What TCP/IP Protocol Headers Can Tell Us About the Web*

F. Donelson Smith, Félix Hernández Campos, Kevin Jeffay, David Ott
University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, NC 27599-3175 USA
http://www.cs.unc.edu/Research/dirt

Abstract

We report the results of a large-scale empirical study of web traffic. Our study is based on over 500 GB of TCP/IP protocol-header traces collected in 1999 and 2000 (approximately one year apart) from the high-speed link connecting The University of North Carolina at Chapel Hill to its Internet service provider. We also use a set of smaller traces from the NLANR repository taken at approximately the same times for comparison. The principal results from this study are: (1) empirical data suitable for constructing traffic generating models of contemporary web traffic, (2) new characterizations of TCP connection usage showing the effects of HTTP protocol improvement, notably persistent connections (*e.g.*, about 50% of web objects are now transferred on persistent connections), and (3) new characterizations of web usage and content structure that reflect the influences of "banner ads," server load balancing, and content distribution. A novel aspect of this study is a demonstration that a relatively light-weight methodology based on passive tracing of only TCP/IP headers and off-line analysis tools can provide timely, high quality data about web traffic. We hope this will encourage more researchers to undertake ongoing data collection and provide the research community with data about the rapidly evolving characteristics of web traffic.

1. Introduction and Background

By any measured quantity — bytes, packets, or flows — web traffic has become the single largest consumer of Internet resources [9, 22, 30]. Good characterizations of how web traffic "looks" in the network are essential for networking experiments investigating end-to-end performance issues in the web. Usually this involves constructing a model of web traffic and using the model to introduce synthetically generated web traffic into a simulation or laboratory network. For example, a critical element of networking research involving the effect on web traffic of TCP's end-to-end congestion-control mechanism, or router-based mechanisms such as RED, is the generation of synthetic web traffic for experiments [4, 8, 18, 25]. Paxson and Floyd [28] have presented compelling arguments for the importance of using application-dependent but network-independent traffic sources layered over (real or simulated) TCP implementations in such experiments. Constructing a traffic generator for TCP applications depends ultimately on the availability of high quality measurement data that represents application characteristics. In this paper we address this requirement for web traffic. We present the results of a measurement study that collected over 500 GB of TCP/IP headers in 1999 and 2000 from a high-speed link connecting our university campus to its Internet service provider.

While one motivation for our measurements was to provide the modeling foundation for generating synthetic web traffic, another motivation was to provide new results on the effects recent versions of the HTTP protocols are having on the characteristics of web traffic in the Internet. For example, measurements of TCP connection usage for early versions of the HTTP protocols pointed to clear inefficiencies in design, notably the creation of a different TCP connection for each web object reference [23]. Recent revisions to the HTTP protocol, notably version 1.1 [24], have introduced the concepts of *persistent connections* and *pipelining*. Persistent connections are provided to enable the reuse of a single TCP connection for multiple object references at the same IP address (typically embedded components of a web page). Pipelining allows the client to make a series of requests on a persistent connection without waiting for a response between each request (the server must, however, return responses in the same order as the requests are sent). As browsers and servers have migrated to support the 1.1 version of HTTP, there has been very little data collected from production networks to show how these protocol modifications have changed the usage and behavior of TCP connections. Our results show that HTTP 1.1 is already producing very significant effects. We also show that other rapidly evolving developments such as the presence of "banner ads," server load balancing, and content distribution networks are influencing several observable characteristics of TCP connections in the web.

A final motivation is related to methodology. Our approach emphasizes simplicity and a passive, non-invasive method of measuring. We used widely available packet-capture tools to gather traces consisting only of TCP/IP headers, each time-stamped with its arrival time. Using only the information in the TCP/IP headers and knowledge of the TCP and HTTP protocols, we created trace-processing tools to analyze individual TCP connections and reconstruct properties of the higher-level protocol (HTTP) from the TCP segment headers. We explicitly decided not to capture HTTP protocol headers mostly because of privacy concerns about tracing users' data falling beyond the TCP/IP headers[1] and also to reduce the volume of data in traces covering extended periods of time. We show that the use of relatively straightforward methodology can produce timely, high quality data and we hope to encourage more researchers to undertake ongoing data collection programs. If this happens, it will help provide the research community with more data about the rapidly evolving characteristics of web traffic.

* This work supported in parts by grants from the National Science Foundation (grants CDA-9624662, ITR-0082870, and ITR-0082866), the Cisco, IBM, Intel, Sun, Cabletron, and Aprisma corporations, and the North Carolina Networking Initiative.

