
ELSEVIER

Performance Evaluation 46 (2001) 77–100

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

Traffic model and performance evaluation of Web servers

Zhen Liu*, Nicolas Niclausse, César Jalpa-Villanueva[1]

INRIA, Centre Sophia Antipolis, 2004 Route des Lucioles, BP 93, 06902 Sophia Antipolis, France

Abstract

In this paper we present a new model of Web traffic and its applications in the performance evaluation of Web servers. We consider typical user hypertext navigation behavior within a Web server. We propose a traffic model at the session level, formulated as a stochastic marked point process, which describes when users arrive and how they browse the server. We developed a Web server benchmark: WAGON (Web trAffic GeneratOr and beNchmark), and we validated the traffic model by comparing various characteristics of the synthetic traffic generated by WAGON against measurements.

We then report benchmark and analysis results on the Apache server, the currently most used Web server software. We analyze the impact of the traffic parameters on the HTTP request arrival process and the packet arrival process. We also show that the aggregate traffic is self-similar in most cases, and that, more importantly, the Hurst parameter is increasing in the traffic intensity. We further provide performance comparison results between HTTP1.0 and HTTP1.1 and show that HTTP1.1 could be much worse for users as well as for servers if some control parameters of the server and of browsers are incorrectly set. Indeed, when the server load is relatively high, the page response time under HTTP1.1 increases exponentially fast with the number of parallel persistent connections and with the timeout value used in Apache for persistent connections. We investigate the impact of user network conditions on the server performance. We also propose a queueing model to analyze the workload of persistent connections on the Apache server, and we establish optimal solution of the timeout parameter for the minimization of workload.

Our analyses indicate that it is usually beneficial for both Web servers and Web clients to use HTTP1.1 instead of HTTP1.0. When HTTP1.1 is used, it should be used with pipeline. In terms of the management of persistent connections, it is useful for browsers to implement Early Close policy which combines the advantages of both HTTP1.0 and HTTP1.1. Browsers should in general, except for users with low bandwidth network connections (such as modem), avoid establishing multiple parallel persistent connections from one browser window to the same Web server. On the server side, servers should set small timeout values for persistent connections if fixed timeout control mechanism is used (as in Apache) or if dynamic timeout control mechanism is used and the measured workload is high.

Keywords: Web server performance; Web traffic modeling; Apache server; Server model; HTTP1.0; HTTP1.1; Persistent connection; User perceived quality of service

* Corresponding author. Current address: IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA. Tel.: +1-914-784-6503; fax: +1-914-784-6040.

E-mail addresses: zhen@watson.ibm.com (Z. Liu), nic@noos.fr (N. Niclausse), cjv@xanum.uam.mx (C. Jalpa-Villanueva).

[1] Universidad Autonoma Metropolitana – Iztapalapa, Av Michoacan y La Purisima, Col. Vicentina, 09340 Mexico D.F., Mexico.

0166-5316/01/$ – see front matter
PII: S0166-5316(01)00046-3

1. Introduction

The exponential growth of the number of servers and of the number of users causes performance problems of access to Web objects, due to the saturation of Web servers and of the Internet. One of the main preoccupations of Web server administrators is to propose a fast and reliable service to satisfy their users. The tremendous success of the Web makes this task difficult to accomplish. Indeed, not only is it necessary to provide an efficient service for a given time instant, but also it is required to anticipate traffic growth in order to maintain the quality of service (QoS) for short or medium term. It is thus important to understand the statistical properties of the Web traffic and to develop appropriate traffic models for the performance evaluation of Web servers and Web applications.

Many studies have been focused on Web traffic characterizations: request and document size distributions, transfer times, request arrivals, number of visits, etc., see [1,2,6,7,10,11,19,25,28,29]. In particular, heavy tail distributions for document sizes, popularity, and requests and the self-similar nature of Web traffic are observed [1,6,7], and both light-tailed and heavy-tailed behaviors of the traffic patterns exist.

Another line of research in the field has been the performance analysis of dominant transport protocol HTTP used in the Web. At the time HTTP1.1 was proposed, a comparison study between HTTP1.0 and HTTP1.1 was carried out [22]. The effect of persistent connections, pipelining, document compression, bandwidth, and latency, on the performance was investigated using a single Web page containing 42 embedded images. A work to characterize Web response time is presented in [14]. Four proxy logs files are used, they are replayed using a URL re-player multi-process program. The workload is replayed by reading a log file of URLs, sending HTTP requests, and timing the transfer. Ten experiments were done to investigate the effects of proxy caching, network bandwidth, traffic load, and persistent connections. The periodicity of the response times was also studied. More recently, two other studies on the performance comparisons between HTTP1.0 and HTTP1.1 were reported. One focuses on the effect of the CPU and memory capacities [4]. The other reports preliminary studies on the performance difference of the two versions of HTTP under different network connection conditions (bandwidth and propagation delay) [15].

There exist several Web benchmarks. WebSTONE [30] is a distributed, multi-process benchmark. The WebSTONE generates HTTP traffic that allows to stress an HTTP server. The load is generated by requesting pages and files from the server as fast as the server can send them. The existing servers currently in use on the Web are represented by four different files (small, large and mixed pages). SPECweb [27] works in the same way as WebSTONE (both are based on LADDIS benchmark for file systems) with the difference that the workload is composed of a mixture of four file classes, where the weight of each class is obtained from the analysis of logs. Benchmark hbench:Web [17] follows the same paradigm as WebSTONE and SPECweb. hbench:Web derives its workload by analyzing existing Web server logs to determine the site's page set, a collection of user profiles, and the inter-arrival time between users. In particular, the workload is generated according to the empirical distributions of user inter-arrival times. httperf [20] permits generating HTTP workload in several ways: as HTTP calls generated deterministically and at a fixed rate, as sessions consisting of a number of HTTP call bursts spaced by a fixed user think time (also sessions are generated deterministically and at a fixed rate), as URL requests generated over and over again, and as URL request sessions generated at a given rate.

SURGE [3] is a benchmark tool that tries to imitate a stream of HTTP requests originating from a fixed population of Web users. A user is modeled by an ON/OFF process (user equivalent) who during the ON period makes requests for Web files and during the OFF period lies idle. Within an ON period there are active OFF time periods corresponding to the time between transfer of components of a page. Inactive OFF

time periods correspond to the user think time. The workload is generated using an analytic approach to capture properties observed in real Web workloads concerning file sizes (what is stored in the file system), request sizes (what is transferred over the network from the server), file popularity, and temporal locality. Distributional models for the file size, the active OFF times, and the embedded references, were developed using a client trace data set. The most important advantage of SURGE compared to the other benchmarks is that SURGE allows to generate much more realistic traffic. Using this new model the authors of [4] analyzed the performance of Web servers (Apache and IIS) and studied the impact that server hardware (CPU, memory, etc.) has on the performance. They also compared performance of HTTP1.0 and HTTP1.1 and concluded the advantage of HTTP1.1.

In our work we propose a new model of Web traffic and use this model in the performance evaluation of Web servers. We consider typical user hypertext navigation behavior within a Web server. We propose a traffic model at the session level, formulated as a stochastic marked point process, which describes when users arrive and how they browse the server. We provide results of statistical analyses and goodness-of-fit of various simple parametric distributions.

We have developed a benchmark tool of Web servers: WAGON (Web trAffic GeneratOr and beNchmark). It comprises a generator of Web traffic, a robot which sends and analyzes requests and a monitoring tool. It can generate different types of traffic requests, which, in turn, can be sent out to the server from different machines with different (simulated) delays and bandwidths. Using this tool we validated the traffic model by comparing various characteristics of the synthetic traffic generated by WAGON against measurements. More details on this model and WAGON are available in [12,21].

The basic advantage of our model compared to WebSTONE-like benchmark is its closeness to the real traffic. Compared to SURGE, one of the advantages of this new traffic model is its simplicity and its ease of parameterization. Indeed, it is at a slightly higher level (session level) and has a direct correspondence with user's hypertext navigation behavior. The random variables used in the model, such as the number of clicks in a session and inter-click idle times (or user think time), are simple to interpret and are relatively easy to measure and their analyses have independent interest. The statistical analyses of the traffic parameters within this framework also reveal some interesting properties of the Web traffic. It turns out that the session arrivals are well described by a renewal process, and in many cases, they are simply a Poisson process. This nice property allows for a simple way to model arrivals and to parameterize the traffic intensity. This is in contrast with the HTTP request arrivals which usually form a long-range dependent process, and is thus more difficult to analyze. Another advantage is that this new traffic model allows the benchmarks to factorize users' navigation behavior from the implementation issues of the servers, browsers and protocols. This is in contrast with other models at the HTTP request level which are dependent on both the server (Apache, IIS, etc.) and the navigator (Netscape, IE, etc.) and the ways they implement the protocol (HTTP1.0 or 1.1, parallel connections, persistent connections, pipeline, etc.). Last, our model intrinsically describes the nature of dynamic changing user population of Web server.

With this traffic model and our benchmark WAGON, we carried out a series of experiments, most of which are difficult to accomplish with other existing benchmarks. We analyze the impact of the traffic parameters on the (Apache) server performance and the user perceived QoS. We show that the aggregate traffic is self-similar in most cases, and that, more importantly, the Hurst parameter is increasing in the traffic intensity. We further provide performance comparison results between HTTP1.0 and HTTP1.1 and show that HTTP1.1 could be much worse for users as well as for servers if some control parameters of the server and of browsers are incorrectly set. We also investigate the impact of user network conditions on the server performance. Indeed, when the server load is relatively high, the page response time under

HTTP1.1 increases exponentially fast with the number of parallel persistent connections and with the timeout value used in Apache for persistent connections.

