic of familiar applications from well-known ports, such as port 23 for telnet, port 25 for SMTP/E-SMTP, port 110 for POP3, port 21 and 20 for ftp and ftp-data, etc. These applications occupy a considerable proportion of the entire Internet traffic. We adopt Linux Socket Filter, a Linux implementation of BPF-like kernel filter, to pre-filter these useless packets in the kernel space, which is efficient and able to avoid many unnecessary kernel-user space switches made by system calls^[10].

(2) Protocol headers, such as TCP headers, IP headers, and link-level frame headers are not necessary to save. Our experimental results in section 5 illustrate the traffic composition in bytes on the CERNET-US network link. It shows that the Web traffic without overhead of TCP/IP protocol headers occupies about 44.0% of entire network traffic.

(3) It is arbitrary to filter out all information other than port 80 because not all the Web servers serve on port 80 and because proxy servers have their own ports. However, Web traffic can be easily identified from their first 200 bytes according to the specification in RFC 2616^[11]. So, TCP connections of other protocols can be identified as useless flows and removed from memory in their early stages of appearance.

(4) The tracing system cares only about HTTP headers. HTTP contents should not stay in memory if possible. In most HTTP response headers, content length is specified so that we can tell whether the packets are part of HTTP contents or not from their sequence numbers. After eliminating these contents, the memory requirement for continuously online tracing is considerably small. Anja Feldmann proposed this idea and we have given a concrete implementation. Detailed description of how to remove HTTP contents is in section 3.

(5) The process of reconstruction, parsing and logging can be incorporated since it is not necessary to parse data after transmission is over.

To sum up, as the memory size of common workstations is constantly increasing, we believe that memory online tracing will become more and more feasible.

3.2 Structure and Design of *MOT*

When started, *MOT* sets up a default filter to capture all packets except those ports that belong to well-known protocols other than HTTP protocol. This filter is set up to receive all packets for all connections of HTTP Protocol traversing the probe point that *MOT* is monitoring. Figure 1 shows the functioning flow chart of *MOT*.

MOT maintains state for each active “flow”. As shown in Figure 1, after kernel filtering and IP reconstruction for those IP fragmentation packets, the first action is to perform an “existing flow look-up”. In this paper, we define a flow as a unidirectional TCP traffic stream with a unique $\langle \text{source-IP-address, source-port, destination-IP-address, destination-port} \rangle$ 4-tuple. Any flow for which the monitor has not seen a packet within the last 64 seconds is considered to be an *expired flow*. Similar flow definition was proposed in [12] and adopted by Kevin Thompson et al in [13]. A flow look-up therefore involves a matching of source and destination addresses and port numbers in the received packet against the equivalent values in the stored flow state. A big hash table is adopted to enhance the performance. If the flow look-up was successful, the retrieved flow state is used, or if it failed, a new flow structure is allocated. If retrieved flow state indicates that it is a useless flow, i.e. it is not of HTTP protocol, the packet is discarded directly. Otherwise, we parse the packet and maintain the state of the flow. This module is discussed in section 3.3. The final function that a tracing system may have to perform is garbage collection. As described above, flow state is normally removed when a TCP FIN or RST message is received for the control connection. However, because of effects such as packet losses or route changes, the probe point might never receive the FIN packet and garbage collection has to be performed to remove stale flow state. In our current implementation, garbage collection is performed when the number of flows exceeds a certain configurable threshold. All expired flows will be eliminated in garbage collection process to free resources.

3.3 Parsing and State Maintaining

Maintaining states and parsing packets are significant procedures in our system. Event-driven programming model is first adopted in our system to ensure scalability and high performance. We introduce a DFA (Deterministic Finite Automata) of a flow to illustrate all possible states and state transition functions. The state transition diagram of DFA is showed in Figure 2. When a new flow is created, the state of the flow is assigned to START. After the flow state changes to END, it is destroyed and removed from memory.

