SAX: A Tool for Studying Congestion-induced Surfer Behavior

Dinh Nguyen Tran, Wei Tsang Ooi, and Y.C. Tay

National University of Singapore

Abstract. User reaction to traffic congestion can have severe impact on network stability and significant implication for traffic engineering. For example, users who persist in large peer-to-peer transfers despite congestion can drive the network into congestion collapse. On the other hand, users who abort large transfers can smoothen self-similar traffic. We present a tool, called SAX, for studying congestion-induced behavior of web surfers. SAX extracts information from HTTP packet traces and infers clicks, abortions and sessions. Measurements with SAX show how users back-off when congestion occurs.

1 Introduction

Despite exponential growth of Internet traffic in the last ten years, widespread outage similar to the congestion collapse observed in the early years [1, 2] have not happened, even with the occasional flash crowds (Olympics, 9/11, etc.). Much credit for the Internet's ability to deal with such traffic bursts goes to the TCP's congestion control mechanism. TCP congestion control, however, only affects the traffic in a connection, not the number of connections. This number is determined by the users, who therefore play a role in controlling the traffic.

Such traffic control by users is particularly prominent during web surfing. A web surfer reacts to congestion in two ways: **(U1)** *she may abort a slow download by clicking "Stop", "Reload" or another hyper-link*, and **(U2)** *she may cut short her surfing session.* Such behavior is a form of congestion-induced **user back-off**: (U1) stops a download and (U2) reduces the number of completed downloads.

User reaction to network congestion can significantly affect network stability and traffic engineering. Indeed, aborting large HTTP pares down the tail of the file size distribution. This action in effect smoothens out self-similar traffic, possibly making elaborate traffic engineering for countering burstiness unnecessary.

As part of a larger study on the interaction between bandwidth supply and demand [3], we are interested in finding evidence for (U1) and (U2) in traffic traces. Figs. 1 and 2 show our main results, obtained from analysis of a 50GB tcpdump trace taken from a link in an academic network over two work days. Fig. 1 shows evidence for user back-off (U1). As download bandwidth decreases, the probability of aborting a download increases. The cumulative distribution function (cdf) is represented by the smoother curve. Fig. 2 shows evidence for user back-off (U2). As session bandwidth decreases, the number of completed downloads per session decreases; here, we focus on session bandwidths below 20KBps — the cdf indicates that they make up 95% of the data.

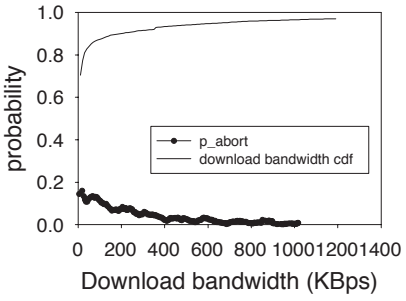

Fig. 1. Evidence for user back-off (U1).

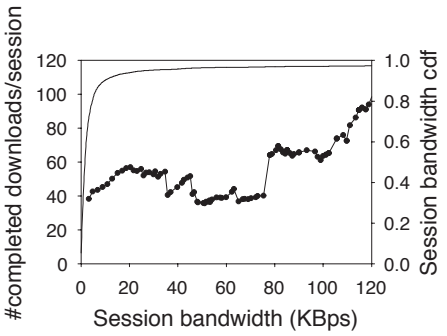

Fig. 2. Evidence for user back-off (U2).

The rest of this paper presents how we analyze the HTTP packet trace to obtain the results in Figs. 1 and 2 using SAX, a tool we developed to infer surfer actions from packet traces.

2 Models for Congestion and User Surfing Actions

Before we present SAX, we need to propose a good measure of "congestion". It is difficult to quantify network congestion in general using some metric. For example, download time is not a good measure of congestion, since it is affected by round-trip time, server load, etc. We therefore focus our analysis on a single bottleneck link [1], and define the congestion level on the link as the number of concurrent downloads at any instant on this link, denoted as k. Fig. 3 confirms that this metric is appropriate: our measurements show k traces the session arrival rate as it rises and falls over 48-hour.

