A Behavioral Model of Web Traffic

Hyoung-Kee Choi

School of Electrical and Computer Engineering

John O. Limb

College of Computing

Georgia Institute of Technology

{hkchoi, limb}@cc.gatech.edu

Abstract

The growing importance of Web traffic on the Internet makes it important that we have accurate traffic models in order to plan and provision. In this paper we present a Web traffic model designed to assist in the evaluation and engineering of shared communications networks. Because the model is behavioral we can extrapolate the model to assess the effect of changes in protocols, the network or user behavior. The increasing complexity of Web traffic has required that we base our model on the notion of a Web-request, rather a Web page. A Web-request results in the retrieval of information that might consist of one or more Web pages. The parameters of our model are derived from extensive traces of Web traffic. Web-requests are identified by analyzing not just the TCP header in the trace but also the HTTP headers. The effect of Web caching is incorporated into the model. The model is evaluated by comparing independent statistics from the model and from the trace. The reasons for differences between the model and the traces are given.

1 Introduction

World Wide Web (Web) traffic continues to increase and is now estimated to be more than 70 percent of the total traffic on the Internet [4]. Consequently, it is important that we have an accurate model of this important source of traffic so that we can simulate the performance of new system designs, and experiment with alternative designs. With a parametric model we can track the changes in parameters over time and estimate the nature of the future traffic.

Web traffic modeling is difficult for two reasons. Firstly, many of the system components interact with one another. Web browsers and Web servers from different vendors behave differently and have different parameter values. *HyperText Transfer Protocol* (HTTP) is changing and different versions coexist and interact. While *Transmission Control Protocol* (TCP) is relatively stable, different implementations of TCP behave slightly differently depending on the operating system.

Secondly, a Web interaction becomes more complex because of the changing nature of the Web environment. Browsing patterns of different users are diverse. A user may purposely or accidentally open multiple browsers and generate pages from these browsers at the same time. A user may abandon the on-going page in the middle by moving to another page or clicking the Back or Stop button. Current publishing tools enable a browser to request multiple pages at once (e.g. frames [3] and Java-scripts). The frame allows authors to present independently designed pages inside of sub-windows as if they were a single page. Java-script cooperates with *HyperText Markup Language* (HTML) code and enables authors to present multiple pages in an independent window.

An implication of all this volatility is that there is no single or quintessential template of a Web interaction. Thus, the boundary between Web pages has become blurred, particularly when a single request generates multiple pages, and objects belonging to different pages are overlapped at the time of downloading. A Web page has been a basic unit of most past work [8][12]. A request by a user resulted in a single page being fetched. A one-to-one correspondence between a request and a page no longer exists. Hence, we need to select a more general entity as a basic unit and we adopt a *Web-request*. A Web-request is a page or a set of pages that results from an action of a user. This results in a model that more imitate user behavior.

Without having a method to accurately determine the boundary of a Web-request from the trace of activity the model will not be accurate. We determine the boundary of a Web-request using HTTP header information as well as TCP information. The HTTP header allows us to access higher-level information such as a *Uniform Resource Indicator* (URI), content type, content length and the status of a URI. This additional information makes our method accurate. One could have a browser record the boundary of a Web-request by modifying the source of popular browsers. Past work modified the browser "Mosaic" [12]. However, Mosaic is now an outdated browser. It is difficult to instrument the two most popular browsers, "Netscape" and "Internet Explorer". A difficulty associated with instrumenting a browser is that it would have to be distributed widely in order to obtain a representative cross-section of traffic. For this reason we ruled this out as an option.

The aim of our model*in*g is to produce a pattern of traffic on a simulated network that closely resembles to the pattern of traffic on a real network that is supporting the same number of users. By “closely resemble” we mean that we should be able to use the model in the design of the network including such parameters as buffer sizes and be able to determine accurate measures of Web performance, such as average time to receive a Web-request [19]. This requires a trace that is large enough to be statistically representative in all the parameters that we attempt to capture in the model.

Our modeling process consists of four steps. First, we collect a trace. Second, we identify a set of parameters which describe Web interactions and extract distributions of the parameters from the trace (see **Table 2**). Third, we generate the synthetic traffic, based upon the parameters. Finally, the synthetic traffic is statistically compared with the real traffic.

The model we present in this paper differs from previous models in a number of ways. The basic unit of our model is not a Web page but a Web-request. The boundary of a Web-request is determined by HTTP header information as well TCP header information. Our model simulates detailed dynamics of TCP/IP as well as HTTP. Finally, the integrity of the model is tested by comparing independent parameters of the trace and the model.

The results indicate that our model mimics the real traffic with respect to duration of a Web-request and the variation of the demanded bandwidth. Further, the traffic generated by the model exhibits a degree of self-similarity, close to that shown by the real traffic.

The remainder of this paper is organized as follows. Section 3 overviews HTTP and our model. Section 4 describes our processing of the traces. In Section 5, the detailed model is described. Section 6 describes the generation of traffic based on the model. Section 7 evaluates the model. We conclude, in Sections 8 and 9, with a discussion of the limitations and the extensions of the model.

2 Related Work

In the past, several studies have attempted to characterize Web traffic. Crovella and Bestavros [10] showed evidence of self-similarity in Web traffic, based upon distributions of object size and user viewing time and the effects of caching and user preference.

In [8], Mah derived statistical properties of a set of Web traffic parameters. The boundary of a Web page was determined by searching for a gap in the communication stream that was greater than a period of one second. This is a less accurate method for determining Web page boundaries, because they relied on just the TCP information in their analysis of traces. This work did not present or test a Web traffic model.

In [12], Badford and Crovella built a Web model so called "SURGE", based on the parameters: 1) Distribution of object size on the server, 2) Distribution of the size of requested objects, 3) Object popularity, 4) Number of in-line objects, 5) Temporal locality, and 6) User viewing time. The model showed self-similarity as evidenced by the variance-time plot. The browser "Mosaic" was modified to record user activities in the client. The instrumented browser was distributed to 37 clients in Boston University. This represents a relatively small cross-section of users, and because the browser on the client side is modified it is difficult to measure other user communities. Further, the configured HTTP version was 0.9, which was used before 1996. Because Mosaic is no longer evolving, this work can not be extended to current versions of HTTP. The code would need to be modified for a number of popular browsers in order to record representative traces. They did not examine how closely SURGE imitated a stream of HTTP requests.

Deng [9] measured Web traffic characteristics of individual subscribers and proposed a two-state ON/OFF model for the arrival process at the access link. The model did not simulate the detailed interaction of Web traffic and the accuracy of the model was not evaluated.

3 Overview

There are four different versions of HTTP currently being available. In what might be called "pure" Version 1.0 [1] objects are downloaded back-to-back with each object requiring one TCP connection. In Version 1.0 with multiple connections, the browser opens multiple parallel connections to download objects for the earliest display of the page. The browser sets the limit on the number of multiple connections. Objects beyond these limits are downloaded after completing one of the outstanding connections. In Version 1.0 with "Keep-alive", multiple connections are possible, but a connection is not closed immediately on the chance that a new request for the connection will arrive before a time-out. Version 1.1 [2] permits persistent connections and requests are pipelined. The persistent connection is very similar to the Keep-alive connection, the exception being for a proxy.