The paper is organized as follows. In Section 2 we describe the traffic model and we report the statistical analysis results. In Section 3 we present the benchmark tool WAGON and the setup of the experimentation environment. We also present the model validation results as well as the results pertaining to the impact of the distributions of the random variables of the traffic model. We discuss how the performance characteristics depend on these statistical laws. In Section 4 we analyze the performance of the Apache server from various angles: the effect of transport protocols (HTTP1.0 vs. HTTP1.1), the impact of network conditions of the users, and the effect of the browsers' and server's control parameters of persistent connections. Finally in Section 5 we conclude the presentations and provide some practical guidelines with regard to the implementation of HTTP1.1 in Web browsers and Web server softwares.

2. Traffic model and statistical analysis

In this section we present our work on the HTTP traffic. We first propose a stochastic model of HTTP traffic at the session level. We then present statistical analysis results of some Web servers for such a model. We shall also provide validation results of the traffic model by comparing performance characteristics observed using synthetic traffic against those of real traffic.

2.1. Traffic model

We propose a stochastic model of HTTP traffic at session level. We confine ourselves to HTTP requests to the same Web server. We are interested in the typical behavior of users traversing the hypertext of a Web server, namely, when they arrive and how requests are generated. This can actually be well described by the notion of session: a sequence of clicks of a user on the hyperlinks of the same server.

The traffic model we propose can be described by a stochastic marked point process. The arrival times correspond to the beginning times of sessions. Each arrival is associated with the following random variables (the marks):

- number of clicks during the session;
- inter-click idle times (or user think times), i.e. time durations elapsed between the completion of the transfer of the previous requested page and the beginning of the transfer of the current page;
- transfer and CPU costs of the successive clicks.

Fig. 1 illustrates a user session, where τ_i denotes the time taken to retrieve Web pages (including embedded documents), and γ_j represents the inter-arrival times of clicks.

Note that the random variables representing transfer and CPU costs of the successive clicks can take different forms depending on the abstraction level of the model and on the performance evaluation technique. When it is used by a simulation tool or an analytical approach, they can be the total sizes of the pages (including embedded objects), or, in a more detailed level, vectors of sizes of objects of the pages. When it is used in a benchmark, they can be addresses of the successively required pages, i.e., the addresses of the first page and the successively clicked hyperlinks. In such a case, embedded objects are not specified and are requested automatically by browsers. Note also that CPU cost of a static page is usually negligible. It could however be dominant for dynamic pages.

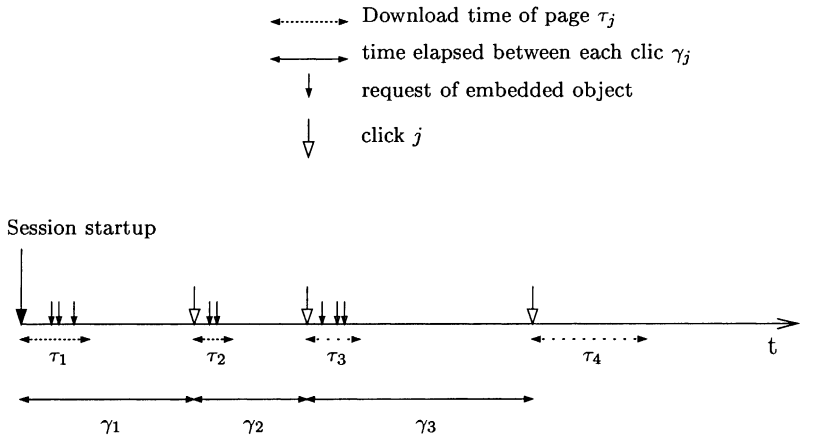

Fig. 1. User session.

It turns out (see below) that the session arrivals are well described by a renewal process, and in many cases, they are simply a Poisson process. This nice property allows for a simple way to model arrivals and to parameterize the traffic intensity. The distribution function of the number of clicks of a session seems to depend heavily on the traces. It can be short-tailed (such as geometric distribution) or sub-exponential (such as Pareto, Lognormal and Inverse Gaussian). The marginal distribution of the inter-click idle times most often belong to the class sub-exponential distribution functions including Pareto, Lognormal, Inverse Gaussian and Weibull distributions.

2.2. Statistical analysis

We report now some of the statistical analyses we performed on traces (log files) of Web servers. They are concerned with the Web servers at W3C (http://www.w3.org/), INRIA (http://www.inria.fr/) and www.clark.net (http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html). Some characteristics of these server traces are presented in Table 1.

The goal of this part of the work is to identify the statistical laws of the above described random variables of the traffic model. For ease of use of the traffic model in performance evaluation, we consider only the class of parametric distributions. Moreover, we are interested in distribution functions with a small number of parameters.

Table 1
Characteristics of the servers www.w3.org, www.inria.fr, www.clark.net, and the INRIA proxy cache

	www.w3.org	www.inria.fr	www.clark.net	INRIA proxy
Time period	February 1997	October 1996	September 1995	November 1999
Duration (h)	24	24	18	24
Total number of requests	275 000	50 000	150 000	125 000
Total number of pages	4726	7773	9294	9712
Average page size (KB)	23	15	13	9

Thus, we fix a small set of most commonly used exponential type and sub-exponential distribution functions including Exponential, Geometric, Weibull, Normal, Lognormal, Pareto and Inverse Gaussian distributions. These are the hypothetical distributions that samples are to be tested on. For each sample, we calculate the maximum likelihood estimators (except for some scale parameters) according to each of the hypothetical distributions, and compare the goodness-of-fit of these distributions.

In order to develop a tool that recognize the best fitting distribution function, we used, in conjunction with statistical test, a combination of three metrics: λ^2 [3,24], Cramer-von Mises and Anderson-Darling [8]. The first metrics is based on splitting the values into bins (as in the χ^2-test) and count the number of samples in each bins, and compare it with the theoretical distribution to match (see [23]):

$$\lambda^2 = \frac{\sum_{i=1}^{N}((Y_i - np_i)^2/np_i) - K - \text{d.f.}}{n-1},$$

where N is the number of bins, $p_1, p_2, \ldots, p_N$ the probability that the matching distribution fits in the ith bin. Y_i is the number of samples in bin i ($\sum_{i=1}^{N} Y_i = n$), $K = \sum_{i=1}^{N}(Y_i - np_i)/np_i$ and d.f. $= N - 1 -$ Est is the "degrees of freedom" (Est is the number of estimated parameters). The other two metrics are based on empirical cumulative distribution functions,

$$Q = \int_{-\infty}^{\infty} (F_n(x) - F(x))^2 \psi(x)\, \mathrm{d}F(x).$$

If $\psi(x) = 1$, it is called *Cramer–von Mises* statistics. If $\psi(x) = [F(x)(1 - F(x))]^{-1}$, it is the *Anderson–Darling* statistics.

As we mentioned previously, sample data of each of the random variables are extracted from the log files. Due to the lack of information available in *Common Log Files*, we used heuristics to identify sessions. In particular, we use two thresholds $T_{\min}$ and $T_{\max}$ to identify clicks with requests: the inter-click times should be larger than $T_{\min}$ (e.g. 2 s) and smaller than $T_{\max}$ (e.g. 10 min). Such kind of heuristics is also used in [16] for identifying the number of files in a page and other data from traces. More theoretical investigations were reported in [25] where different path reconstruction methods were described which are based on the available information in log files (cookies, referrer). The authors propose a Markovian model for routing probability.

For the session arrival process, according to this combined metric, the Poisson process has the best fit in these servers, with rate $\lambda = 0.39$ for www.w3.org, $\lambda = 0.037$ for www.inria.fr and $\lambda = 0.21$ for www.clark.net. Previous results in the literature concerning the analysis of HTTP and IP traffic have rejected the Poisson hypothesis, see e.g. [7,23]. However, if one considers the arrival process of sessions instead of the arrival process of requests or packets, the Poisson assumption turns out to be valid in these cases. It is worthwhile noticing that Poisson assumption does not hold for all session arrival processes. However, in all the statistical analyses we performed, the session arrivals do form a renewal process.

The numbers of clicks of sessions are independent. Their distributions are sub-exponential for the above mentioned three traces: inverse Gaussian ($\mu = 5.96$; $\sigma = 3$ for www.inria.fr) Pareto ($a = 0.748$; $\beta = 0.807$ for server www.w3.org, $a = 0.94$; $\beta = 1.16$ for www.clark.net).

3. Synthetic traffic generation and traffic model validation

3.1. Synthetic traffic generation using WAGON

We have developed a benchmark tool of Web servers: WAGON. It is composed of a generator of Web traffic, a robot which sends and analyzes requests and a monitoring tool. It can generate different types of traffic requests, which, in turn, can be sent out to the server from different machines with different (simulated) network delays and bandwidth constraints. It is written in Java for portability.

For this, user can use the graphic interface of WAGON and can specify

- Server features: server name or IP address, home page address of the server, server log files, etc.
- Client features: laws and parameters of the random variables in the traffic model; transport protocols and parameters used.
- Experiment configuration: local client caching, sampling rate, bandwidth constraint and network delay of different types of clients, etc.

Experiments using WAGON are carried out on a set of machines connected through a LAN. One machine is used as the server machine with the Web server software (such as Apache, IIS and Jigsaw) and the Web contents (i.e. the Web hypertext) installed on it. Other machines are considered as client machines in charge of sending HTTP requests to the server machine. Each of these machines is assigned one or several types of clients.

URL addresses are generated by WAGON according to the popularity of the pages of the Web hypertext. More precisely, for each page we compute the probability of getting a session started from it. We also compute the routing probabilities for each page: given the page is visited, what is the probability that a page it refers to will be visited next.

During a benchmark, various performance measures can be obtained and observed on-line through the monitoring tool of WAGON. Of particular interest are request latency and user perceived throughput.

3.2. Experimentation setup

In the sequel we shall present various experimentation results obtained using the traffic model and WAGON. Unless otherwise stated, the experimentation setup is the following.