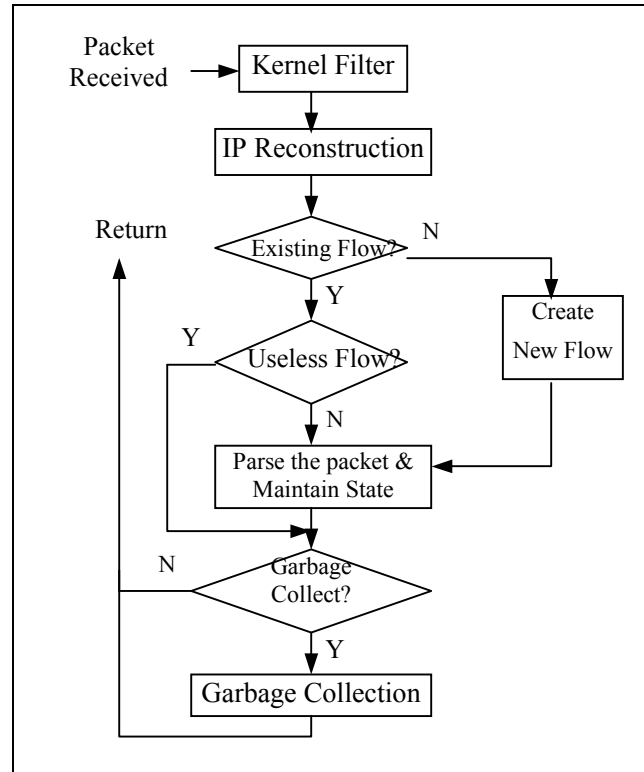


Figure 1. Functioning flow chart of *MOT*

The input symbols in the DFA are combinations of the packet patterns and flow parameters. Here are the explanations in detail:

a. HTTP Request pattern is found in the packet. The pattern is specified in RFC 2616 [11] as “Method SP Request-URI SP HTTP-Version CRLF”. Therefore, this flow must be a HTTP request. The beginning position of the request header is recorded.

b. HTTP Response pattern is found in the packet. The pattern is “HTTP-Version SP Status-Code SP Reason-Phrase CRLF”. So, this flow must be a HTTP response, and we record the beginning position of the response header.

c. Neither HTTP request nor response pattern is found and first 200 bytes have been checked. We expect that it is impossible for this flow to contain Web traffic. So the state changes into USELESS. We do not restrict the pattern of the first HTTP header in a TCP connection to appear at the very beginning of the flow because SOCKS [14] proxy may be used and authentication and setup information of SOCKS may be transferred first. This design makes it possible to drop upcoming packets in the same flow directly.

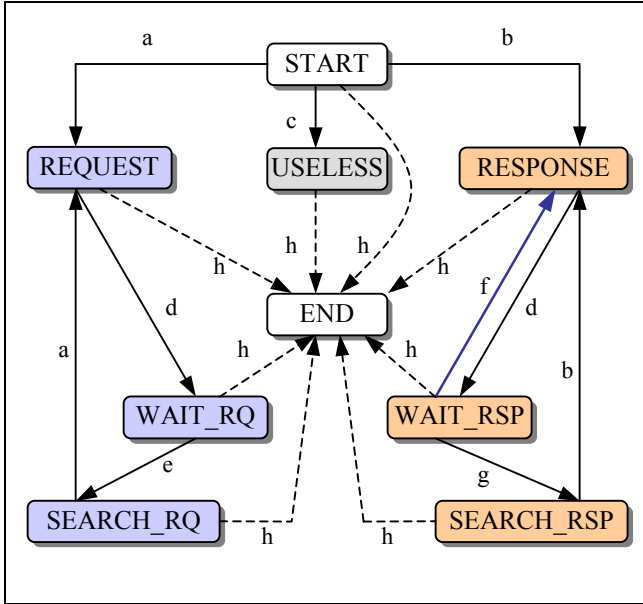


Figure 2. State Transition Diagram of Flows

d. Two consecutive CRLF detected or more than 50,000 bytes of the flow have passed since the beginning of header was found. It can be seen that the terminator of HTTP header (either request or response) is found or the packet containing the terminator is lost because 50KB is enough for most headers. If it is lost, we arbitrarily consider the length of this header to be 50,000 bytes. Otherwise, accurate length can be calculated. Due to the possibility of receiving out-of-sequence packets, data in the header before the terminator may not be received yet. So, we change the state to WAIT_RQ (WAIT for entire ReQuest header) or WAIT_RSP (WAIT for entire ReSPonse header).