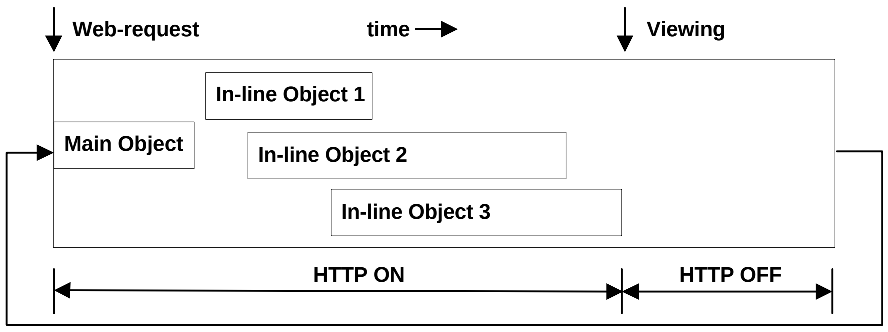

Figure 1: Overview of the basic model of the browser-server interaction

A typical Web page consists of a Hypertext document. The Hypertext document is an HTML-coded text with links to other objects that make up the whole page. An *object* is an entity stored on a server as a file. There are two kinds of objects, a main object and an in-line object. The file containing an HTML document is a *main object* and the objects linked from the Hypertext document are *in-line objects*.

The basic model of Web traffic is shown in **Figure 1**. A new Web-request is immediately generated after expiration of the viewing period. The model simulates an ON/OFF source where the ON state represents the activity of a Web-request and the OFF state represents a silent time after all objects in a Web-request are retrieved. The duration of On state and Off state correspond to On-time and viewing time, respectively. Viewing time does not necessarily mean the time a user spends viewing the page; it denotes any time that the browser is inactive. Viewing time would include, for instance, the situation where the browser is iconized while a user is working with another application. On-time is the time taken to fetch all objects in a Web-request. The ON state can be split into successive TCP connections used to deliver objects. The parallel connections for in-line objects are opened consecutively after the single connection for the main object. The duration of the connection depends on the size of objects.

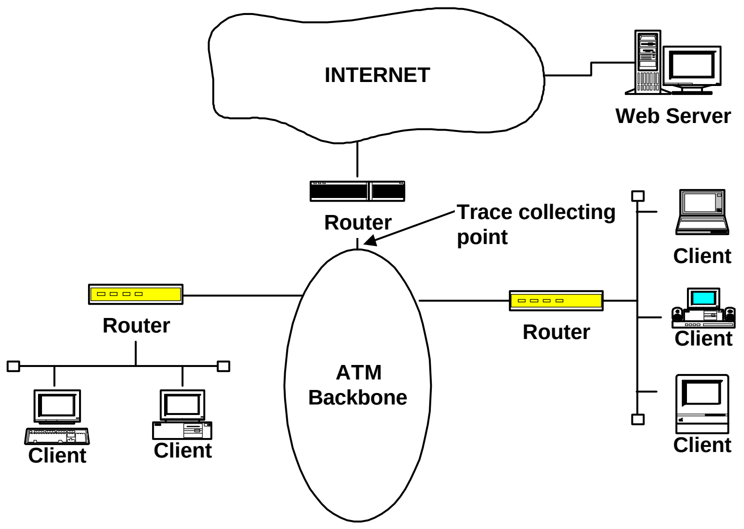

Figure 2: Perspective of the campus network.

4 Measurement and Analysis

4.1 Traffic Measurement and Parsing

We would like to measure the traffic close to a browser source because we are modeling the traffic sent by, and to, the browser. At the same time, we want a large cross-section of traffic. We meet these objectives by recording a trace of traffic on the backbone network of the Georgia Tech campus (see **Figure 2**). The Georgia Tech campus network is composed of two B-class IP addresses (130.207 and 128.61) and a number of C-class IP addresses (about 170). We are able to record a large cross-section of Web interaction from the campus where more than 1800 users participated in one-hour long trace. We further filter the traffic to gather only those sessions that originate on the campus and terminate elsewhere in the network. In this way we expect to obtain model parameters that are not unduly affected by the behavior of campus servers. Further, we exclude machine-generated traffic such as Web-crawler traffic. Traffic from major search engines is filtered out because it originates from outside of the campus. The traffic from the campus search engines is not recorded because it terminates inside the campus.

Date	Number of Clients	Number of Web-requests	Methods			HTTP Status Codes				
			GET	POST	HEAD	1xx	2xx	3xx	4xx	5xx
10/7/98	1934	24014	448810	5811	320	197	138948	39903	1594	2488

Table 1: Summary of the trace of October

We use primarily a trace that was collected from 11 A.M. to 12 P.M. on Wednesday October 7 1998 running on a Sun Ultrasparc2 (180 MHz) workstation. More than 1900 clients participated in Web browsing sessions producing about 24,000 Web-requests. The details of the trace are listed in **Table 1**. We regularly record traces on the campus backbone and this particular trace is typical. We do not claim that our traffic source is representative of Web traffic in general. However, we are able to develop and evaluate a methodology that can be extended to other traffic source.

We use two tools to collect and parse data, Tcpdump [5] and Tcpshow [6]. The binary-mode option was set while Tcpdump was running. Tcpdump recorded TCP/IP headers as well as 300-byte

TCP payloads. The 300 bytes of the TCP payload are large enough to capture the HTTP request header and the HTTP response header within the range of interest. For our study, the HTTP header is important for obtaining additional information about Web traffic. The HTTP header information is used in the parsing and analysis phase to separate Web-requests, to decide if the connection is Keep-alive connection or Close connection and to check if the object is cached.

Parameter	Description
Object Size	The size of objects (main and in-line) stored on the remote server
Request Size	The size of HTTP header sent when requesting URIs
Number of In-line objects	The number of embedded objects in a page which requests are made
Viewing (Off) Time	User thinking time
Number of (Non-)Cached Pages	The number of consecutive pages that are (not) locally cached in the browser.
Parsing Time	The time for a browser to parse the HTML code
In-line Inter-arrival Time	Inter-arrival time of connections for in-line objects
Stall Time	The time spent in a stall
Number of segments in a stall	The number of TCP segments in a stall

Table 2: List of parameters modeled

Tcpshow interprets a binary-mode trace. From the binary-mode trace, two new traces are created in off-line processing. One contains the HTTP header information (HTTP trace). The other contains the TCP header information (TCP trace). The parameters (see **Table 2**) are categorized into the HTTP layer and the TCP layer depending upon associated traces. HTTP-layer parameters are searched in the HTTP trace and TCP-layer parameters are searched in the TCP trace.

The parser script, written in *Practical Extraction and Report Language* (PERL), sorts out the HTTP trace by the client IP addresses and checks if the client is inside of the campus. The sorted trace for a single client is separated into distinct Web-requests using the technique discussed in Section 4.2. The start and end times of the Web-requests are then recorded and used to parse parameters in the TCP trace. Empirical distributions of HTTP-layer parameters are obtained. In the TCP trace, the parser script parses TCP-layer parameters based on the boundaries that were recorded when HTTP-layer parameters were parsed.

Once empirical distributions of the individual parameters are obtained, we compare each distribution with different standard probability distributions and select the best fit. The *Quantile-*

Quantile plot (Q-Q plot) [13] is used to test the fit of the data to the model. If the model fits the data perfectly then the plotted points lie on a straight line. The best standard probability distribution is determined to be the one that minimizes the root-mean-square of the deviation from a straight line. We select the best distribution from among Weibull, Lognormal, Gamma, Chi-square, Pareto and Exponential (Geometric) distributions.