For the experiments presented in the paper, we have chosen a subset of the Web server hypertext INRIA [2] as the Web server contents. It consists of the Web pages at the highest level of the INRIA server. The majority of the files are HTML files with in average five embedded images. There are 10 000 documents in total, with the average size 7.5 KB. Table 2 summarizes the first-order characteristics of the hypertext. Fig. 2 illustrates the empirical distribution of page sizes.

In all the experiments we shall report below, the Web traffic is generated using the above described traffic model under the additional assumption that the sequences of the inter-arrival times, of the number of clicks and of the inter-click idle times are all renewal processes which are mutually independent.

Although the assumption of mutual independence among the random variables is not justified, it turns out that the resulting synthetic traffic has characteristics close to the real traffic, see discussions in Section 3.3.

[2] http://www.inria.fr.

Table 2
Server content

	INRIA
Number of files	10627
Total size	255Mo
Number of Web pages (with popularity > 0)	1828
Number of HTML pages (id)	1024
Average document size	7.5Ko
Average page size (except icons)	38.5Ko
Average HTML size	4.0Ko
Mean number of embedded documents per HTML page	5.0
Mean number of links per HTML page	8.9

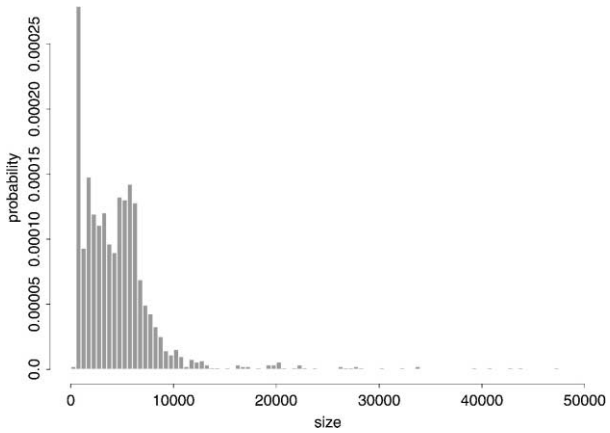

Fig. 2. Page size empirical distribution of INRIA server.

The parameters used for generating Web traffic are summarized in Table 3.

This workload corresponds to approximately 10 requests per second for the case of the smallest load ($\lambda = 0.0005$), and to 180 requests per second for the case of the highest load ($\lambda = 0.01$) we use is these experiments.

The experiments have been carried out on seven client machines and one server machine connected through a LAN of 100 Mbps. The clients machines are PC Pentium II 450 MHz with 128 MB main memory, under FreeBSD 3.2. The server machine is a PC Pentium Pro with 64 MB main memory, under FreeBSD 3.2. For WAGON, we use the JDK version 1.1.8 with JIT (tya) enabled.

Table 3
Client parameters used in the experimentation

Variable	Law	Mean	Standard deviation
Session arrival	Poisson process ($0.0005 \leq \lambda \leq 0.01$)		
Number of clicks	Inverse Gaussian ($\mu = 5$; $\lambda = 3$)	5	1.28
Inter-click idle time (s)	Lognormal ($m = 3$; $\sigma = 1.1$)	36.8	56.4

Table 4
Delay and bandwidth constraints

	Delay	Bandwidth
Modem	300 ms	56 Kbps
T1, DSL	20 ms	1.5 Mbps
WAN	180 ms	150 Kbps
Satellite	500 ms	2 Mbps

In all our experiments, we use only static documents. Therefore, all our conclusions are valid only for static page, and not for dynamic one. In the dynamic case indeed, the server is far more CPU intensively loaded, and the underlying dynamic processes (Perl CGI's, Database requests, Java Servlet, etc.) can play a major role on the performance of servers.

The Web server software we used in the experiments is Apache, [3] the currently mostly used Web server software according to Netcraft's survey: [4] about 12.5 million sites and 60% market share in September 2000. There are two key parameters used in the Apache server that control the number of connections of the server: the maximum number of connections (MaxClients, default value 150) and the timeout beyond which an idle connection is cut off (KeepAliveTimeout, default value 15 s). The maximum number of connections depend largely on the memory and CPU capacity of the Web server. The KeepAliveTimeout parameter, however, is to be set in accordance with the arrival traffic pattern. Unless otherwise specified, these default values are used in most experiments.

For network constraints, bandwidth are limited by WAGON for each client, whereas delays are simulated at the OS level, using the Dummynet [26] module available in the FreeBSD kernel. Delays are therefore simulated at the packet level.

We investigate the following different types of connections such as modem, T1 link, WAN, satellite connections, etc. In order to simulate such connections, we put the delay and bandwidth constraints as in Table 4.

3.3. Traffic model validation

We now present some results of experimentation which aims at validating the traffic model. We are interested in different characteristics of the resulting synthetic traffic: the self-similarity, request arrival process, document popularity and the stack distance. The results presented in this subsection are obtained using parameters of Table 3 with arrival rate $\lambda = 0.008$.

Self-similarity. According to the various experiments we carried out, the synthetic traffic is in most cases self-similar (except for the degenerate case with deterministic input, see discussions in Section 3.4). In Figs. 3–5, we illustrate the curves of the number of packets observed at different time scales in the experimentation network. Fig. 6 shows the estimated Hurst parameter $H = 0.73$.

HTTP request arrival process. We now investigate the auto-correlation structure of the arrival process of HTTP requests. Due to the lack of precision of log files, we shall analyze the sequence of number of HTTP requests per second instead of inter-arrival times. We observe from the auto-correlation curves of

[3] http://www.apache.org.
[4] http://www.netcraft.co.uk/survey/.

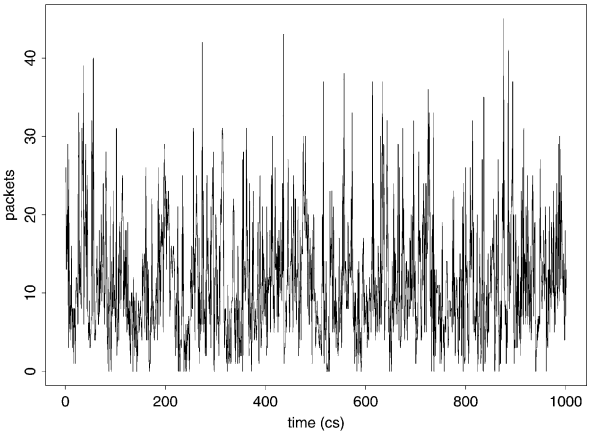

Fig. 3. Number of packets, scale = 10 ms.

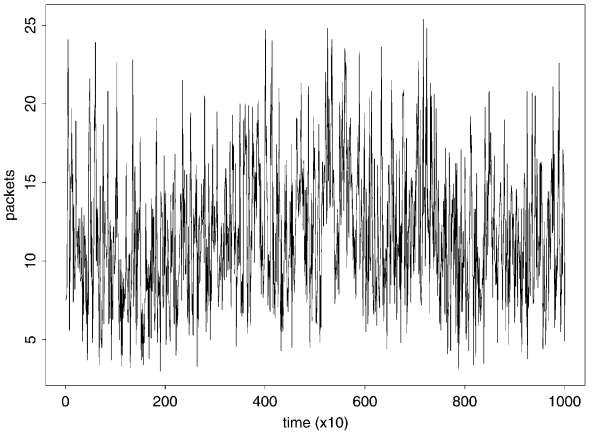

Fig. 4. Number of packets, scale = 100 ms.

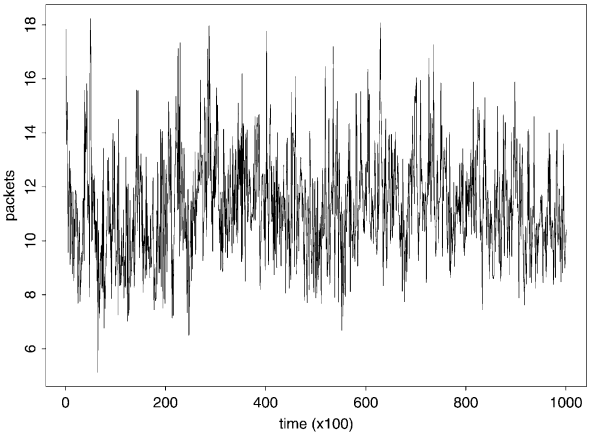

Fig. 5. Number of packets, scale = 1 s.

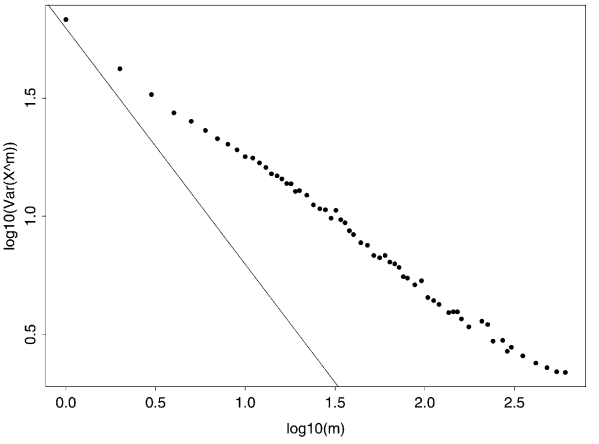

Fig. 6. Hurst estimation using variance analysis – H = 0.73.

Figs. 7 and 8 that the synthetic traffic and the real traffic are both highly auto-correlated with quite similar auto-correlation structure.

Document popularity. We next examine the page popularity resulting from the synthetic traffic. Fig. 9 illustrates the comparisons of real data and WAGON popularity curves (in the same log–log coordinate). The real data comes from the log files of INRIA during September 1998. We can see that the two popularity curves are quite close to each other, except for the lowest ranked documents.

