Using Empirical Distributions to Characterize Web Client Traffic and to Generate Synthetic Traffic

Henrik Abrahamsson Bengt Ahlgren

{henrik,bengta}@sics.se
Swedish Institute of Computer Science
Box 1263, SE-164 29 Kista, Sweden

Abstract— **We model a web client using empirical probability distributions for user clicks and transferred data sizes. By using a heuristic threshold value to distinguish user clicks in a packet trace we get a simple method for analyzing large packet traces in order to get information about user OFF times and amount of data transferred due to a user click. We derive the empirical probability distributions from the analysis of the packet trace. The heuristic is not perfect, but we believe it is good enough to produce a useful web client model.**

We use the empirical model to implement a web client traffic generator. The characteristics of the generated traffic is very close to the original packet trace, including self-similar properties.

I. Introduction

Measurements on the Internet backbone [4], [16] show that HTTP comprises approximately 70-75 % of the total traffic. To understand the behavior of the aggregated traffic, it is therefore important to understand how the HTTP traffic behaves. We have the goal of implementing a traffic generator which can be used to generate realistic best-effort background traffic in lab networks. When introducing multiple traffic classes, as in diffserv, the background best-effort traffic will to some extent disturb higher priority traffic, such as voice, depending on queue management and scheduling algorithms. To be able to make realistic lab experiments with traffic classes, we need a web traffic generator.

We start with a packet trace of web traffic. But we do not want to model the number, size and inter-arrival times of TCP packets since these quantities are governed by the TCP flow control and congestion control algorithms. The timing of a connection's packets as recorded in a trace reflects the conditions in the network at the time the connection occurred. Due to this adaptation to the network done by TCP, a trace of a connection's packets cannot easily be reused in another context, because the connection would not have behaved the same way in the new context [12]. Instead we use the packet trace to characterize the behavior at a higher level rather than at the packet level.

We base our web client model on user clicks and statistics of the amount of data transferred as a result of each click. In the packet trace we detect when a user clicks on a link to get the next web page and from that we deduce how much data that was transferred in response from the web server, as well as the time between the end of the transfer to the next click. This time of silence preceding a click is here called *user OFF time.* The packet trace analysis uses heuristics to deduce user clicks without the need to parse HTTP requests. This makes it possible to analyze very large traces and traces which only has the packet headers recorded.

We use the empirical model to implement a web client traffic generator. We show that the resulting aggregated traffic from many sources have the same properties, including self-similarity, as the traffic in the original trace.

The contributions of this paper include heuristics for detecting user clicks in a packet trace, a simple empirical web client model and a realistic web client traffic generator.

The remainder of the paper is organized as follows. Section II gives a brief introduction to the HTTP protocols, a description of the packet trace and the method used to extract information from the trace. The resulting empirical distributions are presented in Section III and synthetic traffic generation using these distributions is described in Section IV. The paper is ended with related work and conclusions.

II. Analyzing web traffic

A. The HTTP protocols

The application-level protocol HTTP exists and is used in more than one version. There is yet no formal standard that everybody follows.

HTTP/1.0 [13] is a simple protocol. The web browser establishes a TCP connection to the web server, issues a request, and reads back the server's response. The server indicates the end of its response by closing the connection. When a browser using HTTP/1.0 fetches web pages it sets up a new TCP connection for each requested document. Web pages often have many embedded images, which each is retrieved via a separate HTTP request. Thus, to retrieve a web page with five images, six different TCP connections are required. The first TCP connection transfers an HTTP GET request to receive the HTML document that refers to the five images. A very simple browser would, when the HTML document is received, open one new TCP connection to get the first image. After sending the response the connection is closed by the server and another connection is opened to get the second image and so on. The use of a new TCP connection for each image serializes the display of the entire page. Netscape introduced the use of parallel TCP connections to compensate for this serialization. When the HTML document is received normally four TCP connections are opened in parallel for the first four images which decreases the transaction time for the user.