e. All the packets before the terminator's position is received or the timeout occurs during the WAIT_RQ state. We dump the entire header to the log-file and eliminate the header from memory. The state transits to SEARCH_RQ to look up next HTTP request.

f. All packets before the terminator's position are received or the timeout occurs during the WAIT_RSP state, and content length is also specified in the header. We dump the entire header to the log-file and eliminate the header from memory. In contrast to *e*, we simply slide the TCP receiving window of the current flow over the content and change the state to RESPONSE again, because we know the length of the content.

g. Similar with *f* but the content-length is not specified in the header. Since we do not know how long the content is, we have to search the beginning of the next header, if exists.

h. RST, FIN packet is received or the current flow is expired. All states except END will transit to the END state if this condition is met.

In a word, the DFA is carefully designed to resolve several problems in reconstruction and parsing, such as multiple requests in persistent connections, packet loss and retransmission and some other difficulties of HTTP reconstruction mentioned in [1].

4. Studies Enabled By MOT

Data collected by MOT can be used for various Web-related analytical studies. In contrast to the data set of search engines that is extremely large and static, MOT provide a specific, dynamic and always up-to-date data set from the users' point of view, instead of the crawlers' point of view. Probing different links can yield data sets for different user groups. Mining MOT logs, we can easily answer the following questions:

- ✓ How does the access rate vary in different time scales?
- ✓ What is the page size distribution?
- ✓ Which web site is the most popular one in any give month?
- ✓ What is the ratio of multimedia content data to all Web data?

MOT also helps researchers to do more complicated work:

- ✓ Web change analysis. Since all HTTP headers are logged, we can extract the "last modified" header field, analyze each page's variation, and summarize the variation law.
- ✓ Link analysis. Combining IP addresses and cookie information, anonymous users' browsing activities can partly be extracted. We can find more general access patterns than before in the abundant information from MOT. We may find hubs and authorities in users' eyes.
- ✓ Geographical distribution of network latency and transfer rate. According to the results of analysis, ICPs (Internet Content Provider) may devise more effective mirror strategies.
- ✓ Protocol performance evaluation. Since MOT provides both TCP level and HTTP level information for every request and response, researchers can do more comprehensive studies, such as benefits of compression and delta encoding [6] and performance of Web caching [5].

Page Size (Bytes)	Concurrent Connections	Bandwidth (Mbps)	Requests Generated	Requests Caught	Accuracy
6519	100	33.1	2,000,000	2,000,000	100.0%
6519	600	43.9	6,000,000	6,000,000	100.0%
6519	2000	50.2	2,000,000	1,992,136	99.61%
6519	4000	55.8	4,000,000	3,613,871	90.34%
18156	100	67.2	2,000,000	2,000,000	100.0%
18156	600	86.5	6,000,000	6,000,000	100.0%
18156	2000	88.6	2,000,000	1,999,204	99.96%
18156	4000	91.1	4,000,000	3,824,178	95.60%

Table 1. Single Node Testing Results in Synthetic Environment

5. Experimental Results

In this section we present preliminary results obtained from our use of *MOT*. In section 5.1 we present results in a synthetic environment, while section 5.2 presents measurement results from a probe point on the CERNET-US network link. The results presented are meant to show the availability and performance of our system rather than conclusive results about the use of HTTP protocol in the Internet.

Our experimental environment is a Pentium III 650E MHz Workstation with two 40GB SCSI disks, 512MB Memory, and two 100Mbps Fast Ethernet adapters. Operating System is Linux, RedHat 6.2 distribution, and the kernel version is upgraded to 2.2.16. All system code is implemented in C.