It is worthwhile noticing that in the literature (see e.g. [5]) results were reported indicating that the document popularity can be characterized by Zipf type distribution with a slope close to or larger than 1. It turns out that in our case, for both real data and the WAGON traffic, Zipf type distribution does not seem to be suitable. One reason may be that in our experimentation we have chosen a subset of the INRIA Web server. If we consider the whole Web site of INRIA, Zipf distribution does fits well, see Fig. 10 where the straight line has slope -1.

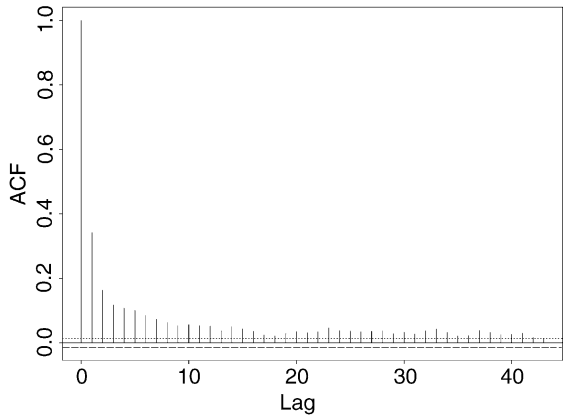

Fig. 7. Autocorrelation (number of requests per second), INRIA server.

Temporal locality. Another characteristic of interest is the so-called temporal locality of HTTP requests which plays an important role on the efficiency of caching algorithms. A usual way (cf. [1]) to measure temporal locality is to compute the "stack distance": the distances between requests to the same document (by counting the number of different requests between them). It was observed in [1] that the marginal of the stack distance is Lognormal.

While using this measure on our real data (computed with log files) and the synthetic traffic data, we also observed that the Lognormal distribution yields the best fit to the marginal distribution of the stack distance for both cases.

3.4. Impact of traffic parameters

We now investigate the impact of the random variables of the traffic model on the performance characteristics.

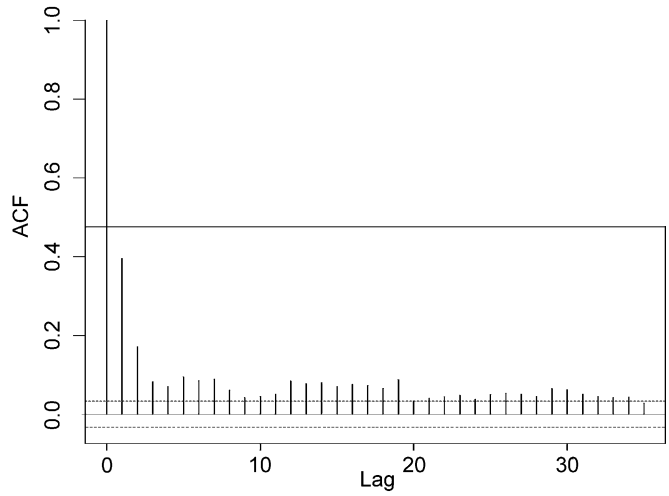

Fig. 8. Autocorrelation: number of requests per second obtained with WAGON.

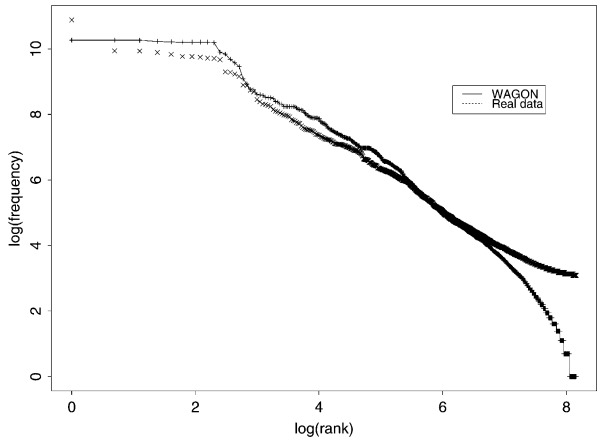

Fig. 9. Document popularity in real and WAGON data.

Impact of the randomness. We first consider the effect of the randomness of the key quantities of the traffic model, in particular, the inter-arrival times of the sessions, the number of clicks, the inter-click idle times. We use WAGON to simulate Web traffic in both the case where these quantities are random with the distributions described in Table 3 and the case where they are all deterministic with the same averages as in the random case.

Consider the resulting HTTP request arrival processes. As we mentioned previously, the marginal distribution of the inter-arrival times of HTTP requests in real data as well as in the synthetic traffic generated according to Table 3 is Weibull. In the case of deterministic input, however, the Lognormal distribution turns out to be the best fit (with positives tests on sub-samples). More importantly, the auto-correlation structure is completely different. As we already saw, in the random case, the auto-correlation and the Hurst parameter are close to those of real traces (Fig. 8). In the deterministic case, the auto-correlation function is close to zero (see Fig. 11) and the Hurst parameter is close to 0.5.

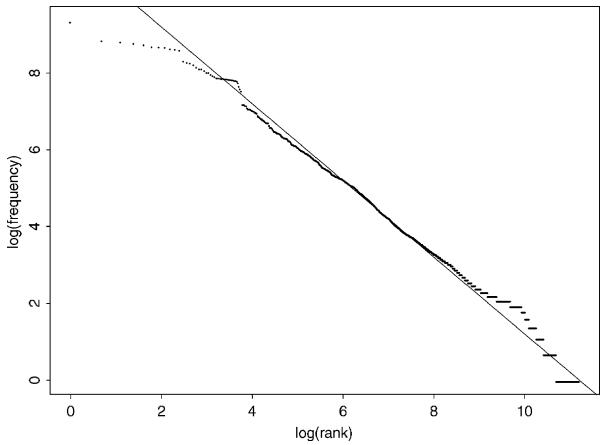

Fig. 10. Document popularity for the whole Web site of INRIA.

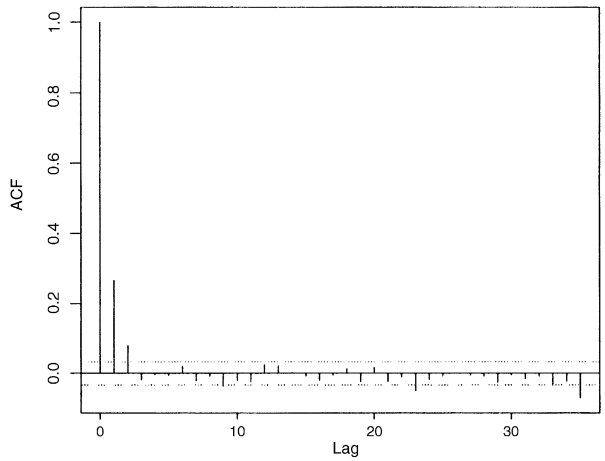

Fig. 11. Autocorrelation: number of requests per second obtained with deterministic input.

Recall that in both cases, documents (size, number of embedded documents per HTML page, popularity) and the mean arrival rate are the same. However, the resulting performance characteristics differ significantly. As a consequence, it is important for benchmarks to generate realistic synthetic traffic. Benchmarks such as SPECWeb [27] or WebStone [30], which use deterministic arrival process have a severe drawback in this regard.

Impact of the arrival rate. As is reported often in the literature, Web traffic as well as other Internet traffic has the long-range dependence and the self-similarity. A key parameter measuring these properties is the Hurst parameter which varies in between 0.5 and 1. A detailed analysis on the way the self-similarity depends on the TCP is presented in [9].

We investigate here how the Hurst parameter depends on the traffic intensity. For this, we vary the session arrival rate and measure the resulting traffic. We consider the situation when the network in between the

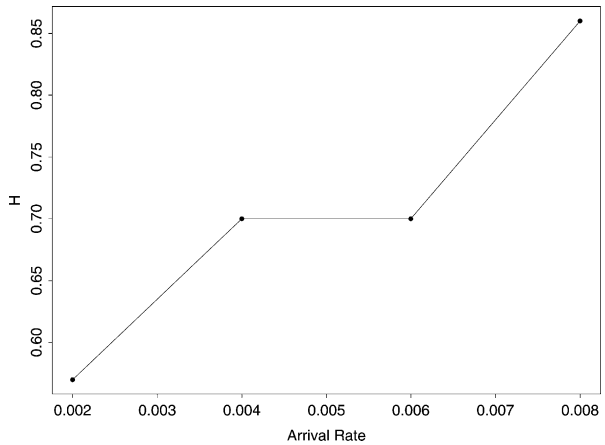

Fig. 12. Hurst parameter estimation when arrival rate increases.

server and the clients becomes the bottleneck when the arrival rate increases. This situation is configured such that all traffic in between server and clients goes through a router of bandwidth 10 Mbps.

In Fig. 12 we provide the Hurst parameter estimations as a functions of the arrival rate. Each of these estimations is obtained through variance analysis. We observe that the Hurst parameter increases in the traffic intensity.

4. Performance evaluation of the Apache server

In this section we analyze more specifically the performance of the Apache server. We are interested in how the server performance and the user perceived QoS depend on issues related to transfer protocols, user network conditions, browser control parameters and server control parameters.

Results reported in [4] are concerned with the impact of server hardware (CPU, memory, etc.) on the performance of Web servers. We shall be more interested in the impact of network conditions. For this purpose, we configure the experimental network in such a way that all traffic in between server and clients goes through a router of bandwidth 10 Mbps. In addition, clients have simulated individual network bandwidth and delay constraints as specified in Section 3.2, Table 4.

In this section we will analyze two QoS measures:

Response Time of Web pages: the time it takes to retrieve an entire page (HTML plus embedded objects)

HTML latency (*user perceived latency*): the time elapsed between the click (beginning of connection to the HTML page) and the time epoch where the first byte of data of the last image is available. For a user, the QoS is perceived mainly by the speed of display of the pages. The quicker, the better. To display a page, the browser needs to have the information on the content (HTML) and also on the embedded objects. The first bytes of images can give information (size) to the browser in order to display them. Therefore, it is important to get the first bytes of each embedded objects as fast as possible.