HTTP/1.1, as it is described in RFC 2616 [14], differs from HTTP/1.0 in numerous ways, both large and small. Of most interest here is the network connection management. The problem in HTTP/1.0 that a new TCP connection is required for each document is resolved by the use of persistent connections and the pipelining of requests on a persistent connection. Persistent connections means that the client and server keep a TCP connection open instead of the server closing the connection after sending the response. The same connection can be used to fetch several images and can be kept open even if the user clicks to another web page as long as the page is located on the same server. Pipelining means that a client can send an arbitrarily large number of requests over a TCP connection before receiving any of the responses. HTTP/1.0, in its documented form, made no provision for persistent connections but some implementations use a Keep-Alive header to request that a connection persist.

B. The packet trace

To get information about user OFF times and the amount of data transferred due to a user click, web traffic was captured using *tcpdump* [8]. Figure 1 shows the

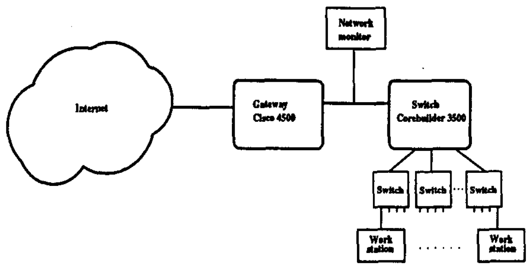

Fig. 1. The network at SICS

network at SICS. The machine running *tcpdump* (called network monitor in the figure) was listening to the 100 Mb/s line connecting all workstations at SICS with the gateway. This was used to capture conversations between machines at SICS and the outside Internet world. Only traffic where users at SICS were clients was captured, not the HTTP traffic that arise from people outside visiting the SICS web pages. The packet trace was taken between 18:50:04 000222 and 11:17:51 000301 and includes 8317992 packets transferred between TCP port 80 on web servers and 181 different clients at SICS. The amount of HTTP traffic varies of course during the day and during the week (Fig. 2) depending on how many people are using the network. But also when the traffic is studied on lower time scales from hours down to milliseconds there is a lot of variation in the number of bytes and packets sent.

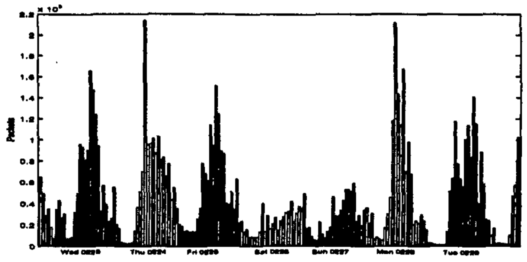
Fig. 2. Packets per hour between 19:00 000222 and 11:00 000301

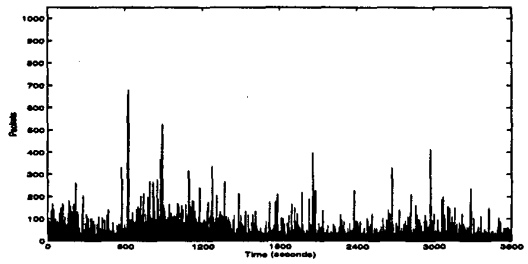

Fig. 3. Packets per second 11:00-12:00 Mon 000228

Figure 3 shows the traffic during one of the busiest hours where at most 52 client were active. A few years ago Leland *et al.* [10] showed that LAN traffic is bursty on many time scales in a way that can be well described using *self-similar processes* and later Crovella *et al.* [6] showed that this also holds for web traffic. The degree of self-similarity is expressed using the so called *Hurst* parameter. This parameter can take any value between 0.5 and 1 and the higher the value the higher the degree of self-similarity. For Poisson traffic the value is $H = 0.5$. An often used heuristic graphical method to estimate the Hurst parameter is the Variance-Time plot which relies on the fact that a self-similar process has slowly decaying variances. For a detailed discussion of self-similarity and the methods used for estimating the Hurst parameter see Leland *et al.* [10] and Crovella and Bestavros [6]. Figure 4 shows an estimate