An alternative measure of congestion level is the per-download bandwidth $b_k = (link\ bandwidth)/k$. Essentially, a b_k-axis reverses the k-axis, and the interesting, high-congestion part of the curve is compressed near the vertical axis; thus, presentation-wise, k seems a better choice than b_k.

To formalize actions (U1) and (U2), we need to model sessions, downloads and abort. In our model, a user surfs the web through a series of **sessions**, each consisting of a series of HTTP requests. A user sends HTTP requests by typing in URLs, clicking on bookmarks or hyper-links, etc. Each of these actions is modeled as a **click** and each click generates one **download**. A download may consist of multiple and possibly parallel HTTP requests. (Our download corresponds to Barford and Crovella's **web object** [4] and Choi and Limb's **web-request** [5].) One user may launch multiple downloads in parallel from different browser windows.

A user may be frustrated by — and abort — a download that takes too long to finish. For example, a user who is presented with several web search results may click one link, find the download too slow, and abort it by clicking on another link. We denote the probability that a download is aborted as p_{abort}. After the download is aborted,

[1] This singling out one link from the Internet is analogous to how classical demand-supply analysis isolates one market in a larger economy.

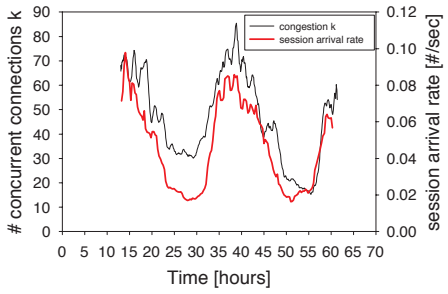

Fig. 3. How congestion k and session arrival rate changes over time.

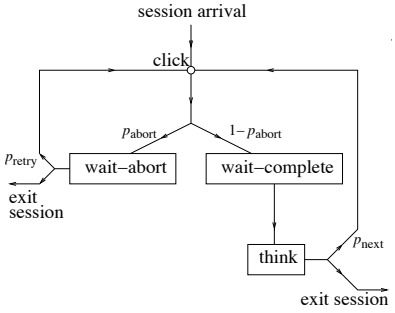

Fig. 4. User Surfing Model

the user might click again. We denote the probability of a click following an aborted download in the same session as p_{retry}. We say a download is completed, if it is not aborted. Probability p_{next} denotes the proportion of completed downloads that are followed by another click in the session. Fig. 4 shows this user behavior model, with three user states. A user stays in either **wait-abort** state (for aborted download) and **wait-complete** state (for completed download) while a download is in progress. A **think** state, where the user is viewing the completed download, follows the wait-complete state. One can elaborate on this simple model by adding sleep time between sessions and non-reactive elephantine downloads [3].

Analysis of the same HTTP packet trace in Section 1 illustrates how users behave when congestion-level changes. Fig. 5 plots p_{abort}, p_{next} and p_{retry} against k. As expected, p_{abort} increases with the congestion measure k, while p_{next} decreases. As for p_{retry}, there are two possibilities: at low congestion levels, when throughput is still satisfactory, an increase in congestion may prompt a user to retry; with poor throughput at high congestion levels, however, any increase in congestion may prompt a user to abandon the session. These possibilities are consistent with the slight increase in p_{retry} for small k in Fig. 5, and the slight decrease for large k. Note that, as expected, p_{retry} is less than p_{next} at every congestion level k, indicating that a user is less likely to continue a session after an abort. These graphs provide further evidence for user reaction to congestion.