4.2 Web-request

A Web-request is a page or a set of pages resulting from a request by a user. By definition, (1) a Web-request is initiated by a human action and (2) the first object in a Web-request is an HTML document. Accurate identification of Web-requests is essential for our model to be accurate. We determine the Web-requests by setting up rules for a request. We apply these rules to our parsing of the trace to identify Web-requests. Our rules are:

- For simultaneously requested multiple pages by a user from multiple browsers but from the same client, each request represents a Web-request by definition (1). In this case, the second or later Web-requests might include an object belonging to a page of previous Web-requests. If the Web page were a basic unit of the model, this inclusion would skew the model parameters. It does not matter if the basic unit is a Web-request because the boundary is determined by whether or not we have a user request.
- If a single request generates multiple pages (*e.g.* frame and Java-script), they belong to the same Web-request. However if subsequent pages are retrieved by a user from within the frame they would represent a complete Web-request by definition (1). We do not consider an automatically redirected page (HTTP status code 301 and 302) to be a Web-request.
- If a user clicks a hyperlink of a single object[1] such as an image (.jpg, .gif), a sound (.avi or .mp3) or a text document (.txt, .ps or .pdf), these single objects do not represent a Web-request by themselves (2). That's because the first object is other than an HTML document. Instead, they are

[1] It is difficult to distinguish a hyperlinked single object from a regular in-line object because their trace records look the same. We could have used one of HTTP header information, “referal”, to distinguish these objects. Because “referal” does not always guarantee to distinguish these two cases, we decided not to use.

included in a Web-request as in-line objects.

In order to determine a boundary of a Web-request, we take advantage of information in the HTTP header by inspecting the extension of the requested objects, the MIME type of the response or both. The extension indirectly implies the contents of an object. Objects are mostly named according to their type; for instance, most graphical images are named ".gif" or ".jpg" and the most popular extension for an HTML document is ".htm[l]". This tendency is further enforced if the page is designed by Web publishing utilities[2]. **Table 3** shows the list of extensions of HTML documents found in our traces. **Other** in the last column of **Table 3** is mainly due to query requests whose URIs are *WWW-URL-encoded*.

Extensions of objects do not always correlate with the contents of objects. To prevent inadvertently missing a Web-request by relying on the extension, the MIME type in the HTTP response header is also checked. The MIME type directly implies the type of an object. "text/html" is the reserved MIME type for an HTML document by *Internet Assigned Number Authority* (IANA).

In summary, a request becomes a Web-request:

- If it is used to request for an object whose extension contains either ".htm", ".asp" or "cgi". A URI that finishes with "/" implies "index.html" in the directory, also becomes a Web-request.
- If it results in a response of MIME type "text/html" and the HTTP status code 200 (OK).

Total	htm	/	cgi	asp	sml	Stm	other
28833	12340	4515	4425	1231	59	58	6205

Table 3: List of HTML document extensions.

5 Model

From our traces and their analysis we have derived the parameters of our model. In the following we indicate a parameter by bold face.

5.1 HTTP Model

Statistics of HTTP parameters are shown in **Table 4**. **Number of in-line objects** is the number of

[2] Microsoft's Frontpage 98, Adobe's Pagemill and Netscape's Composer . They use ".htm" as the default extension.

in-line objects in a Web-request. The in-line objects that we count are only those which need to be downloaded. A requested object does not need to be downloaded if it is found in the cache with valid time-stamps. Hence **number of in-line objects** is always less than or equal to the total number of objects. The mean **number of in-line objects** is 5.55, almost three times larger than the mean observed in [8]. This may be explained by the following:

- The number of multimedia objects in a page is increasing with time, as pages become more complex.
- The hit ratio of the local cache has dropped. Because the number of Web servers has increased since the work [8] was done, the number of Web servers that a user accesses has also increased, so that a user's access pattern becomes less predictable.
- The technique that was used to separate pages in [8] underestimated the parameter as discussed in Section 4.2.

The distribution of **number of in-line objects** (see **Figure 5.a)** matches a Gamma distribution. Its histogram is shown in **Figure 5.b**.

Viewing time is the inactive interval between Web-requests. The viewing time parameter measured in previous work [8], by definition, was always greater than one second. However our histogram of **viewing time** in **Figure 6.b** shows a large number of values less than one second. These samples may be attributed to situations where a user abandons the Web-request before the browser has finished fetching all of its objects and where Web-requests are requested from multiple browsers. The distribution of **viewing time** is fitted well by a heavy-tailed Weibull distribution (see **Figure 6.a)**, which is consistent with previous work [8][9][10].

In-line inter-arrival time is the time between the opening of one in-line object and the next. It measures the starting time between subsequent in-line objects. The inter-arrival time, up to the maximum permitted number of simultaneous objects, is just a few tens of milli seconds. Further in-lines are sent only after outstanding objects complete, so that inter-arrival times may be as large as a few seconds or more. The distribution of **in-line inter-arrival time** matches a Gamma distribution.

Parsing time is the time spent parsing the HTML code in order to determine the layout of a page after fetching the main object. This quantity depends on the client machine. It is well matched by a Gamma distribution.

Both distributions of **main-object size** (see **Figure 7.a**) and **in-line-object size** are well fitted by a Lognormal distribution. The mean of **main-object size** is larger than **in-line-object size** and the variance of **in-line-object size** is greater than **main-object size**. The HTTP object size is easily obtained from the *content-length* field in the HTTP response header. The histogram of **main-object size** is shown in **Figure 7.b**.

Request size is the size of the HTTP request header. It is best fitted by a Lognormal distribution.

5.2 TCP model

The performance of the Web depends critically on TCP[3] and less on the *Internet Protocol* (IP) and the lower layers. The two most popular versions of HTTP use the TCP connection in different ways. HTTP 1.0 with multiple connections always closes a connection after an object is retrieved. HTTP 1.0 with Keep-alive connection holds the connection open between retrieval of consecutive objects by using a "Keep-alive" mechanism. Our model assumes that TCP is closed after the retrieval of each object. The model is easily extended to incorporate a Keep-alive or persistent TCP model.

A "classical" slow-start model does not fit the measured data [15]. In our model, a TCP connection consists of a number of stalls. In TCP, segments are transmitted in a burst until the TCP window is closed. The interval between two successive bursts is called a stall. We use a TCP model in which the number of segments transmitted and the time spent in a stall are chosen from an empirically derived distributions.

The statistics of the TCP parameters are shown in **Table 5**. The distribution of **number of segments in a stall** changes little after the sixth stall. **Number of segments in a stall** after the sixth stall

[3] We do not explicitly model TCP transmission errors. However, the effect of transmission errors is indirectly incorporated into our model through the measured values of the parameters.

uses the same statistics as the sixth stall. Similarly, **stall time** changes little after the seventh stall. **Stall time** after the seventh stall uses the same statistics as the seventh stall. We observed the number of segments more than six in the stall greater than four. We truncated the distributions above six segments in a stall because the portion of more than six was negligible.

5.3 Web Caching Model

In HTTP, cacheable objects are stored with a tag containing the expiration-time; after this time an object is no longer valid [1][2]. The expiration-time is estimated from the *last-modified* field in the HTTP response header. The longer the time since the object was modified, the longer the expiration-time. Before HTTP sends a request, it checks the local cache. If the requested object is found in a local cache, HTTP checks the expiration-time. If the object is valid, HTTP reads the object from the cache instead of retrieving it from a remote server. If not, HTTP sends a conditional request with the expiration-time tag in the *if-modified-since* field. The server compares the tag with the time the object was modified. The server responds with the HTTP status code 304 if it is still valid. If not, the server downloads the object.

We model two types of events. One is when the expiration-timer of the cached object is expired and the server proves the validation of the cache. The other is when the cached object is confirmed as expired by server or when the object is not cached; in both cases an object is downloaded from the server. We do not model the case where the object in the cache is valid because such cases do not generate network traffic.

We can easily determine a cached object from the HTTP status code in the HTTP trace. It is a cached object if the HTTP status code is 304. In a Web-request, some objects can be cached and some objects are not. The Web-request is defined to be a cached Web-request if more than a half of the objects in the Web-request are cached. The reasoning is that two Web-requests share an object with low probability. Further, if the first object in a Web-request is cached, all following objects tend to be cached. **Figure 9** shows the histogram of the fraction of non-cached objects in a Web-request. Two peaks at zero and one indicate whole objects are either cached or non-cached. Based upon the definition of a cached Web-request, we measure **number of consecutive cached Web-requests** and **number of consecutive non-cached Web-requests** from the trace.