5.1 Tests in Synthetic Environment

In this case, all traffic that traverses the measured link is synthesized by means of Apache Bench Tool ^[15] to make concurrent requests. This test is necessary to obtain the max performance of *MOT* because in a real environment we only know how many requests and responses our monitoring system record, but how many Web pages users actually view is unknown. We can avoid this situation in synthetic environment. Results in synthetic environment are shown in table 1. When the number of concurrent HTTP connections is less than 2000, the system shows perfect accuracy. Along with bit-rate and number of concurrent connections increasing, packet loss happens and not all Web requests and response are caught. Since we do not save and parse HTTP contents and only care for headers, the ratio of header size to content size in

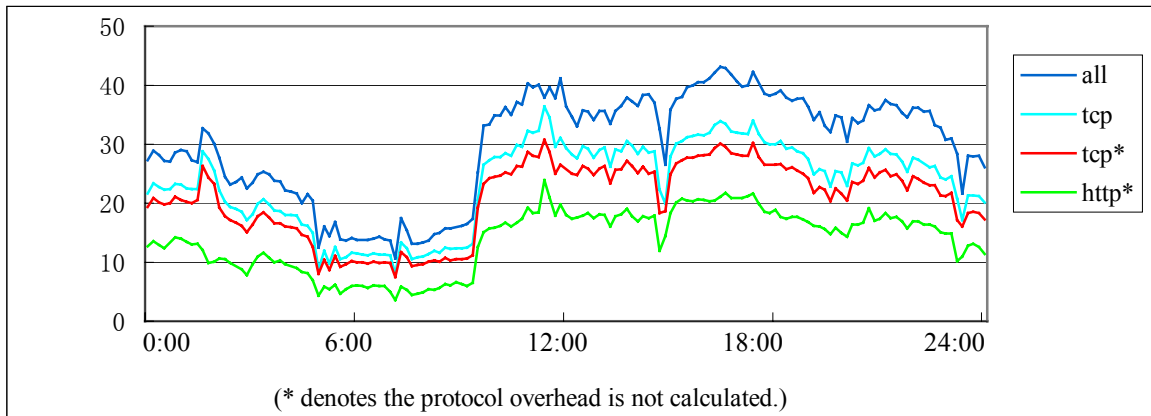


Figure 3. Processing Bit-Rate and Traffic Composition Over 24-hour Period

entire traffic is also related to max performance. This is why performance increases when page size increases. Since all synthetic loads are Web traffic, we believe that our system can achieve better performance in real environment.

5.2 Tests in Real Environment

We have been using our system to monitor Web traffic on the CERNET-US link. Figure 3 depicts processing bit-rate and traffic composition for 24 hours. We can see that the network is very quiet around 6 A.M. and is quite busy around 4 P.M.

Figure 4 illustrates the byte volume during 412,070 seconds. It shows that the Web traffic without overhead of TCP/IP protocol headers occupies about 44.0% of entire network traffic.

6. Conclusion and Future Work

We have presented the design and implementation of a new tool for monitoring Web traffic on high-speed links. *MOT* is based on libpcap library, expanded from the principles of BLT and incorporates several novel features that enable it feasible to monitor Web traffic on a high-speed links. First, feasibility of memory online tracing is analyzed and proved. Second, the process of flow reconstruction, parsing and logging is incorporated. Non-Web traffic can be identified immediately so that the useless information is discarded in early stages. Third, *MOT* maintains per-flow state to extract and record related information. Fourth, it extracts content length information from response headers and split HTTP contents from headers to save the memory and parsing time. Moreover, the situation of SOCKS proxy and IP fragmentation is considered for accuracy of reconstruction.