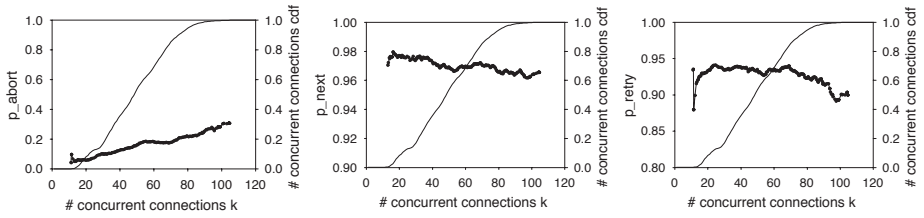

Fig. 5. How p_{abort}, p_{next} and p_{retry} vary with congestion measure k.

We obtain Fig. 5, by inferring user surfing actions indirectly, through analysis of packet-level HTTP traces. We chose this approach, as we found other alternatives impractical — using human observers to log surfing activities is time consuming, and modifying existing web browsers [6] to log user actions requires large-scale deployment of the modified browsers. Comparatively, analyzing packet traces is simpler as it involves only passive collection of packet traces and development of analysis tool. We describes this analysis tool next.

3 SAX: Surfer Actions eXtractor

We develop SAX as an off-line tool to infer user actions from HTTP packet traces. as a tool to infer user actions from HTTP packet traces. SAX takes HTTP packet dump as input, analyzes the relationships among the packets, and groups the packets into downloads. Each download is classified as either completed or aborted and is further grouped into sessions.

The difficulties of extracting HTTP information from packet level data are well-documented by Feldmann [7]. Additional issues encountered by SAX are:

(D1) Packets from parallel connections may belong to the same download, and packets from a persistent connection may belong to different downloads.
(D2) Two downloads from the same user might overlap in time – before a web page finishes downloading, the user may click on one of its links, thus initiating another download.
(D3) Aborted and completed downloads need to be distinguished.
(D4) Software-generated, automatic downloads do not reflect user behavior and need to be filtered out.

Grouping Packets into Downloads. A **request-response** is a collection of packets, containing an HTTP request, followed by an HTTP response with response header and data for that request. A **download** is a collection of request-responses, the first of which is initiated by a click. A click takes place when (C1) the user requests a page by entering the URL in the address bar of a browser or launch the browser through another application (such as an RSS reader) or (C2) the user clicks on an URL in a previously retrieved page.

To identify a click, we consider both cases above separately. A click generated by (C1) is identified by an HTTP request without a referrer, while a click generated by (C2) is identified by an HTTP request with a URL that corresponds to a hyper-link or submit button, with Referer field pointing to a previously downloaded web page.

After identifying the click that starts a download X, we proceed to group request-responses that belong to the same download. The subsequent request-responses in X consists of HTTP requests and responses for embedded objects required to display the containing web page. Such embedded objects are downloaded automatically by the browser. Just like (C2), the request header for such objects contains a Referer field that specifies the containing web page. However, the URLs of the requested objects appear as embedded objects in a previously downloaded document. Using this fact, SAX

includes in X any HTTP requests for embedded objects that have Referer fields (recursively) pointing to objects that are already included in X. Browser-initiated requests for Redirect links are also included in X.

HTTP requests generated by browsers due to auto-update, however, should be excluded. SAX uses a temporal threshold τ_{auto} to identify such requests. An embedded object Y with a Referer pointing to download X, is added to X only if the arrival time for Y is within τ_{auto} of X's last packet; otherwise, Y is identified as an auto-update.

One expects the period for auto-updates to be in minutes, whereas the requests for a web page and its embedded objects should occur within seconds of each other [8]. There is therefore considerable latitude in specifying the threshold τ_{auto}. Our measurements shows that 95% of the silent gap between Y and X is within 2 minutes. We thus set τ_{auto} to 2 minutes in our measurements.

Identifying Aborted Downloads. Having identified downloads, we now classify these downloads as either completed or aborted. Such classification was not performed in previous web traffic characterizations efforts.

We say a request-response is **not completed** if (i) the entity's length is specified in the response header and the amount of data received by the browser is less than the entity's length; or (ii) HTTP 1.1 and chunked transfer-coding are used and the browser has not received an end-of-chunk indication; or (iii) HTTP 1.0 is used and the entity's length is not specified in the response header (i.e. SAX assumes the server will send a `FIN` when the download completes).