Our Web caching model is a *two-state renewal process* where a Web-request is an event. A renewal process remains in one state for a number of self-returning events and moves to next states at the time of a renewal. This model has two states; one state represents a Web-request that is locally cached (Cached state) and the other represents a normal Web-request (Non-cached state). We have extracted the distributions of the number of self-returning events of the two states; **number of consecutive cached Web-requests** and **number of consecutive non-cached Web-requests**. The model remains in the Cached (or Non-cached) state until the number of Web-requests generated in this state is the same as the number of Web-requests obtained from the distribution. Then, the model switches to the Non-cached (or Cached) state with probability one.

Number of consecutive cached Web-requests is best fitted by a Geometric distribution and **number of consecutive non-cached Web-requests** is best fitted by a Lognormal distribution (see **Figure 8.a**). The histogram of **number of consecutive non-cached Web-requests** is shown in **Figure 8.b.**

Parameters		**Mean**	**Median**	**S.D.**	**Best-fit**
Request size		360.4 bytes	344 bytes	106.52	Lognormal
Object size	Main	10709.8 bytes	6094 bytes	25032.1	Lognormal
	In-line	7757.74 bytes	1931 bytes	126168	Lognormal
Parsing time		0.132 sec	0.056 sec	0.187	Gamma
Number of In-line objects		5.55	2	11.35	Gamma
In-line Inter-Arrival time		0.86 sec	0.17 sec	2.15	Gamma
Viewing (OFF) time		39.45 sec	11.71 sec	92.57	Weibull
Number of Web-requests	Non-cached	12.58	5	21.623	Lognormal
	Cached	1.74	1	1.74	Geometric

Table 4: Summary statistics for HTTP parameters

Parameters \ Segment Number		**1**	**2**	**3**	**4**	**5**	**6**
Number of Segments in stalls (%)	1st stall	66	34				
	2nd stall	17	50	33			
	3rd stall	14	29	43	14		
	4th stall	17	21	17	11	17	11
	5th stall	21	21	21	11	11	15
	6th stall	29	24	18	10	7	12

Parameter		**Mean**	**S.D.**	**Best-fit**
Stall Time	1st stall	0.317 secs	0.640	Inverse Transform
	2nd stall	0.283 secs	0.685	Inverse Transform

	3rd stall	0.296 secs	0.753	Inverse Transform
	4th stall	0.335 secs	0.863	Inverse Transform
	5th stall	0.376 secs	0.882	Inverse Transform
	6th stall	0.396 secs	0.858	Inverse Transform
	7th stall	0.438 secs	0.976	Inverse Transform

Table 5: Summary statistics for TCP parameters

5.4 Correlation

Parameters in our model are assumed to be uncorrelated with themselves as well as with one another so that the generation of one parameter is independent of the generation of other parameters. In order to test the accuracy of this assumption we calculated auto and cross correlation functions of our parameters. We selected only client sessions which had more that 40 samples of Web-requests. We calculated the correlation function up to the lag of 20 from these samples. We find that, generally, correlations are very low. An exception is the significant amount of auto-correlation in **request size** (see **Figure 10)** where the average auto-correlation at lag one is 0.59. Note that **request size** is far smaller than either **main-object size** and **in-line-object size**. Consequently we would expect the auto-correlation of **request size** to not significantly affect the accuracy of the model. **Table 6** shows the average auto-correlations at lag one.

Request size	Viewing time	Number of in-line objects	Parsing time	In-line inter-arrival time	Main-object size	In-line-object size
0.59	0.23	0.18	0.07	0.05	0.16	0.13

Table 6: Auto-correlations at lag one

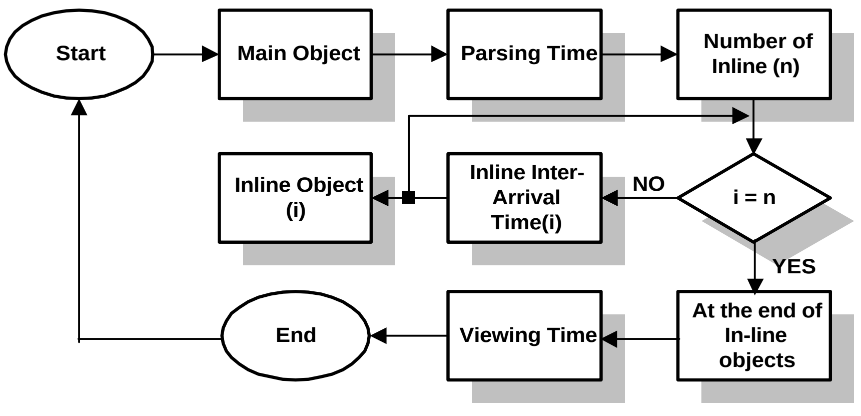

Figure 3: State transition diagram for Web traffic generation

6 Traffic Generation

The traffic model simulates an ON/OFF source. The state transition diagram shown in **Figure 3** governs the flow of the traffic. At the beginning, the traffic corresponding to the main object is generated and is delayed for the period of **parsing time**. During this period a Web browser fetches the main object and parses **number of in-line objects** as well as the page layout. The model, however, generates **number of in-line objects** from the best-fit distribution and waits for the expiration of **parsing time**.

After the start of one in-line object there is a delay to the start of the next. The first in-line object starts after expiration of **parsing time**. The second in-line object does not wait until the first in-line object finishes but starts one **in-line inter-arrival time** after the start of the first. Subsequent in-line objects start until the number of in-lines started equals **number of in-line objects**. The number of outstanding connections in the model is not restricted to four or six which is the case of Netscape and an Internet Explorer. Instead we model the number of outstanding connections by the distributions that are collected from the trace. In the model, depending upon **in-line object size** and **in-line inter-arrival time**, the number of outstanding connections will vary. Frequently, **in-line inter-arrival time** is less than the duration of the connection, which is mainly determined by **in-line object size.** Hence, the model indirectly simulates the parallel downloading of in-line objects. After all objects are transmitted the model is silent for **viewing time**. After the expiration of **viewing time**, the model starts to generate a new Web-request.

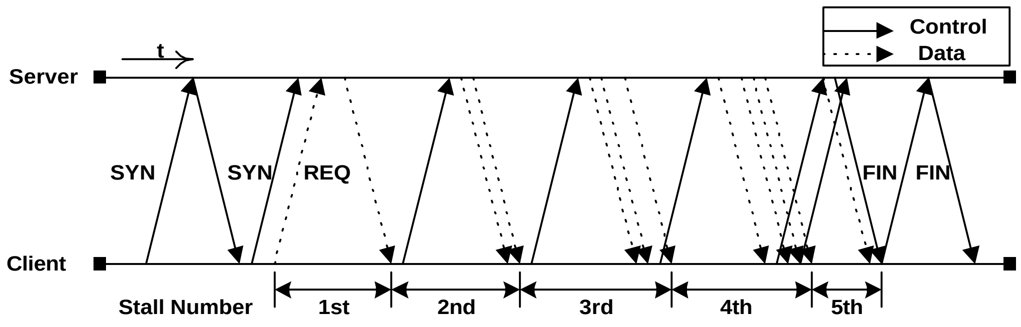

Figure 4: TCP model

In our TCP/IP model, the duration of a connection consists of a number of TCP stall times and two three-way handshaking periods (SYN and FIN). An object size is divided into a number of *Maximum Segment Sizes* (MSS) of 1460 bytes which will become 1500 bytes by adding a 40 byte TCP/IP header. At the beginning, a client synchronizes with a server by exchanging SYN packets and then sends a request packet. A server downloads TCP segments stall by stall where **stall time** and **number of segments in a stall** are generated from distributions. Inside of a stall, a client acknowledges every other data segment. A client and a server close a connection by exchanging FIN packets after downloading a whole object. **Figure 4** shows our TCP model. Our TCP model is described in more detail elsewhere [18].