Most existing results of performance evaluation of HTTP1.0 and HTTP1.1 are concerned with the first measure, in addition to the network traffic, see e.g. comparisons between HTTP1.0 and HTTP1.1 [4,15,22]. Our benchmark WAGON allows us to measure detailed transaction information on the client side so that we will report comparison results on this user perceived QoS.

Note that in HTTP1.1, one can use the pipeline mechanism to send requests on the same connection without waiting for the completion of the previous transfer. Our extensive experiments show that HTTP1.1 with pipeline is uniformly better than HTTP1.1 without pipeline. Thus, in the following, as far as HTTP1.1 is concerned, we shall only report results of HTTP1.1 with pipeline.

However, it seems (cf. [32]) that request pipelining is not supported by the two most popular Web browsers (Netscape 4 and IE). We think that according to our analyses as well as those in the literature, one of the ways for HTTP1.1 to achieve significantly its performance gains over HTTP1.0 is through request pipelining on a persistent connection. We thus suggest that this request pipelining be implemented in browsers.

4.1. How persistent should persistent connections be

We first analyze the effect of the persistent connections of HTTP1.1. As it is now well known, most Web documents are small objects. However, in HTTP1.0, every object transfer requires to establish a new TCP session and to go through the slow-start phase. Thus it is interesting to reuse previously established

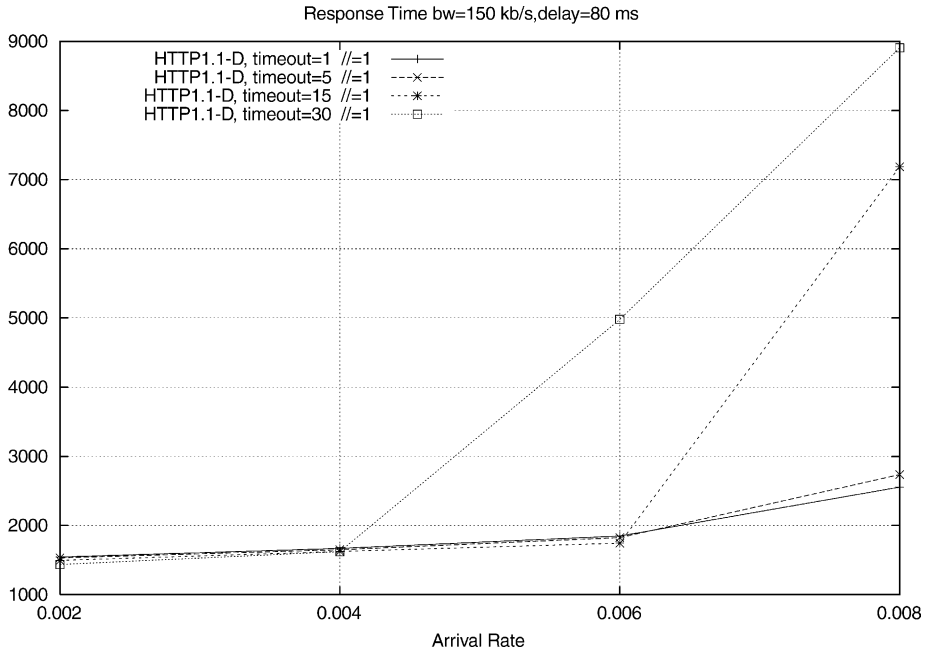

Fig. 13. Response time with Apache when the timeout increases.

connections to transfer such objects. The persistent connection is proposed in HTTP1.1 so that TCP sessions are kept alive. However, this has a cost. Mogul [18] pointed out that persistent connection has a higher main memory cost.

In Apache, the control parameter of persistent connections is KeepAliveTimeout which controls the duration above which an idle session is disconnected. The default value is 15 s. It is reported in [4] that when the disc system is the bottleneck, it is beneficial to keep connections open just for the transfers of a Web page.

In Fig. 13, we study the response time by varying the timeout parameter of the server. We consider only clients with bandwidth 150 Kbps and network delay 80 ms. The other configurations have similar behavior.

We observe that when the server load is small, it is beneficial to keep connections alive for long time. However, when the server load increases, this advantage vanishes and becomes a drawback due to its memory cost. The improvement of persistent connections (when there are) with large timeout values are rather small. This gain depends heavily on the RTT (round trip time) between the client and the server. At low traffic intensity, the improvement due to persistent connections is at best around 10%. When the traffic intensity grows, the increase in the response time is exponentially fast in the timeout value. This phenomenon could be explained theoretically below as an issue of workload or stability region.

4.2. Queueing model for the Apache server

Based on the traffic model presented in Section 2, we analyze the workload brought in to the server by a session. We fix the parameter of the maximum number connections (MaxClients) and analyze the

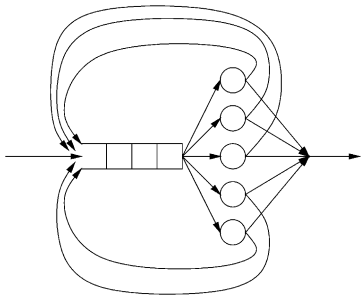

Fig. 14. Apache server model.

impact of the timeout parameter (KeepAliveTimeout). As this parameter has effect only on persistent connections, we confine ourselves to the HTTP1.1 sessions with persistent connections.

We propose to use a multi-server queueing system to model the Apache server. Consider a G/G/m queueing system with repetitive customers, illustrated in Fig. 14. There are m servers and an infinite-size queue in the system. Customers arrive according to a general, say stationary and ergodic, process and get served according to FCFS (first come first serve). The servers represent the processes serving the persistent connections. Customer arrivals correspond to the session arrivals.

When a customer starts service on one of the servers, it iterates the service and idle phases. The service time corresponds to the transfer time of an entire Web page. The first service time also includes the connection establishing time and the slow-start phase in TCP. The first transfer is thus usually longer.

The idle time corresponds to the click idle time. The customer stays on the server until either the session finishes or the connection is timed out while it is in the idle state. If the connection is timed out, the session goes back to the queue when the next click is done.

As with the synthetic traffic generation, we assume that the transfer times of Web pages (resp. inter-click idle times, numbers of clicks) are independent and identically distributed random variables. Moreover, these random variables are assumed to be mutually independent.

Let X_1 be the (generic) random variable representing the transfer time of a Web page when a new connection is used (it thus includes the connection establishing time and also the slow-start delay). Let X_2 be the (generic) random variable representing the transfer time of a Web page when an existing connection is used. Let Y be the (generic) random variable representing the click idle time, and N the random variable representing the number of clicks in a session. Finally, let Δ denote the timeout duration.

Define $p = P(Y \geq \Delta)$ as the probability that the connection is timed out. It can be shown using Wald's identity (see e.g. [13, p. 264]) that the expected number of times that a session is timed out is given by $E[Q] = E[N-1]p$. Indeed, if $Y_1, Y_2, \ldots, Y_n, \ldots$ is the sequence of inter-click idle times, then, thanks to Wald's identity,

$$E[Q] = E\left[\sum_{n=1}^{N} 1(Y_n \geq \Delta)\right] = E[N]E[1(Y_1 \geq \Delta)] = pE[N].$$

Similarly, using again Wald's identity we can show that total expected work brought in by a session W is

$$\begin{aligned} E[W] &= E[X_1] + pE[N-1]E[X_1] + (1-p)E[N-1]E[X_2] + (1-p)E[N]E[Y|Y < \Delta] \\ &= E[X_1] + E[N-1]E[X_2] + pE[N-1](E[X_1] - E[X_2]) + (1-p)E[N]E[Y|Y < \Delta]. \end{aligned}$$

Denote by $f(x)$ and $F(x)$ the density function and the cumulative distribution function of Y, respectively. Then,

$$\begin{aligned} E[Y|Y<\Delta] &= \int_0^\infty P(Y \geq x|Y<\Delta)\,\mathrm{d}x = \int_0^\infty \frac{P(x \leq Y < \Delta)}{P(Y<\Delta)}\,\mathrm{d}x \\ &= \frac{1}{1-p}\int_0^\Delta P(x \leq Y < \Delta)\,\mathrm{d}x = \frac{1}{1-p}\int_0^\Delta \int_x^\Delta f(y)\,\mathrm{d}y\,\mathrm{d}x \\ &= \frac{1}{1-p}\int_0^\Delta \mathrm{d}y \int_0^y f(x)\,\mathrm{d}x = \frac{1}{1-p}\int_0^\Delta F(y)\,\mathrm{d}y. \end{aligned}$$

Hence,

$$E[W] = E[X_1]+E[N-1]E[X_2] + (1-F(\Delta))E[N-1](E[X_1]-E[X_2]) + E[N]\int_0^\Delta F(x)\,\mathrm{d}x,$$

so that the derivative of $E[W]$ with respect to Δ is

$$\frac{\mathrm{d}E[W]}{\mathrm{d}\Delta} = -f(\Delta)E[N-1](E[X_1]-E[X_2]) + F(\Delta)E[N].$$

Thus, we obtain

Theorem 1. *Assume that the transfer times of Web pages (resp. inter-click idle times, numbers of clicks) are independent and identically distributed random variables. Assume further that these random variables are mutually independent. Session arrivals form a general stationary and ergodic stochastic process. Let*

$$\delta := \frac{E[N-1]}{E[N]}(E[X_1]-E[X_2]).$$

Then the timeout value Δ that minimizes the workload is either 0 *or a solution of the equation*

$$F(\Delta) - \delta f(\Delta) = 0. \tag{1}$$

The difference $(E[X_1]-E[X_2])$ can be considered as the gain of transfer time by using persistent connections. According to our measurements, the average gain is usually upper bounded by twice of the RTT. We can assume without loss of generality that $\delta < 1$ s. A more realistic assumption is actually $\delta < 400$ ms.