SAX detects an **abort** of a request-response if the first `FIN` or `RST` is sent by the browser and the request-response is not completed. A download is **aborted** if and only if one of its request-responses is aborted. Otherwise, a download is **completed**. Some browsers terminate a download immediately if they see some special response headers (e.g. "304 Not Modified"). For such special cases, SAX considers the download as completed.

Extracting URLs from HTML Documents. SAX's method for detecting clicks and grouping request-responses into downloads relies on its accuracy in extracting URLs from the HTML documents. Issues that SAX addresses related to this task are:

- When a user clicks on the submit button of an HTML document X, the requested URL Y may contain the form's information. Some extra processing is necessary to match Y to X.
- Relative URLs need to be converted to absolute URLs for matching.
- A browser may request an embedded object immediately when it finds the URL in a partially received HTML document. SAX therefore needs to extract URLs upon arrival of any HTML fragments to keep pace with the behavior.
- The message body of a request-response can be encoded using chunked transfer-coding or content-coding (`gzip`, `compress`, etc.). SAX partially decompresses the packets in main memory to extract the URLs.

Excluding Background Downloads. Besides background downloads generated by auto-updates of web pages, SAX needs to exclude background HTTP downloads generated by non-browser applications (e.g. Windows Update). SAX therefore compiles a list of the most common URLs that it has seen; any background downloads that appear on this list are excluded during post-processing.

To define "most common", SAX checks periodically (every 20 minutes), for each user, whether a seen URL X is again requested; if so (no matter how many times), X's counter is incremented by 1. URLs with large counters are treated as most common URLs. This accounting allows SAX to differentiate downloads by applications which appear regularly over extended periods of time, from URLs that attract flash crowds.

Limitations of SAX. SAX is unable to process encrypted packets that belong to secure HTTP flows. Also SAX cannot identify some HTTP requests that are triggered by JavaScript since the URL can be created dynamically by the program. HTTP requests generated by JavaScript may have an empty Referer field, confusing SAX to treat it as a new click (C1). Finally, since SAX depends on HTTP packet dump, it cannot identify a download that is partially served from browser cache.

4 From SAX to the Model

The output from SAX includes download description (URLs, timestamps, size, abort/complete, etc.), request-response and Referer information, TCP connections, etc. We now describe how these are processed to give Figs. 1, 2, and 5.

Session Definition. Our user model groups clicks into **sessions**. Two sessions are separated by a **sleep time** T_{sleep}, where the user is not actively surfing the web. Within a session, a **think time** T_{think} separates a click from the previous download completion, while the download is viewed (think state in Fig. 4); in contrast, think time for CARENA separates one click instant from the next [6].

Think time and sleep time vary from one person to another and, for each person, from one session to another. In the traffic trace, think time and sleep time both appear as a silent gap t_{silent} between packets. To distinguish between think time and sleep time, we use a threshold τ_{session}, where $T_{\text{think}} < \tau_{\text{session}} < T_{\text{sleep}}$ in most cases. There is considerable latitude in specifying τ_{session} since, in general, $T_{\text{think}} \ll T_{\text{sleep}}$. Other studies have found average think time within a session to be less than a minute [9, 4, 5, 10]. In our experiments, we generously set τ_{session} to 10 minutes, since our measurements shows that 95% of silent gaps between two downloads is within 10 minutes. (Hlavacs and Kotsis used the thresholds of 8.3 minutes and 30 minutes in their model, depending on whether there was a change in web server address [11].)

Session Bandwidth and Download Bandwidth. Having identified downloads and sessions, we can now compute the session bandwidth b_{session} in Fig. 2. We are interested in this metric since the length of a session may depend on the surfer's aggregated experience of multiple downloads. We quantify this experience with b_{session}, defined as the

number of bytes transferred in a session (aborted or completed), divided by the sum of download transfer time.