The Web caching model influences the final model on **main object size** and **in-line-object size**. The HTTP object size becomes zero except for the main object while in the cached state. Because the expiration-time of main objects is relatively short due to frequent changes, the main object is fetched most of time. While in the non-cached state, both HTTP object sizes are generated from the distribution.

7 Validation of the Model

To validate the model we need measurements from the trace that are independent of any measurements used in constructing the model. Two such measurements are **on-time** and the variation of the demanded bandwidth in time.

7.1 On-time

On-time is not directly used in constructing our model. **On-time** is the function of a number of parameters; **number of in-line objects**, **in-line inter-arrival time**, **main-object size**, **in-line-object size** and **stall time**. These parameters interact with one another and combine to determine **on-time** in the model. Thus, it may be used to check the model by comparing with directly measured **on-time** from the trace. If there is any significant error in our measurements of parameters, the way we combine measurements or the completeness of our model we would expect differences between the model and the trace.

On-time from both the model and the trace match a Weibull distribution with the shape parameters 0.77 and 0.68, respectively. The mean and standard deviation of the traces are 11.34 and 23.85 and those of the model are 10.49 and 20.33. The CDF comparison is shown in **Figure 11**. The similarity between the model and the trace parameters indicates that our model closely mimics the real traffic in terms of the total delay experienced in retrieving a Web-request.

7.2 Bandwidth Demanded

The variation of the demanded bandwidth in time is another independent check. To implement this test, we have recorded the sum of bytes in ten-milli second granularity from the trace and the model. To obtain the value at the next level of the granularity we summed ten consecutive samples from the previous granularity and calculated the arithmetic mean of the samples. We measured four different granularities up to a ten-second granularity and plotted 200 samples.

To obtain the same observation as the trace, we need to assume the starting and ending times of a client. The starting time is the delay from the time a trace starts to the time a client starts transmitting the first segment. The ending time is the delay from the last packet of a client seen in the trace to the end of the trace. A client in the model randomly selects the pair of starting and ending delays. A client starts transmitting the first segment after the starting time. A client stops transmitting if the simulation time left is less than the ending time when a new Web-request begins. We ran the model with the same total number of clients as the trace and observed the same parameter. **Figures 13.a, 13.b** show the result. The horizontal solid line in the figure indicates the mean of the samples.

Particularly, in this section, we are more interested in the largest granularity of 10 seconds to obtain the larger picture of the comparison. The means of bytes transmitted of the model and the trace closely match as shown in **Figure 12**. The mean of the trace is 4656 kbytes and that of the model is 4699 kbytes.

7.3 Self-similarity

The process *X* is called self-similar if the aggregate process of *X* has the same auto-correlation function as *X*. The degree of the self-similarity is expressed using the *Hurst* parameter. The *Hurst* parameter is always less than 1.0 and greater than 0.5. The closer the *Hurst* parameter to 1, the more self-similar the process. We have tested the self-similarity of the model using two methods: *variance-time plot* and *R/S* (Rescaled Adjust) *plot*. We have also tested the burstiness of the traffic using four different time scales. Further discussion about self-similarity can be found in [14][17]. For the next three tests, we have used the same data obtained in the previous section.

Burstiness Test

In self-similar traffic burstiness remains regardless of the level of the aggregation because of the infinite variance of the source. One way to observe this effect is by visually inspecting the time-series plot of such traffic with varying the levels of aggregation. **Figures 13.a, 13.b** exhibit significant burstiness at the four-different granularities.

Variance-time plot

Two *variance-time plots* shown in **Figures 14.a, 14.b** exhibit the level of the self-similarity of the model and the trace. The *Hurst* parameters calculated are 0.805 for the trace and 0.78 for the model.

R/S plot

The *R/S plot* shows that the asymptotic slope is different from 0.5 and 1.0 (see **Figures 15.a, 15.b)**. The estimated *Hurst* parameters are 0.8 for the trace and 0.77 for the model respectively. The group of values near one on the x-axis fall below a slope less than half. That is mainly due to the small number of samples.

8 Discussions

Burstiness test

By comparing **Figures 13.a** and **13.b**, we observe that the model is a little less bursty than the trace in the ten-second-granularity figure. The lower burstiness of the model may be due to the large size of an object with a large advertised window size. In the trace, the window size (or the number of segments) continually increases (unless packet loss) as the number of stalls increases. If the advertised window size is quite large, the window size can be increased beyond six, which is the maximum number of segments in a stall in the model. Hence, the throughput of the model becomes lower than the trace and stalls for large objects in the model are more widely spread in time resulting in smoother traffic. This effect is not observed in lower-granularity figures.

Keep-alive and persistent connections

We collected data from Keep-alive and persistent connections. About 76 percent of connections in the trace are Keep-alive enabled connections. About 40 percent of these connections actually use Keep-alive and the rest are closed; 1) after the expiration of the Keep-alive timer, 2) the user switched to another Web server, or 3) the server closed the connection due to the nature of a page. Most servers have a short period for the Keep-alive timer due to limited resources at the servers.

9 Conclusions

Web traffic has more structure to it than most general types of Internet traffic. Further, the characteristics of the traffic change as browsers and servers evolve, as the behavior of users changes, as the speed of network increases and as the protocols change. This makes Web traffic modeling a challenge.

We have presented a model of Web traffic that attempts to capture the major aspects of the traffic. This has required us to extend the models used in previous work and to correct shortcomings of previous models. For example, the presence of a frame has required us to redefine the basic unit of the model.

From the extensive traces we have gathered of Web traffic, we have extracted the distributions of the parameters used to represent the model. From the model, we have generated synthetic traffic and can now compare it with the trace model. We have shown that independent variables from the trace and from the model agree well. We believe that the model is accurate enough to predict the behavior of traffic as parameters change (such as the speed of a line or the size of Web-requests), however, this remains to be proved.

10 References

[1] T. Berners-Lee, R. Fielding, and H. Frustuk. *Hypertext Transfer Protocol – HTTP/1.0.* Internet RFC 1945, May 1996.

[2] R. Fielding, J. Getty, J. Mogul, H. Frystyk, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1.* Internet RFC 2068, Jul 1997.

[3] World Wide Web Consortium. *HTML 4.0 Specification.* W3C Recommendation, Apr 1998.

[4] K. Thompson, G. J. Miller, and R. Wilder. Wide-Area Internet Traffic Patterns and Characteristics (Extended Version). *IEEE Network Magazine,* Nov 1997.

[5] V. Jacobson, C. Leres, and S. McCanne. *Tcpdump manual and software*.

[6] M. Ryan. *Tcpshow version 1.73* available at http://http.cs.berkeley.edu/~daw/mike/, Nov, 1996.

[7] H. F. Neilson, *et al*. Network Performance Effects of HTTP/1.1, CSS1, and PNG. In *Proceedings of SIGCOMM '97*, Cannes, French Riviera, France, Sep 1997.

[8] B. Mah. An empirical model of HTTP network traffic. In *Proceedings of INFOCOM '97*, Kobe, Japan, Apr 1997.

[9] S. Deng. Empirical model of WWW document arrivals at access link. In *Proceedings of ICC '96*, Jun 1996.

[10] M. Crovella and A. Bestavros. Self-Similarity in World Wide Web traffic, Evidence and Possible cause. In *Proceeding of ACM SIGMETRICS '96*, Philadelphia, PA, Apr, 1996.

[11] V. Almeida, *et al*. Characterizing reference locality in the WWW. In *Proceedings of International Conference on Parallel and Distributed Information Systems (PDIS) '96*, Dec 1996.