[1] IP addresses are made anonymous in our traces to remove privacy concerns related to identifying individual users.

Published in: *Proceedings of* ACM *SIGMETRICS 2001/Performance 2001*, Cambridge, MA, June 2001, *to appear*.

From the analysis of our trace data, as well as data acquired by NLANR during the same periods, we make a number of observations about the evolving nature of web traffic, the use of the web, the structure of web pages, the organization of web servers providing this content, and the use of packet tracing as a method for understanding dynamics of web traffic. Specifically:

- From a methodology standpoint, we conclude that a substantial and detailed analysis of HTTP request/response exchanges is possible given only the TCP/IP headers of packets sent from servers to clients. Given only a unidirectional trace, one can discern transport protocol phenomena, the effects of higher-level protocols (*e.g.*, the use of persistent HTTP connections), and application-level phenomena (*e.g.*, the number of embedded objects and the number of servers delivering this content). Moreover, these data can be quickly and cheaply obtained. The most recent traces were acquired with commodity PCs and publicly available packet capturing software. We have also developed a rich set of tools for processing unidirectional packet traces that enabled us to present much of the data in this paper less than a month after the raw traces were acquired.[2]
- We conclude that traces of web traffic must be quite long (on the order of hours) in order to fully capture the tail of the distribution for measures such as the size of responses from servers to clients. We have observed that a very small number of responses from servers are multiple hundreds of megabytes (some close to a gigabyte) and that the TCP connections carrying some responses can be active for upwards of an hour.
- From a protocol usage standpoint, we observe that (port 80) web traffic has declined recently as a percentage of the total number of TCP connections recorded. For web traffic, 15% of all the TCP connections carrying HTTP request/response exchanges are persistent in the sense that they actually carry more than one request/response exchange. These persistent connections deliver 40-50% of all the web objects requested and these objects account for approximately 40% of all the bytes transferred. Thus, although the fraction of connections that are persistent is small, their use represents a 50% reduction in the total number of TCP connections required to deliver web content (compared to a similar environment in which persistent connections are not used).
- From an analysis of HTTP responses, we see that 65% of all web pages are constructed entirely by responses from a single server while 35% of the pages receive content from two or more servers. Close to 70% of the consecutive top-level page references go to an IP address that is different from the address used for the previous top-level page reference [29]. We speculate that this reflects the increasing trend of large organizations to manage web sites with a "server farm" for load balancing.
- We also see a year-to-year increase in the number of embedded objects per web page and an increase in the frequency of smaller objects. This is possibly due to the pervasive use of "banner adds" and icons to decorate pages. Overall the number of bytes transferred to deliver embedded objects is increasing [29].
- Overall we see a slight year-to-year increase in the frequency of small response objects but a marked increase in the size of the largest objects transferred. We find that the top 15% of object sizes account for 80% of the bytes sent by servers (objects greater than 1 MB in size account for 25% of the bytes).
- From an analysis of HTTP requests we see novel uses of web requests to implement applications such as "web email." These uses result in a shift in the distribution of request sizes because files are being transferred to servers (*e.g.*, email attachments) in the context of an HTTP request.

The remainder of this paper is organized as follows. Section 2 reviews the literature in previous measurement studies of network traffic in general and web traffic in particular. We also review the methods used to gather and process network traces. Section 3 gives an overview of our measurement methodology and presents summary statistics for the traces comprising our study. Section 4 presents our methodology for reconstructing and analyzing HTTP connections given only the information contained in TCP/IP headers. Section 5 presents the analysis of user browsing metrics such as think time. Section 6 gives our analysis of the usage of TCP connections by HTTP including the effects of persistent HTTP connections. Section 7 analyzes HTTP request and response sizes. Section 8 discusses the limitations of the methodology used and how these limitations introduce uncertainty in the reported results. Finally, Section 9 summarizes the results and conclusions of our study.

2. Related Work

Two important measurement efforts that focused on application-specific traffic models, but which preceded the growth of the web, were conducted by Danzig *et al.*, [7, 13, 14], and by Paxson and Floyd [26, 27]. More recently, measurements to characterize web usage have become a very active area for research. Because caching and content delivery are widely considered vital to the long-term viability of the web, most of the high-quality data currently available is focused on providing inputs to cache or content delivery evaluations, *e.g.*, [15, 16, 19, 32, 33]. For these studies, the critical data are traces or logs of URL references, typically collected at proxies or servers. There is much less data available that is focused on how web browsing behaviors by users result in the creation of network traffic. For networking studies, the critical data are related to characterizing the TCP connections between web browsers and servers in terms of connection establishment rates and the sizes and timing of exchanges of request and response data.