A user may abort a slow download. Therefore, another metric of interest is the **download bandwidth** b_{download}, i.e. the number of bytes transferred in a download (aborted or completed) divided by its time span (see Fig. 1).

Δ-intervals. Recall that we use k, the number of concurrent downloads on the bottleneck link, as a metric for network congestion (Fig. 5). Measuring k, however, is not trivial as a download may contain silent gaps, may spread over parallel connections, and share a TCP connection with another download.

This issue led us to the following idea: Partition the trace into equal (non-overlapping) intervals of time. Let $(t, t+\Delta)$ denote the interval between times t and $t+\Delta$, where $\Delta > 0$, and $length(t, t+\Delta) = \Delta$; we call $(t, t+\Delta)$ a **Δ-interval**. Let $\mathcal{D}$ be the set of downloads, $d \in \mathcal{D}$, and I_d be the time interval between start and end of the download d. We measure the number of concurrent downloads in $(t, t+\Delta)$ by

$$k = \frac{\sum_{d \in \mathcal{D}} length(I_d \cap (t, t+\Delta))}{\Delta}. \tag{1}$$

This idea is illustrated in Fig. 6. Note that, if no download starts or ends during the interval, then Eqn. (1) gives precisely the number of downloads spanning that interval.

We also use Δ-intervals to measure the probabilities, as described below.

Measuring the Probabilities. Our user model has parameters p_{abort}, p_{next} and p_{retry}. To measure these probabilities, let n_{click}, $n_{\text{completed}}$, n_{abort}, n_{retry} and n_{next} be (respectively) the number of downloads, completed downloads, aborted downloads, retries after aborts, and clicks after think time. Then, one could calculate the probabilities by

$$p_{\text{abort}} = \frac{n_{\text{abort}}}{n_{\text{click}}}\ ,\ p_{\text{retry}} = \frac{n_{\text{retry}}}{n_{\text{abort}}}\ ,\ p_{\text{next}} = \frac{n_{\text{next}}}{n_{\text{completed}}}\ . \tag{2}$$

However, it is not clear how n_{click}, etc. are to be measured. An obvious choice is to measure them per session (Fig. 4), calculate each probability for every session, then aggregate the probability over the sessions. This approach has three problems:

- p_{retry} is a conditional probability, so it is ill-defined if $n_{\text{abort}} = 0$ for a session; p_{next} has a similar problem if $n_{\text{completed}} = 0$ for a session.
- How should the per-session values for p_{retry} (say) be aggregated over the sessions to give one p_{retry} value for each k?
- If a session ends with a completed download, $n_{\text{retry}} = n_{\text{abort}}$, so $p_{\text{retry}} = 1$ for that session; if a session ends with an aborted download, then $n_{\text{retry}} = n_{\text{abort}} - 1$, so p_{retry} for a session can only take values $0, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \ldots$. This discrete spread makes any *smooth* aggregation over all sessions difficult.

Therefore, instead of aggregating after the division (2), we first aggregate the values for n_{abort}, etc., then do the division. This can be done as follows: consider each Δ-interval and measure the number n_{abort} of aborted downloads and the number n_{click} of

downloads in that interval, then divide one by the other to get p_{abort}. Each Δ-interval thus gives a (k, p_{abort}) pair, from which we derive the relationship between the two metrics.

However, the size of Δ forces a tradeoff: a large Δ gives a poor measurement for k, while a small Δ gives noisy measurements of n_{click} and n_{abort}; furthermore, user reaction to congestion within the Δ-interval may occur after that interval.

We resolve this tension thus: for each Δ-interval, let n_{click} and n_{abort} be the number of downloads and aborted downloads from the start of that Δ-interval to the end of the session; their ratio then gives p_{abort} for that Δ-interval. We measure p_{retry} and p_{next} similarly.

Curve Smoothing. By using Δ-intervals for the measurements, we have discretized time; add to this the bursty nature of network traffic, and the data becomes jittery, making it difficult to discern trends.