[12] P. Badford and M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98.*

[13] R. B. D'Agostino and M. A. Stephens. *Goodness-of-Fit Techniques*, Marcel Dekker, Inc., 1986.

[14] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transaction on Networking*, pages 1-15, 1994.

[15] J. Heidemann, K. Obraczja, and J. Touch. Modeling the performance of HTTP over several transport protocols. *IEEE/ACM Transaction on Networking*, 616-630, Aug 1997.

[16] C. A. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW client-based trace. *Technical Report TR-95-010*, Boston University, Dept. of Computer science, Apr 1995.

[17] J. Beran. Statistics for long-memory processes. Monographs on Statistics and Applied Probability. Chapman and Hall, New York, NY, 1994.

[18] Hyoung-Kee Choi and John O. Limb. An Empirical Model of TCP. In preparation.

[19] Hyoung-Kee Choi, Osama Quadan, Dolors Sala, John O. Limb and Jeff Meyers. Interactive Web service via satellite to the home. Submitted for publication.

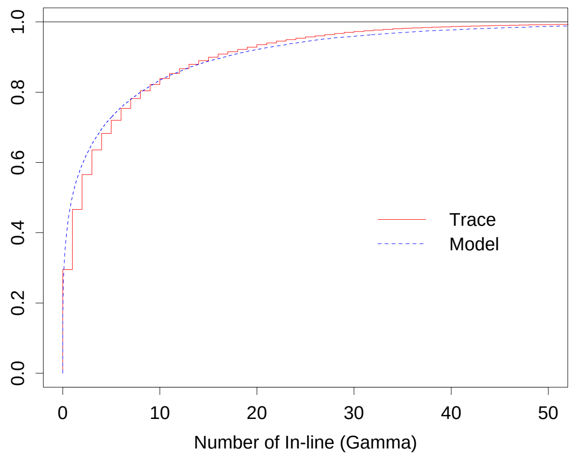

Figure 5.a: CDF comparison of number of in-line objects with a Gamma distribution

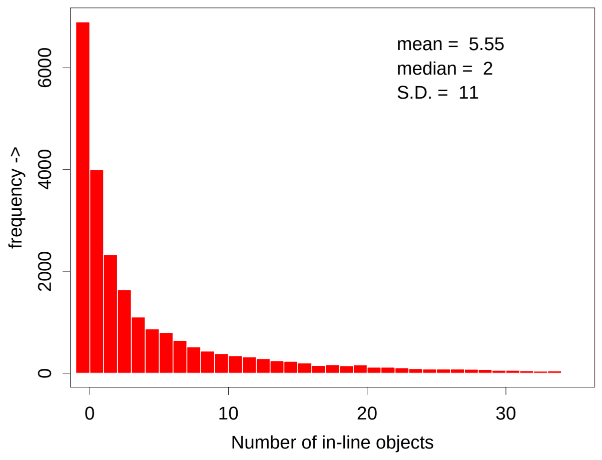

Figure 5.a: Histogram of number of in-line objects

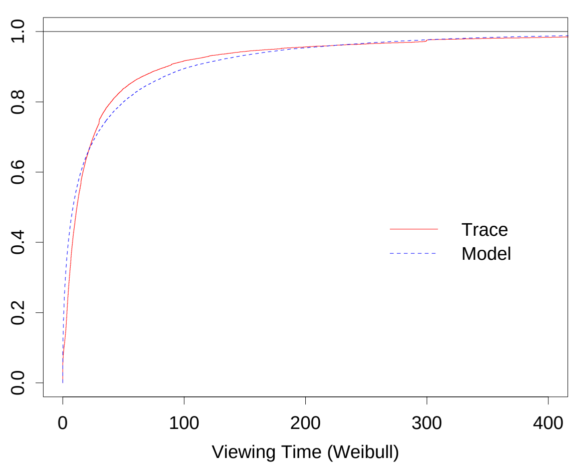

Figure 6.a: CDF comparison of viewing time with a Weibull distribution

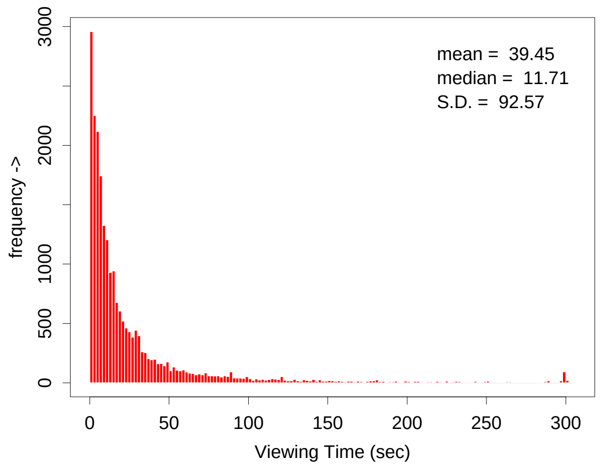

Figure 6.b: Histogram of viewing time. The peak near 300 sec are due to periodically refreshed pages.

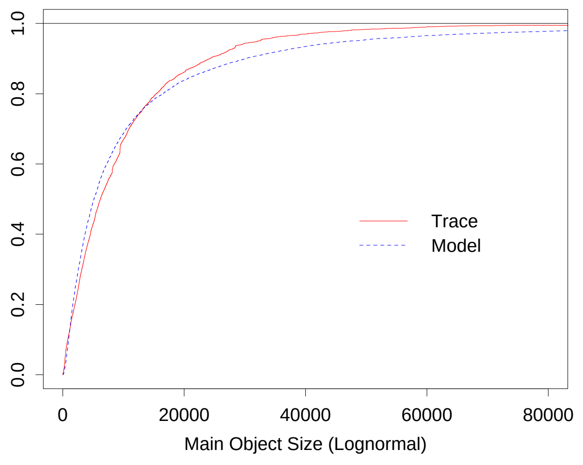

Figure 7.a: CDF comparison of main-object size with a Lognormal distribution

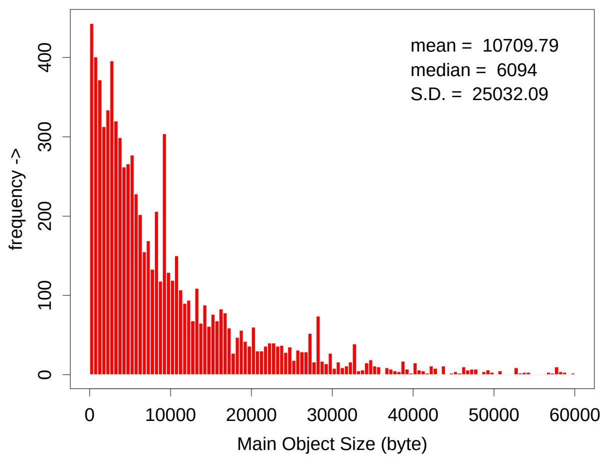

Figure 7.b: Histogram of main-object size

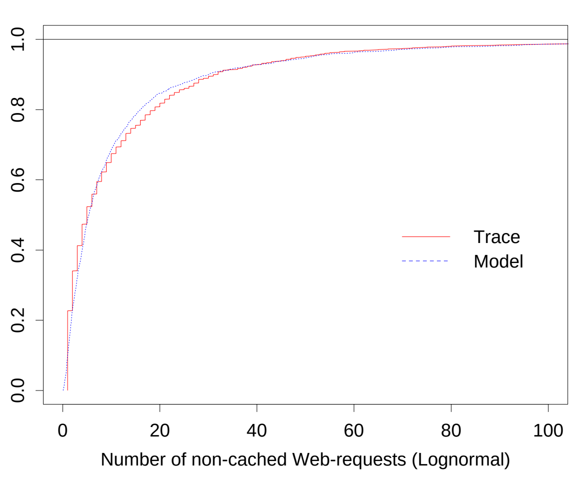

Figure 8.a: CDF comparison of number of non-cached Web-requests with a Lognormal distribution