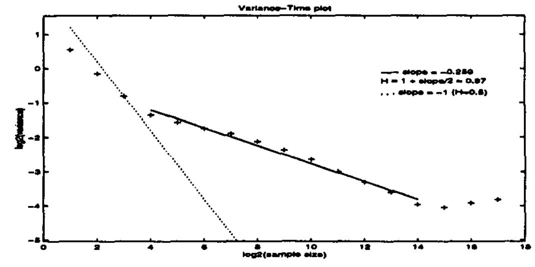

Fig. 4. Estimate of the Hurst parameter

of the Hurst parameter for the hour 11:00-12:00 000228 using the Variance-Time plot. The value is 0.87 so the traffic during that hour can be said to be self-similar meaning bursty on many time scales. Self-similarity expressed using the Hurst parameter seems to be a good way of describing the behavior of real web traffic and it would be good if the generated traffic also have the same properties.

C. Detecting user clicks

When a user clicks on a link to get the next web page, the browser sends a HTTP request to the server. We want to detect these requests and separate them from requests for parts of a web page. To separate a request due to a user click from other requests the time of silence preceding it is investigated. A request is assumed to be due to a user click if it is preceded by enough time of silence, an interval here called T_{click}, where no HTTP traffic is sent to or from this client. The packet trace doesn't contain application level HTTP requests and responses, but only lower level TCP/IP packet headers. Different users use different browsers with different number of parallel TCP connections and where some use persistent connections and some don't. This means that the start and end of connections cannot be used to determine if a user have clicked on a link to fetch a new web page. Instead only the time between the last HTTP response (or request) and a new request is considered, irrespective of which TCP connection the client uses for the transfer. Since the HTTP client sends almost only requests, we assume that every TCP packet from a client - carrying some payload data (not pure acknowledgment or control packet) - is transferring a HTTP request. If the transfer of a TCP packet that carries a request is preceded by a period of T_{click} seconds where no data is transferred to or from this client then we assume that this packet represents a user click. The problem is to determine the value of T_{click}. The value should be large enough, so that requests for parts of the same web page is not counted as user clicks, and small enough to separate different user clicks.

Similar problems have been addressed by Mah [11] and by Crovella and Bestavros [6]. When investigating packet traces in order to determine the number of files per web page, Mah uses the threshold value 1 second to separate connections that belongs to different web pages. The main reason for the choice of this value was that users will generally take longer than one second to react to the display of a new page before they order a new document retrieval. When investigating causes of self-similarity in WWW traffic, Crovella analyses OFF times and concludes that times in the range of 1 ms to 1 second is likely to be strongly determined by machine processing and display time for data items that are retrieved, not due to users examining data.

For each request we calculated the time of silence preceding it. From these times the requests were sorted and counted. Figure 5 shows the result with time of silence in 0.2 second bins ranging from 0 to 2.0 seconds. We chose the value $T_{click} = 1$ second, even though the values in Figure 5 might suggest that an even smaller value would have been reasonable.

In order to validate that the method and threshold value described really gives reasonable results we used a proxy X-server that logged time-stamps on the mouse button-up events when using Netscape. This was used to log the actual clicks made and at the same time tcpdump was used to capture all web traffic to and from the client. The packet trace was analyzed using the threshold value $T_{click} = 1$ second in order to detect user clicks. The time-stamps of the detected clicks were compared to the time-stamps in the X-server log. The result is shown in Table I. If a click detected in the trace has a time-stamp equal to (or very close to) a time-stamp in the log file it is called a *hit*. If a detected click in the trace does not correspond to a real click it is called a *false* click and if a click in the log file is not detected in the packet trace it is said to be *missed*.