When the inter-click idle time has a Weibull distribution, $f(x) = (bx^{b-1}/a^b)\,\mathrm{e}^{-(x/a)^b}$ and $F(x) = 1-\mathrm{e}^{-(x/a)^b}$. As $\mathrm{e}^{(x/a)^b} \simeq 1+(x/a)^b$ we obtain from (1) that the optimal timeout satisfies $\Delta \simeq \delta b < b$. If the shape parameter $b < 1$, in which case Y has sub-exponential tail distribution, we conclude that Δ should be smaller than 1 s. The same conclusion holds for exponentially distributed inter-click idle times.

If, however, the inter-click time has a Lognormal distribution, $f(x) = (1/x\sigma\sqrt{2\pi})\exp(-\frac{1}{2}((\ln x - m)/\sigma)^2)$, then we need to numerically solve Eq. (1). For the case where our traffic is generated, $m = 3$; $\sigma = 1.1$, which implies mean 36.8 s and standard deviation 56.4 s, the optimal value turns out to bounded by 2 s, even for δ close to 1, see Fig. 15. If we assume that $\delta < 400$ ms, we then obtain $\Delta < 1$ s.

We conjecture that the optimal value of the timeout is increasing in δ and is usually small (within seconds). Since the inter-click idle times are usually larger than 2 s, we see that Early Close is near

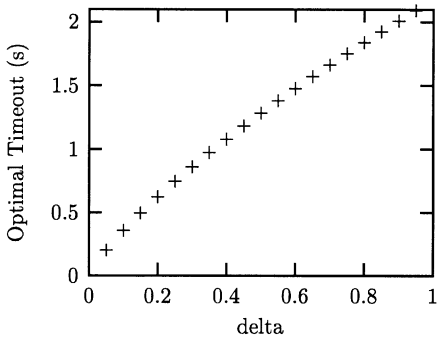

Fig. 15. Optimal timeout value for Lognormal ($m = 3$; $\sigma = 1.1$).

optimal for the workload minimization. Under the Early Close scheme (see [4]), persistent connections are closed by clients after each Web page (i.e. after retrieving HTML document and all of its embedded objects).

Note that the total arrival traffic intensity is given by $\lambda E[W]$, where λ is the arrival rate of sessions. We conjecture that the usual stability condition $\lambda E[W] < m$ holds for this queueing system with m servers. In this case, the solution of Theorem 1 also maximizes the stability region of the Web server.

As a conclusion of the results of Sections 4.1 and 4.2, we suggest that if HTTP1.1 is used, browsers implement Early Close policy and servers set small timeout values if fixed timeout control mechanism is used (as in Apache). If dynamic timeout control mechanism is used, however, then small timeout value should be used once the measured workload is high.

4.3. Comparison of HTTP/1.0 and HTTP/1.1

Previous comparison results on protocols HTTP/1.0 vs. HTTP/1.1 are mostly concerned with amount of traffic that HTTP/1.1 saves [22] and the effect of server hardware components [4]. We provide here comparison between the protocols HTTP/1.0 and HTTP/1.1 with different network conditions. These comparison results complement those of [4,22].

For HTTP/1.0, we use four parallel connections. For HTTP/1.1, we use a single persistent connection using pipelined requests. The KeepAliveTimeout parameter is the default value (15 s). We also look at the Early Close scheme where persistent connections are closed by clients after each Web page. In the sequel, HTTP/1.1 with default KeepAliveTimeout parameter is denoted as HTTP/1.1-D, HTTP/1.1 with Early Close is denoted as HTTP/1.1-EC.

The experimentation results are summarized in Fig. 16 for response times and Fig. 17 for latencies (both as a function of arrival intensities).

When the traffic intensity is small, HTTP/1.1-D provides smaller response time, whatever client network condition is. However, when the traffic grows, HTTP/1.1-D performs very badly. Indeed, the number of simultaneous connections reaches its maximum (MaxClients). Apache server in this case refuses new connections until an old one has finished (this happens when an existing connection times out or when a client closes its connection). However, with Early Close policy, HTTP/1.1 is still better than HTTP/1.0, even when the traffic load is very high.

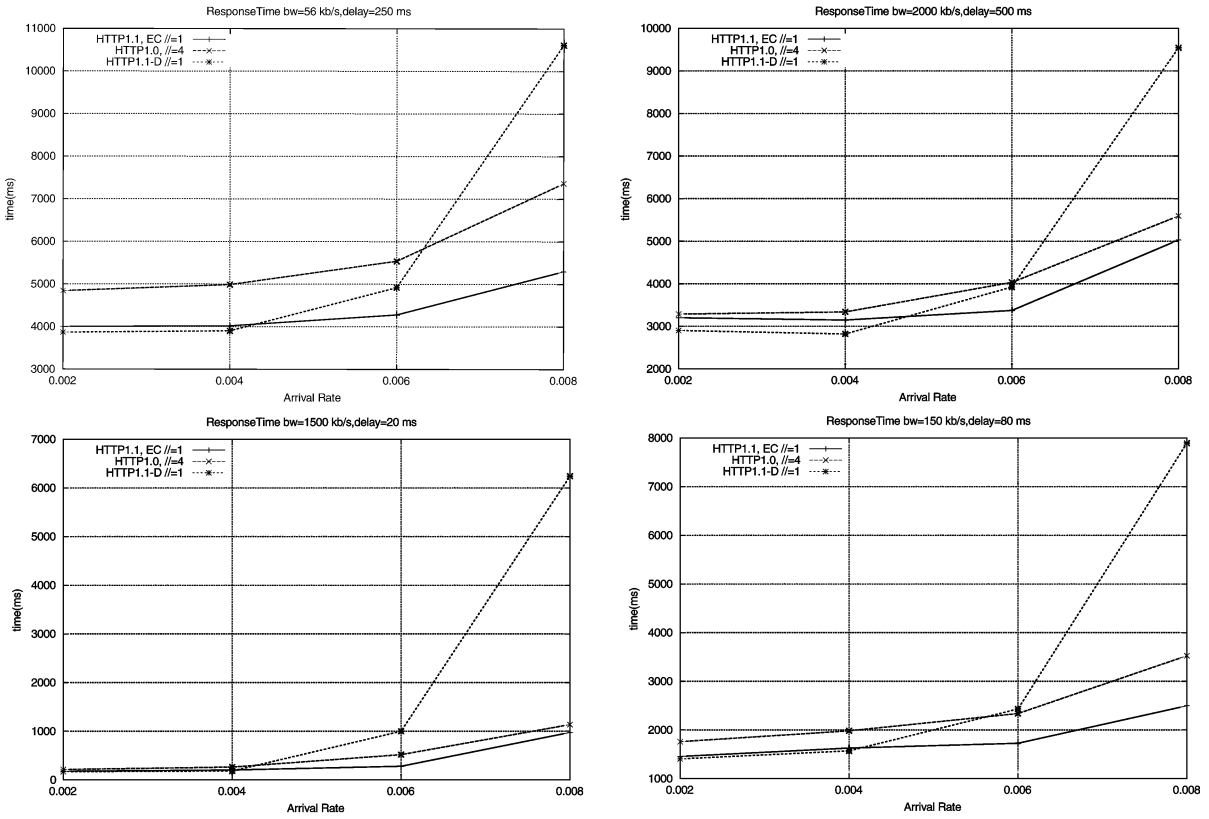

Fig. 16. Comparison between HTTP1.0 and HTTP1.1: mean response time.

Consider now the user perceived QoS: HTML latency. The results are quite different. HTTP/1.1 no longer outperforms HTTP/1.0 in this regard. Indeed, with a single connection, HTTP/1.1 serializes its requests of all the objects of a page. Thus, multiple parallel connections of HTTP/1.0 can yield a smaller latency, even if all these connections have to go through the connection establishment phase and the TCP slow-start phase.

HTTP1.0 uniformly outperforms HTTP1.1-D with respect to the HTML latency. The difference in latency between the two protocols increases as the traffic intensity grows. Compared to HTTP1.1-EC, the advantage of HTTP1.0 is particularly important with small-bandwidth clients (e.g. modems), in which case, the latency of HTTP/1.0 is better off by about 25%. The difference in latency between HTTP1.0 and HTTP1.1-EC decreases as client bandwidth increases. At high speed (1.5 Mbps), the performances are almost equal.

In light of this comparison of latency, one might be interested in implementing multiple parallel persistent connections with HTTP/1.1. This will be the subject of our discussions in the next subsection.

4.4. How many parallel persistent connections should there be

Since parallel HTTP1.0 requests could reduce the latency and HTTP1.1 persistent connections could reduce network traffic as well as page response time, one is tempted to implement parallel persistent