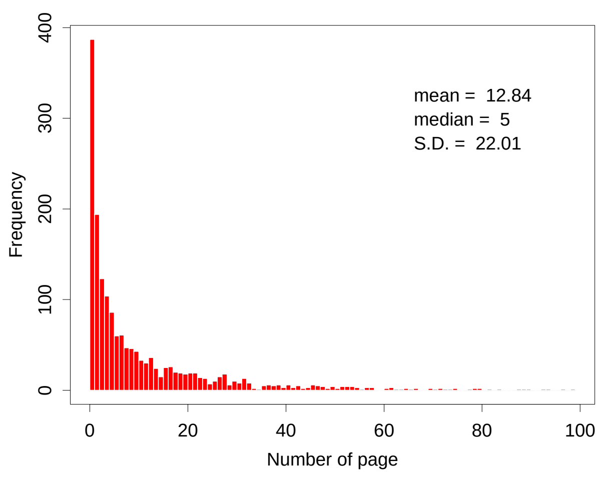

Figure 8.b: Histogram of number of non-cached Web-requests

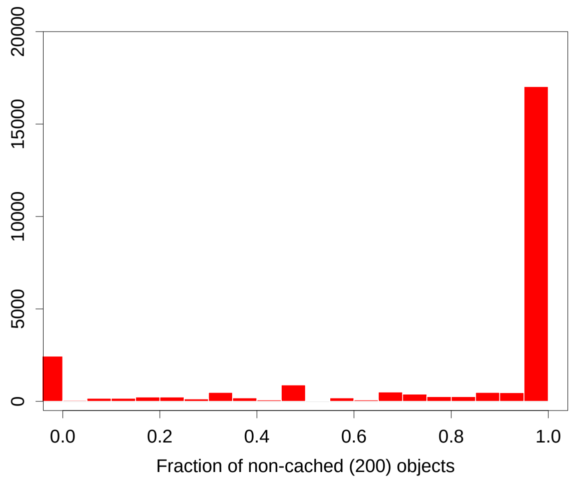

Figure 9: Histogram of the fraction of non-cached objects. A sample is calculated by dividing the number of non-cached objects by number of in-line objects in a Web-request.

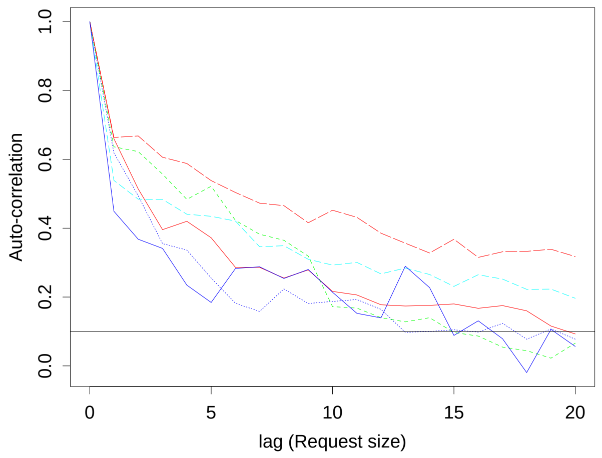

Figure 10: Auto-correlation function of request size. Six different samples are plotted. The mean of auto-correlation at lag one is 0.59.

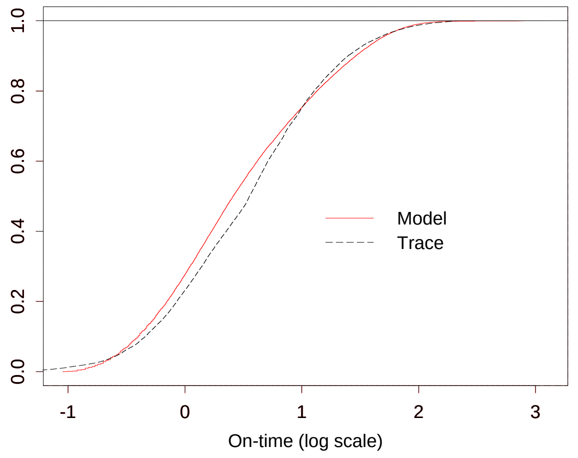

Figure 11: CDF comparison of On-times - On-time of Trace and On-time of Model. X-axis is log-scaled

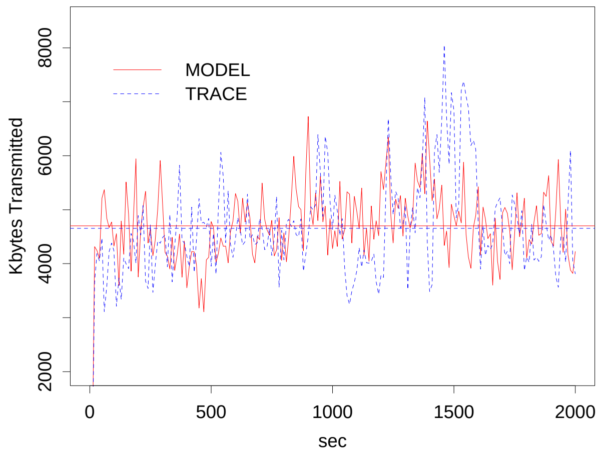

Figure 12: The variation of the number of bytes in time. Two parallel lines indicate the mean of samples.

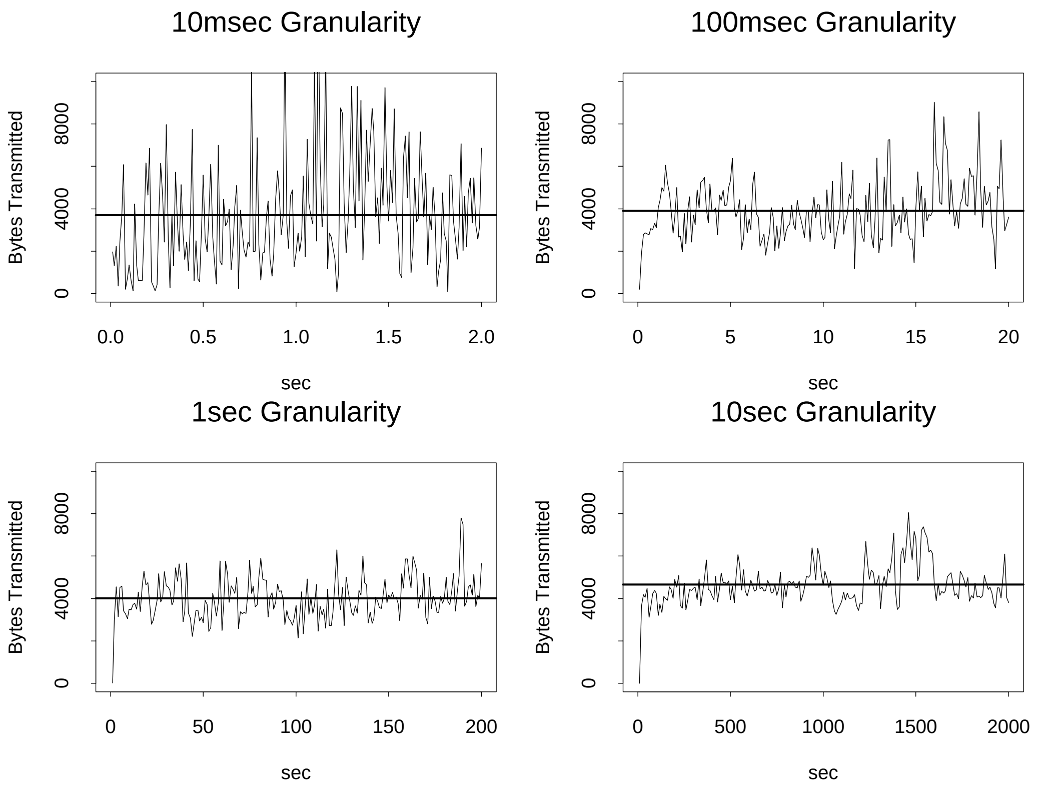

Figure 13.a: Burstiness test for the trace. The next granularity plot is calculated by averaging ten consecutive samples. Each plot shows 200 samples. The solid line in the figure indicates the mean of the samples.