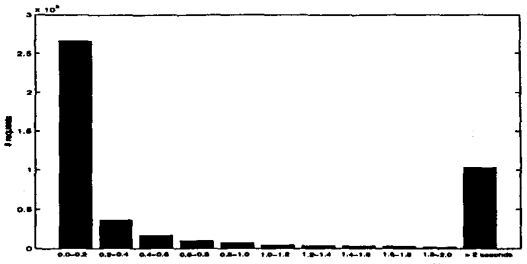

Fig. 5. Time of silence preceding HTTP requests

# detected clicks in trace	629
# real clicks in log	532
Hits	519 (97%)
Missed	13 (3%)
False	110 (18%)

TABLE I
EXAMINATION OF THE CLICKS DETECTED

Approximately 97% of the real clicks were detected and 82% of all detected clicks were correct. It should be emphasized that since only quite a small number of clicks have been investigated and the timing of requests depends on the user, the machine used, what pages are visited and so on the results in Table I should only be seen as coarse estimates. A closer examination of the packet trace shows that twelve of the false clicks were due to retransmissions of requests but the main reason for the many false clicks is that requests for part of a web page is sometimes preceded by more than one second of silence and thus detected as user clicks. In general, clicks are missed because the client quickly clicks to navigate to another web page before the transfer of the previous one was completed. In that case there is no one-second interval of silence preceding the request so the click is not detected. So, not all but too many clicks are detected. A larger value of T_{click} would give less false but more missed clicks. The method used to detect clicks is not perfect but from the results in Table I it seems to be good enough to be useful.

D. User sessions

Since a client that begins with an hour of silence or takes a two week vacation is not very useful in a traffic generator we also need to break up the traffic into user sessions. The notion of a session is supposed to cover the time interval when a user is active and uses the browser to fetch and read web pages. This is vague and hard to define, especially in

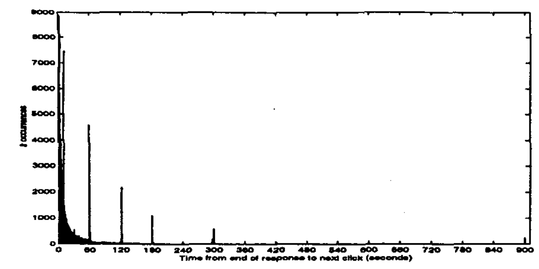

Fig. 6. Histogram of time between end of response and next click

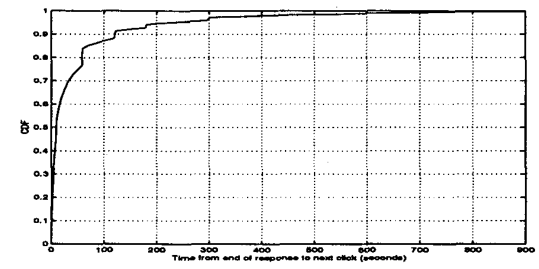

Fig. 7. Empirical CDF of time between end of response and next click

terms of packets sent and received. We define a session to be an interval in which a user creates WWW traffic without being silent for more than a certain time. That is, a session starts when the first web page is fetched (the first request is made) and ends when the last page is received (but not yet read). If no request or response is sent for a certain time, a threshold value here called $T_{session}$, then the next request is the start of a new session. We used the value $T_{session}$ = 15 minutes.

E. Data analysis

Awk was used to extract the needed information from the tcpdump file. The extracted data was later investigated further using Matlab. Only information about packets carrying payload data was extracted. The input to Matlab was a matrix where each such packet was represented with a time-stamp in microseconds, a unique client id, direction (client request or server response), and the packet size without headers. A Matlab program was written which for each client went through the times between requests and responses and used T_{click} to detect user clicks and determine user OFF-times and the amount of data transferred as response to a user click.

III. Empirical distributions

In this section, we use the packet trace and the heuristics from the previous section to develop the two empirical distributions needed to model web client traffic.