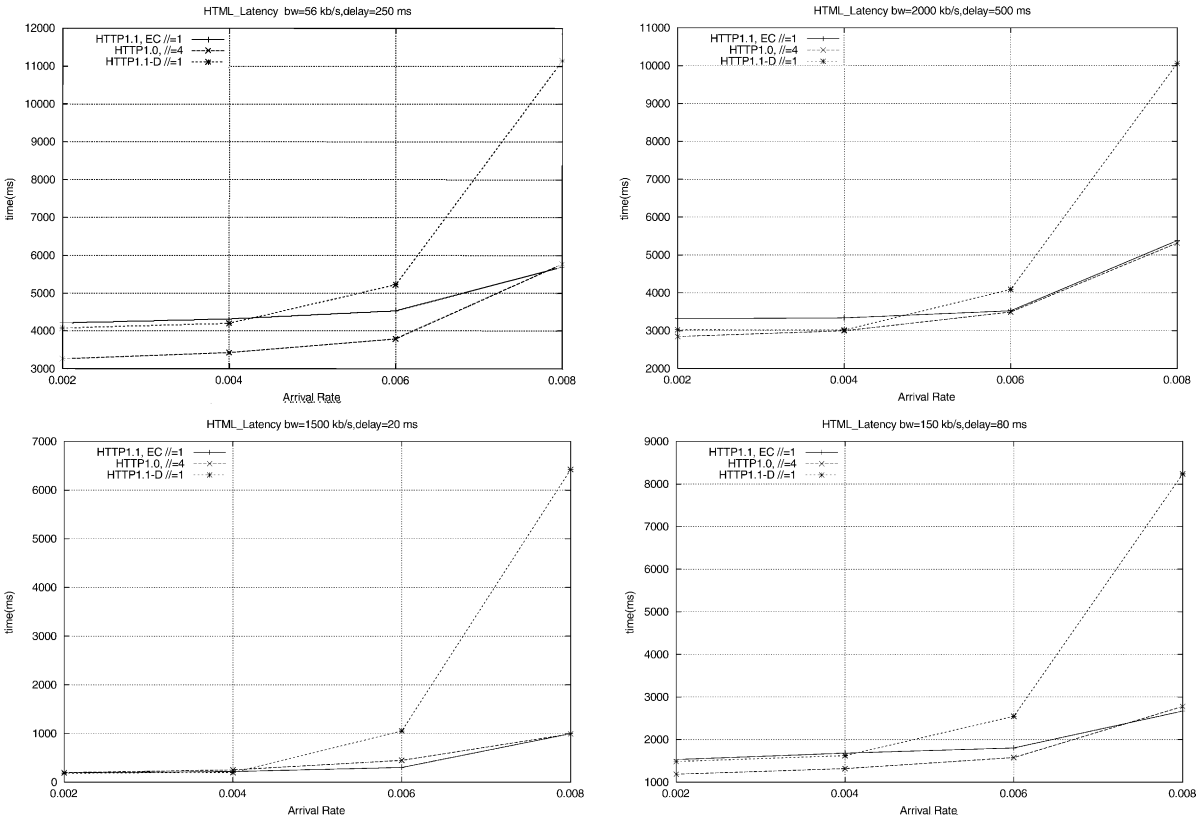

Fig. 17. Comparison between HTTP1.0 and HTTP1.1: mean HTML latency.

connections in order to combine the advantages of both protocols. It seems to be indeed the case in the two most popular browsers [32]: Netscape seems to use up to six parallel persistent connections to the same Web server, and IE seems to use two.

We show in Fig. 18 that under both HTTP1.1-D and HTTP1.1-EC, the response time increases exponentially with the number of parallel connections. With the same number of parallel persistent connections, the performance under HTTP1.1-EC is usually better than that under HTTP1.1-D, except for the cases of light loads. The gain of HTTP1.1-D over HTTP1.1-EC in light load, however, is much smaller (several orders of magnitude difference) than the gain of HTTP1.1-EC over HTTP1.1-D in medium or heavy load.

Concerning the latency, Fig. 19 shows that under HTTP1.1-D, the latency increases exponentially with the number of parallel connections. The behavior of HTTP1.1-EC, however, is quite similar to that of HTTP1.0. There is gain, especially for low bandwidth connections, in using multiple parallel HTTP1.1-EC persistent connections when the load is light. This gain vanishes when the network bandwidth increases. In the heavy load case, even under HTTP1.1-EC, it is harmful to use multiple parallel persistent connections: the latency increases exponentially with the number of parallel persistent connections.

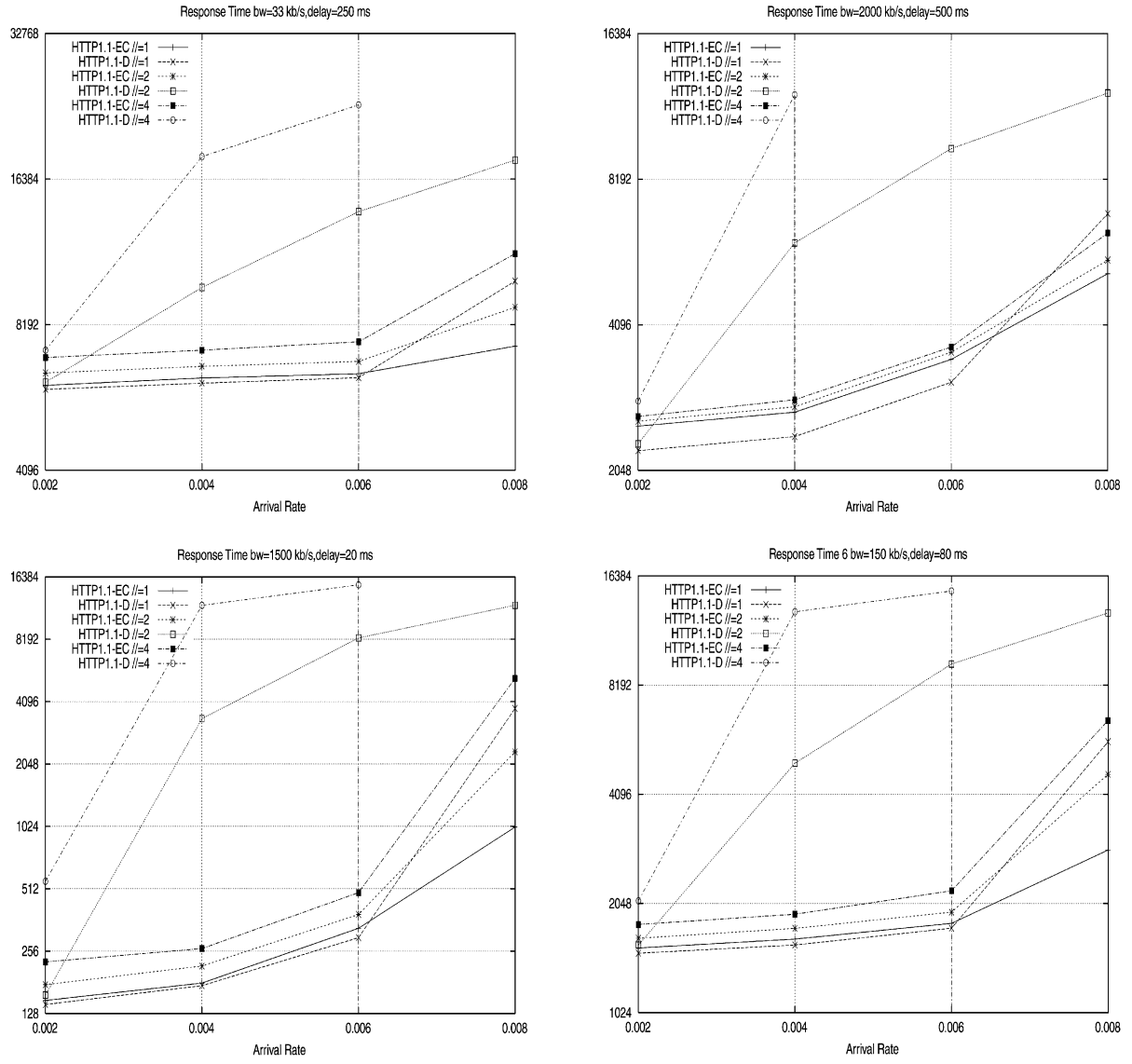

Fig. 18. Effect of multiple parallel persistent connections: mean response time.

In view of the results of Sections 4.3 and 4.4, we conclude that HTTP1.1 with Early Close combines the advantages of both HTTP1.0 (latency minimization with parallel connections) and HTTP1.1. We think that browsers should, in general, implement HTTP/1.1 with Early Close (and pipelining). We suggest that if multiple parallel persistent HTTP1.1-EC connections are used in browsers, they should be used only for low bandwidth network connections. In other words, except for low bandwidth network connections, browsers should open only one persistent connections to the same Web server from one browser window. Multiple parallel persistent connections are usually harmful.

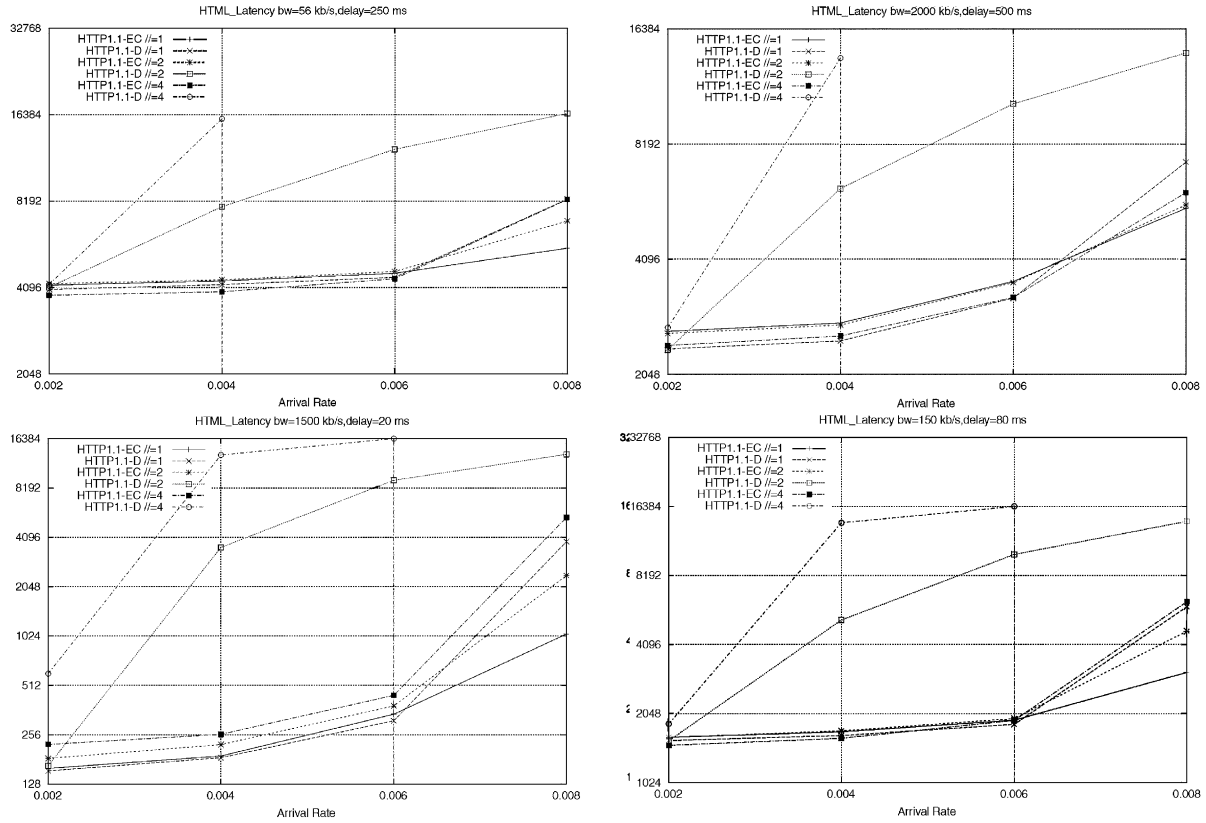

Fig. 19. Effect of multiple parallel persistent connections: mean HTML latency.