To smooth out the jitter, we use a sliding window of size L units. For example, to smooth out a function of k, a unit is one integer value. If we have (k, p_{abort}) measurements sequenced by k, then we consider consecutive (k, p_{abort}) measurements within that window (i.e. such that $x \leq k \leq x + L$, where x is the start of the window), and average the k values and the p_{abort} values to give an aggregate pair; we then slide the window by 1 (i.e. consider window $x+1 \leq k \leq x+1+L$), and repeat the aggregation.

In our measurements, we chose $L = 6$ for smoothing k in Fig. 5; for smoothing a function of download bandwidth in Figs. 1 and 2, each unit is 50KBps. We set $\Delta = 1$ minute in our measurement. The smoothened curves yield our main results, which are presented in Figs. 1, 2 and 5.

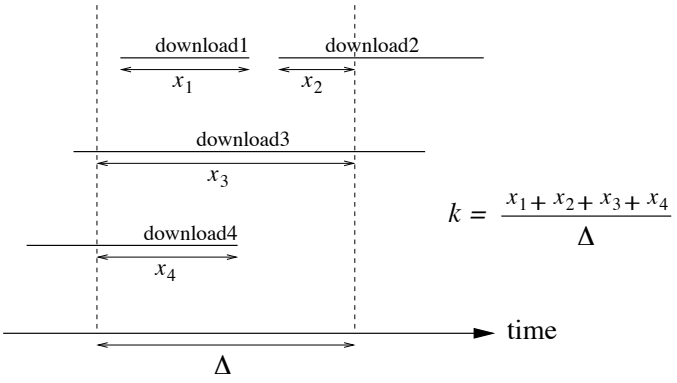

Fig. 6. Using intervals to measure the number of concurrent downloads k.

5 Related Work

User Behavioral Model. Our model is the first to study how users react to congestion. Choi and Limb's behavioral model [5] and Barford and Crovella's user equivalent [4] do not include sessions (U2), whereas the layered model by Hlavacs and Kotsis does not provide for user reaction to delays [11].

Rossi et al.'s measurement study of how transfer delays cause users to interrupt TCP connections is relevant [12], but they do not offer a user model. Furthermore, a download may be more than one TCP connection.

Studies on how users react to server delays [13, 14] are only marginally relevant since, in our context, the users may be visiting different web sites, and each user may visit multiple web sites.

Characterizing Web Traffic. Mah [15] was one of the first to model HTTP traffic by analyzing packet dumps. He used a threshold (1 second) between arrival times of packets to determine whether two HTTP connections belong to the same download. Such a heuristic can fail if two downloads overlap (D2). A similar approach is used by Barford-Crovella [4], Lan-Heidemann [16], Smith et. al. [17], Molina et. al [18], and Abrahamsson-Ahlgren [19]. Choi and Limb pointed out the inadequacy in relying on a 1-second threshold [5]; instead, they parsed the HTTP headers to detect the start of downloads. However, header information may not be enough, and it may be necessary to extract information from the body as well.

None of the previous work characterizes downloads as completed or aborted (D3). Rossi et al. [12], used TCP `FIN` and `RST` to distinguish completed and aborted TCP connections, which does not correspond to downloads due to parallel or persistent connections (D1).

Among the related work, Abrahamsson-Ahlgren [19] is the only study that groups downloads into sessions. They use a threshold of 15 minutes to separate HTTP requests into different sessions. Other previous studies did not distinguish between think time and sleep time.

Packet Analysis Tool. Feldmann's BLT is a tool for extracting HTTP information from sniffed packets [7], much like HTTPdump [20] and the more general Nprobe [21]. One could use such information for further studies of compression, traffic invariants, proxy caching, etc. In principle, we can process BLT output to identify user actions, such as clicks and aborts. However, the heuristics used by such general tools for handling missing packets, erroneous HTTP format etc. filter out some information that are needed by studies like ours, e.g. for distinguishing between a click and a download of an embedded object.