A. OFF times

In Figures 6 and 7 a histogram and the cumulative distribution function (CDF) of the time from the end of the response to the next user click are shown. There were a total of 90621 user OFF-times in the data set. The minimum time was 1.000003 seconds, just above the T_{click} threshold of one second. The maximum is determined by the value of $T_{session}$=15 minutes. The median time was 9.8 seconds and the mean 49.39 seconds with a standard deviation of 109.7 seconds. The coefficient of variation was 2.2.

There are several peaks in the histogram, most noticeable at 10, 60, 120, 180 and 300 seconds. A closer examination of the packet trace shows that for each of these peaks there is a single client that causes most of them. For instance, the peak at 10 seconds originates from a client that for 12 hours repeatedly sends requests to check if the web page has been modified and the peak at 300 is due to somebody updating their stock-exchange rates every five minutes. It is not obvious whether these periodic OFF-times should be included or characterized as anomalies and thus be removed from the data set. The requests are not really user clicks but the OFF times are a part of the real traffic so we let them contribute to the empirical distribution in Figure 7.

B. Amount of data transferred due to a single user click

The amount of data transferred from servers as response to a single user click varies a lot. On one occasion 31940163 bytes were transferred and at other times no data at all was received by the client. In Figure 8 only the part of the CDF that covers values below 250000 bytes is shown. The median was 7145 bytes and the mean was 39142 bytes

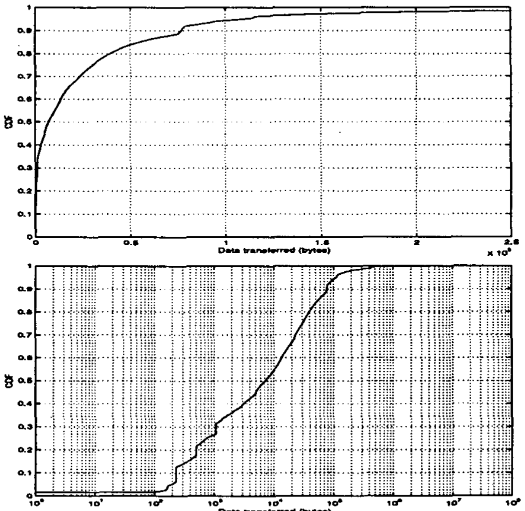

Fig. 8. Empirical CDF of the amount of data transferred as response to a user click (linear x-axis at top and logarithmic at bottom).

with a standard deviation of 317753 bytes. The coefficient of variation was 8.1.

IV. Generating traffic

By using the distributions in Section III, traffic can be generated that resembles a number of users surfing the web

— reading web pages (OFF times) and clicking on links to get the next one (data transfer).

A. The traffic generator

The traffic generator was implemented in the C programming language and has a client and a server part. Values from the empirical distribution for OFF-times (Fig. 7) and data transferred (Fig. 8) was pre-computed and written to a file using the inverse transformation method described, for instance, by Jain [9]. The client reads from the file the OFF time and the amount of data that should be transferred and sends the latter as a request to the server. The server side just accepts requests and replies by sending the demanded amount of data. The number of different clients is given as input to the program and each client is represented by a process that repeatedly goes through the loop of requesting and receiving data according to the distribution of data transferred due to a single user click and then goes to sleep for a time described by the distribution of OFF times until the next request is made.

B. Evaluation

Traffic resembling 60 clients was generated on a 10 Mb/s link between two machines. In order to validate that the OFF times and the amount of data transferred follows the distributions the generated traffic was captured using tcpdump and analyzed in the same way as the original packet trace. Figures 9 and 10 show the distributions for the generated traffic and a comparison with the originals (dotted line). In the traffic generator no request is sent for a zero bytes response so the CDF for the amount of data transferred lies somewhat lower than the original.