5. Concluding remarks

We have presented a new model of Web traffic and its applications in the performance evaluation of Web servers. We have proposed a traffic model at the session level, formulated as a stochastic marked point process, which describes when clients arrive and how they navigate on the Web site. We have provided results of statistical analyses and goodness-of-fit of various simple parametric distributions. We have developed a Web server benchmark WAGON, and we have validated the traffic model by comparing various characteristics of the synthetic traffic generated by WAGON against measurements.

Using this benchmark tool we have performed extensive experiments, most of which are difficult to accomplish with other existing benchmarks. We carried out our experiments on the Apache server, the currently most used Web server software. We have analyzed the impact of the traffic parameters on the HTTP request arrival process and the packet arrival process. We have shown that the aggregate traffic is self-similar in most cases, and that, more importantly, the Hurst parameter is increasing in the traffic intensity.

We have provided further performance comparison results between HTTP1.0 and HTTP1.1, with emphasis on the effect of network conditions. We have shown that HTTP1.1 could be much worse for

clients as well as for servers if the timeout value for persistent connections used by the server is too large or if browsers use multiple parallel persistent connections to the same server without Early Close.

With regard to HTTP1.1, we have shown that request pipelining is useful, and that response time depends tightly on the timeout value used by server for managing persistent connections. When the workload traffic is light, it is beneficial (with a small gain) to set a large timeout value. When the workload traffic is heavy, however, the response time grows exponentially in the timeout value. We have also proposed a queueing model to analyze the workload of persistent connections on the Apache server, and we have established optimal solution of the timeout parameter that minimizes the workload. HTTP1.1 with Early Close turns out to be near optimal. We have moreover shown that multiple parallel persistent connections are harmful: the response time increases exponentially in the number of parallel connections.

Based on our analyses, we make the following suggestions for the implementation and for the parameterization of Web server softwares and Web browsers. Browsers should in general support HTTP1.1 with request pipelining and implement HTTP/1.1 with Early Close. HTTP1.1 with Early Close combines the advantages of both HTTP1.0 and HTTP1.1. Browsers should in general avoid establishing multiple parallel persistent connections from one browser window to the same Web server. Multiple parallel connections could be beneficial only to clients with low bandwidth connections (such as modem). On the server side, for the management of persistent connections, servers should set small timeout values if fixed timeout control mechanism is used (as in Apache) or if dynamic timeout control mechanism is used and the measured workload is high.

We are currently evaluating performances of Web server softwares other than Apache. We use the benchmark WAGON to compare these Web servers. We are also extending the traffic model and the benchmark tool for Web caches. More theoretical work is undergoing on multi-server queueing system proposed in Section 4.1. We are analyzing formally the stability condition that allow for the weak convergence of response times. We are also computing approximate solutions of the response time and their minimization by an optimal solution of the timeout parameter. In parallel, we are pursuing the investigations on the monotonicity of the Hurst parameter in the traffic intensity. Finally, we are investigating the multiplexing scheme proposed in HTTP-NG [31]. In view of the results we obtained in this paper, we believe that a solution better than HTTP1.1-EC could be achieved by using sending (or multiplexing) parallel requests on the same connection.

Acknowledgements

The authors are grateful to Dr. Jean-François Abramatic for his support and various useful comments on this work. The work of César Jalpa Villanueva was supported in part by the grant from the CONACYT and the UAM.

References

[1] V. Almeida, M.E. Crovella, A. Bestavros, A. de Oliveira, Characterizing reference locality in the WWW, in: Proceedings of the PDIS'96: The IEEE Conference on Parallel and Distributed Information Systems, Miami Beach, FL, December 1996.

[2] M. Arlitt, C. Williamson, Web server workload characterisation: the search for invariants, in: Proceedings of the 1996 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Philadelphia, PA, May 1996.

[3] P. Barford, M.E. Crovella, Generating representative web workloads fot network and server performance evaluation, in: Proceedings of the ACM Sigmetrics'98, 1998.

[4] P. Barford, M.E. Crovella, A performance evaluation of hyper text transfer protocols, in: Proceedings of the ACM Sigmetrics'99, 1999.
[5] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and zipf-like distributions: evidence and implications, in: Proceedings of the INFOCOM'99, 1999.
[6] M.E. Crovella, A. Bestavros, Explaining world wide web traffic self-similarity, Tr-95-015, Computer Science Department, Boston University, 1995.
[7] M.E. Crovella, M.S. Taqqu, A. Bestavros, Heavy-tailed probability distributions in the world wide web, in: R. Adler, R. Feldman, M. Taqqu (Eds.), A Practical Guide to Heavy Tails: Statistical Techniques for Analysing Heavy Tailed Distributions, Birkhäuser, Boston, 1998, pp. 3–25.
[8] R.B. D'Agostino, M.A. Stephens (Eds.), Goodness-of-fit Techniques, Marcel Dekker, New York, 1986.
[9] A. Feldmann, P. Huang, A.C. Gilbert, W. Willinger, Dynamics of ip traffic: a study of the role of variability and the impact of control, in: Proceedings of the ACM/SIGCOMM'99, 1999.
[10] B.A. Huberman, P.L.T. Pirolli, J.E. Pitkow, R. Lukose, Strong regularities in world wide web surfing, Science 280 (1998).
[11] A.K. Iyengar, M.S. Squillante, L. Zhang, Analysis and characterization of large-scale web server access patterns and performance, World Wide Web 2 (1999) 85–100.
[12] C. Jalpa-Villanueva, Modélisation et optimisaion du Web, Ph.D. Thesis, Université de Nice Sophia-Antipolis, 2001, in press.
[13] S. Karlin, H.M. Taylor, A First Course in Stochastic Processes, Academic Press, New York, 1975.
[14] B. Liu, Characterizing web response time, Master's Thesis, Virginia Polytechnic Institute and State University, April 1998.
[15] Z. Liu, N. Niclausse, C. Jalpa-Villanueva, Web traffic modeling and performance comparison between HTTP1.0 and HTTP1.1, in: E. Gelenbe (Ed.), System Performance Evaluation: Methodologies and Applications, CRC Press, Boca Raton, August 1999.
[16] B.A. Mah, An empirical model of HTTP network traffic, in: Proceedings of the IEEE INFOCOM'97, 1997.
[17] S. Manley, M. Courage, M. Seltzer, A self-scaling and self-configuring benchmark for web servers, Technical Report, Harvard University, 1998.
[18] J.C. Mogul, The case for persistent-connection HTTP, in: Proceedings of the SIGCOMM'95 Conference, Cambridge, MA, August 1995.
[19] J.C. Mogul, Network behavior of a busy web server and its clients, Technical Report, Digital, Western Research Laboratory, October 1995.
[20] D. Mosberger, T. Jin, httperf — a tool for measuring web server performance, in: Proceedings of the Workshop on Internet Server Performance (WISP'98), Madison, WI, June 1998.
[21] N. Niclausse, Modélisation, évaluation de performances et dimensionnement du World Wide Web, Ph.D. Thesis, Université de Nice Sophia-Antipolis, June 1999.
[22] H. Frystyk Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, C. Lilley, Network performance effects of http/1.1, css1, and png, in: Proceedings of the ACM SIGCOMM'97, Cannes, France, September 1997.
[23] V. Paxson, S. Floyd, Wide-area traffic: the failure of Poisson modeling, in: Proceedings of the ACM/Sigcomm'94, September 1994, pp. 257–268.
[24] V. Paxson, Empirically-derived analytic models of wide-area tcp connections, IEEE/ACM Trans. Networking 2 (4) (1994).
[25] P.L.T. Pirolli, J.E. Pitkow, Distributions of surfers' paths through the world wide web: empirical characterizations, World Wide Web 2 (1–2) (1999) 29–45.
[26] L. Rizzo, Dummynet: a simple approach to the evaluation of network protocols, ACM Comput. Commun. Rev. 27 (1) (1997).
[27] SPEC: an explanation of the specweb96 benchmark, The Standard Performance Evaluation Corporation, 1996.
[28] M.S. Squillante, D.D. Yao, L. Zhang, Internet traffic: periodicity, tail behavior and performance implications, in: E. Gelenbe (Ed.), System Performance Evaluation: Methodologies and Applications, CRC Press, Boca Raton, August 1999.
[29] M.S. Squillante, D.D. Yao, L. Zhang, Web traffic modeling and server performance analysis, in: Proceedings of the IEEE Conference on Decision and Control (CDC), 1999.
[30] G. Trent, M. Sake, Webstone: the first generation in http server benchmarking, 1995.
[31] Http-ng working group. http://www.w3.org/protocols/http-ng/.
[32] Z. Wang, P. Cao, Persistent connection behavior of popular browsers. http://www.cs.wisc.edu/cao/papers/persistent-connection.html.