In terms of ongoing and future work, we plan to add data distribution in our system. Admittedly, the computational capacity and throughput of a single workstation is limited. We are experimenting with using a router as a dispatcher to distribute entire traffic to a cluster of workstations. Powered by cluster computing technologies, the system will gain extreme high throughput and availability. We are also about to encrypt IP addresses and cookies in log-files to achieve greater privacy. Meanwhile, we continue to use *MOT* to monitor Web traffic on public Internet and plan to perform a thorough analysis of this traffic's growth and characteristics.

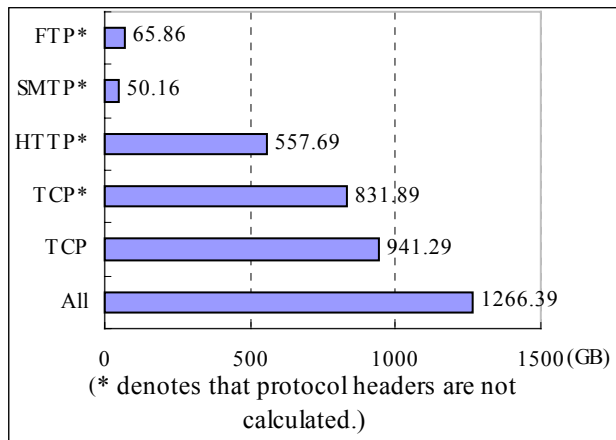


Figure 4. Byte Volume Distribution of Common Protocols

7. Acknowledgments

We wish to express our sincere appreciation to Shuang Gao, Peng Bi, Wei Liu and Martha Dahlen for their valuable suggestions of this paper.

References

- [1] A. Feldmann, *BLT: Bi-Layer Tracing of HTTP and TCP/IP*, 9th International World Wide Web Conference 2000.
- [2] R. Wooster, S. Williams, and P. Brooks. *HTTPDUMP: a network HTTP packet snooper*. 1996.
- [3] Matt Welsh and David Culler, *Virtualization Considered Harmful: OS Design Directions for Well-Conditioned Services*, Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS VIII), Schloss Elmau, Germany, May, 2001.
- [4] LBNL's Network Research Group, *TCPDump - dump traffic on a network*. UNIX man page.
- [5] A. Feldmann, R. Caceres, F. Douglass, G. Glass, and M. Rabinovich. *Performance of Web proxy caching in heterogeneous bandwidth environments*. In Proc. IEEE INFOCOM, 1999
- [6] J.C. Mogul, F. Douglass, A. Feldmann, and B. Krishnamurthy. *Potential benefits of delta encoding and data compression for HTTP*. In Proc. ACM SIGCOMM, pages 181-194, September 1997.
- [7] Balakrishnan, V.N. Padmanabhan, S. Seshan, M. Stemm, and R.H. Katz. *TCP behavior of a busy Internet server: Analysis and improvements*. In Proc. IEEE INFOCOM, April 1998

- [8] S.D. Gribble and E.A. Brewer. *System design issues for Internet middleware services: Deductions from a large client trace*. In Proc. USENIX Symp. on Internet Technologies and Systems, December 1997
- [9] Malan, G. R., and Jahanian, F. An Extensible Probe Architecture for Network Protocol Performance Measurement. Proc. of ACM SIGCOMM'98, August 1998.
- [10] S. McCanne and Van Jacobson, *The BSD Packet Filter: A New Architecture for User-level Packet Capture*. 1993 Winter USENIX conference, January 25-29,1993, San Diego, CA.
- [11] R. Fielding, J. Gettys and J. Mogul, *RFC2616: Hypertext Transfer Protocol – HTTP/1.1*, Network Working Group, 1999
- [12] K. Claffy, *Internet Workload Characterization*, Ph.D. Dissertation, University of California, San Diego, CA, June 1994.
- [13] K. Thompson, G. J. Miller, and R. Wilder, *Wide-Area Internet Traffic Patterns and Characteristics*, trans of IEEE Networking, 1997
- [14] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas & L. Jones, *RFC 1928: SOCKS Protocol Version 5*. April 1996
- [15] Apache Software Foundation, *Apache Benchmark Tool*. <http://httpd.apache.org/docs/programs/ab.html>