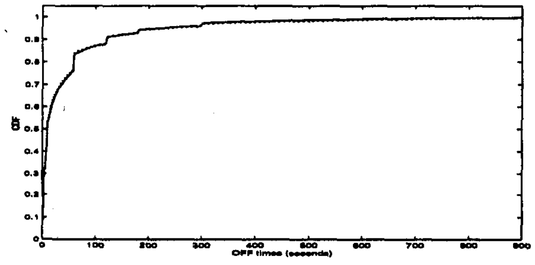

Fig. 9. OFF times

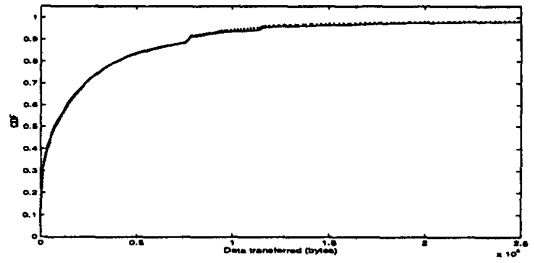

Fig. 10. Data transfered

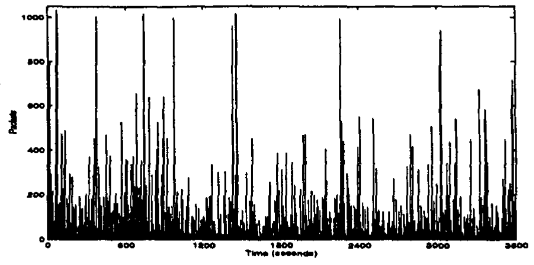

Fig. 11. Packets per second

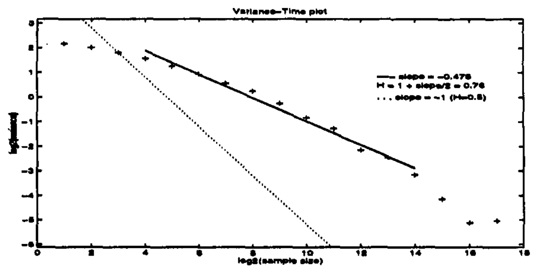

Fig. 12. Estimate of the Hurst parameter

There is a lot of variation in the number of packets that are transferred in one second (Fig. 11) and a Hurst parameter value of 0.76 (Fig. 12) indicates that the generated traffic, like real web traffic, is bursty on many time-scales.

V. Related work

Two approaches can be used to generate network traffic that imitates real web traffic. The first is simply to replay packet traces of real web traffic. But, because of TCP's flow and congestion control the timing of packets in a trace reflects the condition in the network when the trace was taken and this timing would not be the same in another context. The alternative is to gather information about, and mathematically describe, those aspects of the web traffic that one believe is most important and from this model generate traffic. This approach was used by Barford and Crovella [2].

To get information about web traffic three different approaches have been widely used: server logs, client logs and packet traces. Server logs cannot easily be used to describe the client side since a client usually accesses many different web servers. To capture the client accesses between multiple servers, client logs can be used. This approach requires that browsers can log their requests, that the source code for the browser is available so that logging can be added, or that some other way to log the clients behavior is available. Catledge and Pitkow [3], Cunha *et al.* [5] and Crovella and Bestavros [6] use instrumented versions of the Mosaic web browser. Barford *et al.* [1] use HTTP proxies to track all documents referenced by unmodified Netscape Navigator clients. The third approach of gathering data, and the method used here, is to analyze packet traces taken from a subnet carrying HTTP traffic. This method was used by

Stevens [15] to analyze the traffic arriving at a server, and by Mah [11] to model the client side of the HTTP traffic. A further step is taken by Anja Feldman [7] when extracting full HTTP level as well as TCP level traces via packet monitoring.