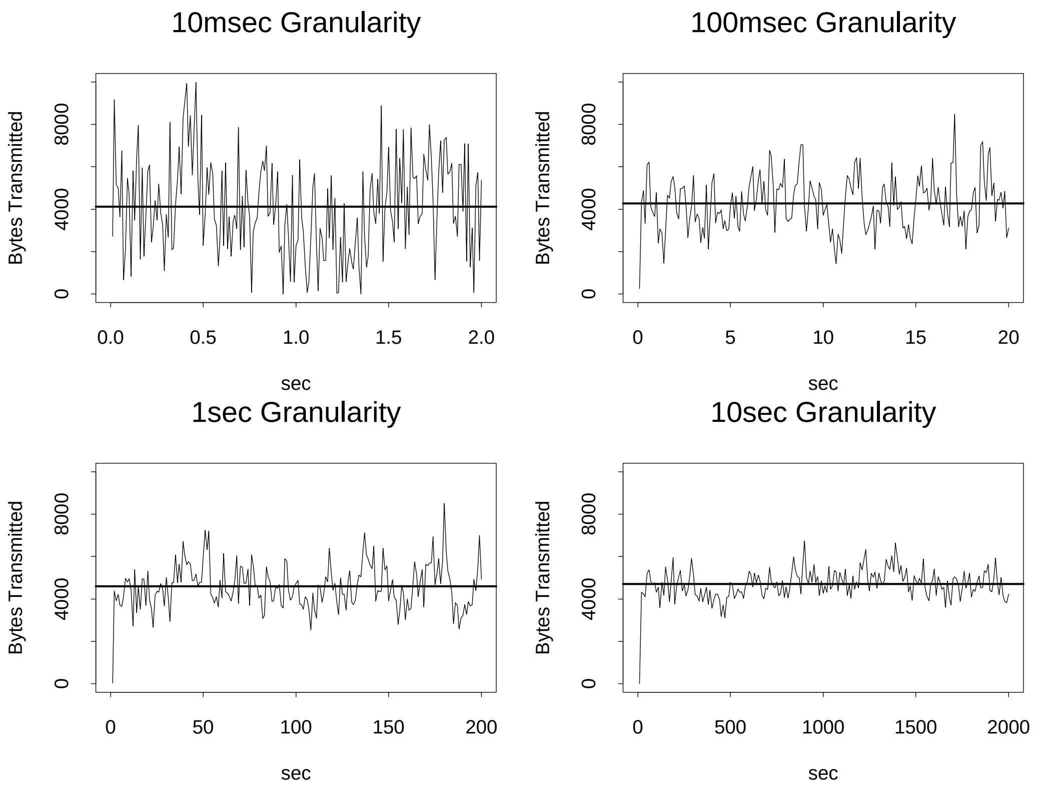

Figure 13.b: Burstiness test of the model. The less burstiness of the model in the 10-second granularity is attributed to the TCP model.

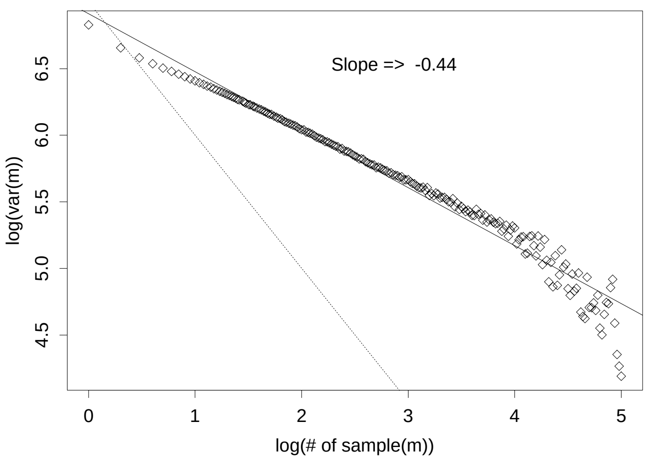

Figure 14.a: Variance-time plot of the model. The Hurst parameter is 0.78. The slope of lower line is -1.

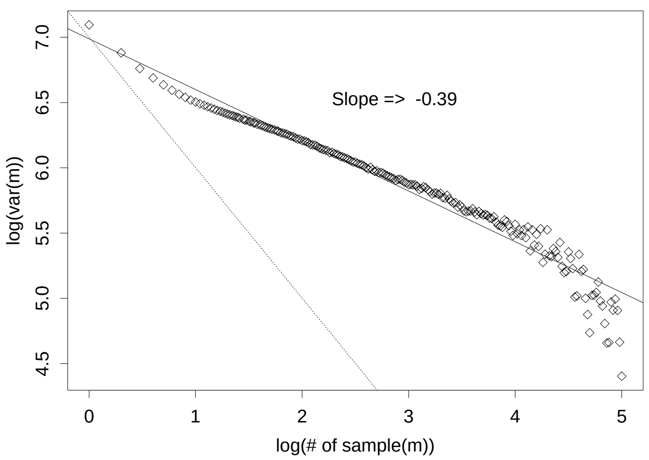

Figure 14.b: Variance-time plot of the trace. The Hurst parameter is 0.805.

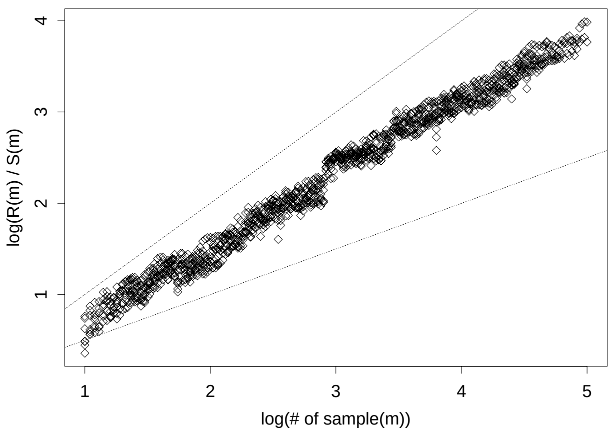

Figure 15.a: R/S plot of the model. The Hurst parameter is 0.77. The slopes of two straight lines are 1 and 0.5, respectively.

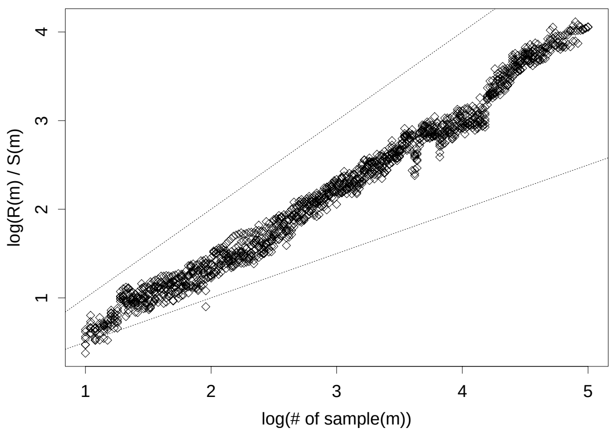

Figure 15.b: R/S plot of the trace. The Hurst parameter is 0.8.