6 Conclusion

We presented an analysis tool called SAX that infer user surfing behavior from HTTP packet traces. SAX was designed to confirm the existence of congestion-induced user

back-offs while surfing. We believe, however, that SAX is useful in its own way, for instance, to researchers studying affects of web caches, or to researchers studying surfing behavior of different social groups. Furthermore, the user surfing model constructed using SAX can be used in a network simulator such as ns-2 to generate web traffic that incorporates user behavior.

References

1. Nagle, J.: Congestion Control in IP/TCP. IETF. (1984) RFC 896.
2. Jacobson, V.: Congestion avoidance and control. In: Symposium proceedings on Communications architectures and protocols, ACM Press (1988) 314–329
3. Tay, Y.C., Tran, D.N., Liu, E.Y., Ooi, W.T., Morris, R.: Modeling web surfers and bandwidth demand/supply for congestion-induced behavior. Submitted (2005)
4. Barford, P., Crovella, M.: Generating representative web workloads for network and server performance evaluation. In: Proc. SIGMETRICS. (1998) 151–160
5. Choi, H.K., Limb, J.O.: A behavioral model of web traffic. In: Proc. Int. Conf. Network Protocols. (1999) 327–334
6. Nino, I.J., de la Ossa, B., Gil, J.A., Sahuquillo, J., Pont, A.: Carena, a tool to capture and replay web navigation sessions. In: E2EMON. (2005)
7. Feldmann, A.: BLT: Bi-layer tracing of HTTP and TCP/IP. Computer Networks **33** (2000) 321–335
8. Casilari, E., Reyes-Lecuona, A., González, F., Diaz-Estrella, A., Sandoval, F.: Characterization of web traffic. In: GLOBECOM. (2001) 1862–1866
9. Arlitt, M., Williamson, C.: A synthetic workload model for internet mosaic traffic. In: Proc. Summer Computer Simulation Conference. (1995) 852–857
10. Reyes-Lecuona, A., González, E., Casilari, E., Casasola, J., Diaz-Estrella, A.: A page-oriented WWW traffic model for wireless system simulations. In: Proc. Int. Teletraffic Congress (ITC-16). (1999) 817–826
11. Hlavacs, H., Kotsis, G.: Modeling user behavior: a layered approach. In: MASCOTS. (1999) 218–225
12. Rossi, D., Casetti, C., Mellia, M.: User patience and the web: a hands-on investigation. In: GLOBECOM. (2003) 4163–4168
13. Arlitt, M., Williamson, C.: Internet web servers: workload characterization and performance implications. IEEE/ACM Trans. on Networking **5** (1997) 631–645
14. Dalal, A., Jordan, S.: Improving user-perceived performance at a world wide web server. In: GLOBECOM. (2001) 2465–2469
15. Mah, B.A.: An empirical model of HTTP network traffic. In: INFOCOM (2). (1997) 592–600
16. Lan, K.C., Heidemann, J.: Rapid model parameterization from traffic measurements. ACM Trans. on Modeling and Computer Simulation **12** (2002) 201–229
17. Smith, F.D., Campos, F.H., Jeffay, K., Ott, D.: What TCP/IP protocol headers can tell us about the Web. In: SIGMETRICS/Performance. (2001) 245–256
18. Molina, M., Castelli, P., Foddis, G.: Web traffic modeling exploiting tcp connections' temporal clustering through html-reduce. IEEE Network **14** (2000) 46–55
19. Abrahamsson, H., Ahlgren, B.: Using empirical distributions to characterize web client traffic and to generate synthetic traffic. In: GLOBECOM. (2000) 428–433
20. Wooster, R., Williams, S., Brooks, P.: HTTPDUMP: Network HTTP packet snooper. `http://ei.cs.vt.edu/~succeed/96httpdump/` (1996)
21. Moore, A., Hall, J., Kreibich, C., Harris, E., Pratt, I.: Architecture of a network monitor. In: PAM. (2003)