VI. Conclusions and future work

We have presented an empirical model for web client traffic. The model is based on user click behavior combined with statistics of the amount of data transferred per click. The user clicks, or actually the silence times before a click, and the amount of data are modeled using cumulative distribution functions. By using a heuristic threshold value to distinguish user clicks in a packet trace, we get a simple method for analyzing large packet traces without the need for parsing HTTP requests. The result of the analysis are data defining the two distribution functions. The simplicity of the heuristic packet trace analysis may have a price in accuracy. A verification, however, shows that the heuristics correctly detect 82 % of the actual user clicks from the packet trace. We believe that this is sufficiently accurate to produce a good empirical model.

We have implemented a web client traffic generator which takes the cumulative distribution functions as input. We have shown that the generated synthetic traffic have the same characteristics as the original packet trace by applying the same analysis to a trace of the generated traffic. We have also verified that the aggregated generated traffic from many clients has self-similar properties just like the original trace.

Future work include analyzing packet traces from more networks and comparing the resulting distribution functions. We plan to use the traffic generator to generate best-effort background traffic in lab experiments with voice-over-IP and multiple traffic classes. We also plan to release the source code to the analysis software and the traffic generator shortly.

Acknowledgments

The authors would like to thank Assar Westerlund for providing the logging proxy X-server and Peter Ehlin and Björn Grönvall for their help when taking the packet traces.

References

[1] P. Barford, A. Bestavros, A. Bradley and M. Crovella, "Changes in web client access patterns: characteristics and caching implications," in *World Wide Web*, Special issue on characterization and performance evaluation, Vol. 2, pp. 15-28, 1999.

[2] P. Barford, M. Crovella, "Generating representative Web workloads for network and server evaluation," in *Proceedings of Performance '98/ACM SIGMETRICS '98*, pp. 151-160, Madison WI, 1998.

[3] L. D. Catledge and J. E. Pitkow, "Characterizing browsing strategies in the World Wide Web," in *Proceedings of the third International World Wide Web Conference*, Darmstadt, Germany, April 1995.

[4] K. Claffy, G. Miller and K. Thompson, "The nature of the beast: recent traffic measurements from an Internet backbone." http://www.caida.org/Papers/Inet98.

[5] C. R. Cunha, A. Bestavros, and M. E. Crovella, "Characteristics of WWW client-based traces," Technical Report BU-CS-95-010, Computer Science Department, Boston University, July 1995.

[6] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: evidence and possible causes," in *IEEE/ACM Transaction on Networking*, Vol. 5, no. 6, pp.835-846,1997.

[7] A. Feldmann, "BLT: Bi-Layer Tracing of HTTP and TCP/IP," to appear at 9th International World Wide Web Conference, Amsterdam, May 2000.

[8] V. Jacobson, C. Leres and S. McCanne, tcpdump software. This software is available at ftp://ftp.ee.lbl.gov/tcpdump.tar.Z .

[9] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, New York, 1991.

[10] W. E. Leland, M. S. Taqqu, W. Willinger and D. V. Wilson, "On the self-similar nature of Ethernet traffic (Extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1-15, 1994.

[11] B. A. Mah, "An empirical Model of HTTP network traffic," in *INFOCOM '97 Conference Proceedings*, pp. 592-600, Kobe, Japan april 7-11, 1997.

[12] V. Paxson and S. Floyd, "Why we don't know how to simulate the Internet," in *Proceedings of the 1997 Winter Simulation Conference*, Atlanta, GA, 1997.

[13] RFC 1945 *Hypertext Transfer Protocol - HTTP/1.0*, available at: http://www.ietf.org/rfc/rfc1945.txt

[14] RFC 2616 *Hypertext Transfer Protocol - HTTP/1.1*, available at: http://www.ietf.org/rfc/rfc2616.txt

[15] W. R. Stevens, *TCP/IP Illustrated, Vol. 3: TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*, Addison-Wesley, Reading, 1996.

[16] K. Thompson, G. J. Miller and R. Wilder, "Wide-area traffic patterns and characteristics (Extended version)," in *IEEE Network*, vol. 11, no. 6, pp. 10-23, 1997.