Web traffic generators in use today are usually based on data from the two pioneering measurement projects that focused on capturing web-browsing behaviors: the Mah [20], and Crovella, *et al.*, [3, 5, 11, 12] studies. Traffic generators based on both of these sources have been built into the widely used *ns* network simulator [6] that has been used in a number of studies related to web-like traffic, *e.g.*, [18, 25]. These models have also been used to generate web-like traffic in laboratory networks [4, 8]. For both sets of measurements, the populations of users were highly distinctive and the sizes of the traces gathered were relatively small. Mah captured data reflecting a user population of graduate students in the Computer Science Department at UC Berkeley. His results were based on analysis of approximately 1.7 million TCP segments carrying HTTP protocols. The measurement programs by Crovella and colleagues reflected a user population consisting primarily of undergraduate students in the Computer Science Department at Boston University and in aggregate represented around 1 million references to web objects. In addition, both sets of data are now

[2] It is our intent to distribute both our tools and the data we have acquired (and are continuing to acquire).

relatively old (especially as measured in "Internet time"). The Mah data were collected in 1995 and the Crovella, *et al.*, data in 1995 and 1998. It is especially important to note that these studies were conducted before significant deployment of HTTP version 1.1 protocol implementations. For comparison, our study involved traces consisting of over 800 million TCP segments generated by a user population of approximately 35,000 and representing the transfer of some 55 million web objects. Moreover, we have developed a capability for nearly real-time analysis of our data (the first draft of this paper was completed within a month of collecting the last traces).

We are aware of at least five projects involving the analysis of large-scale packet-level traces containing web traffic. None of these projects have reported on any analysis of the traces to extract information complete enough to create traffic generating models or characterize user browsing behavior. Gribble and Brewer [19] focused their study on factors that would be of use to web cache designers (locality, cache-control headers, *etc.*) but they also reported some data on request interarrival times and mean sizes for HTML, GIF, and JPEG object types. Balakrishnan, *et al.* [2], were concerned with issues related to improving TCP performance such as TCP loss recovery, ACK compression, receiver bottlenecks, and congestion control for multiple parallel connections. Cleveland, *et al.* [10] created a statistical model for generating TCP connection start times from web clients using a notion of connection-rate superposition but they did not model other TCP connection characteristics or web browsing behavior. Researchers at the University of Washington collected large-scale traces that were used in two studies of web proxy caching [32, 33]. Feldman [17] summarizes the results of over three years of large-scale trace-gathering projects using PacketScope monitors (with special on-line analysis software to process HTTP headers) at several locations in the AT&T WorldNet IP network. She reviews the many challenges faced in reconstructing TCP connections from individual segments and reconstructing HTTP protocol characteristics from TCP connections and HTTP headers (many of which we encountered with our own analysis tools).

Our approach is to use off-the-shelf hardware and publicly available packet capture tools. Privacy considerations limit us from capturing more than the TCP/IP header, however, as we show below, with careful analysis significant and substantial data on protocol usage and the nature, structure, and distribution of web content can be gleaned from just the TCP/IP headers. One drawback to our approach is that all processing of the traces is done off-line. This means that the length of traces is fundamentally limited by the amount of disk space available, the packet arrival rate, and the bytes traced per packet. As a practical matter this limited individual traces of TCP/IP headers to about one hour (we expect to soon add enough disk capacity to trace over eight hour intervals). More elaborate hardware/software instrumentation that supports continuous capture and real-time analysis has been developed elsewhere [17, 19, 21, 32]. For our intended uses of the data, however, we believe our simpler approach represents a viable alternative to instrumentation embedded directly in browser software or to specialized tracing hardware and software that analyzes entire packet contents (including user data) and logs the results. Both of these approaches present significant barriers to widespread use. Given the rapidly falling prices and increasing sizes of PC hard disks, it is quite feasible to use very large pools of storage for intermediate processing of TCP/IP header traces.

3. The Trace Data

The data used in our study are from two sources. The two largest trace collections were obtained by placing network monitors on the high-speed link connecting the University of North Carolina at Chapel Hill (UNC) campus network to the Internet via our Internet service provider (ISP). All units of the university including administration, academic departments, research institutions, and a medical complex (including a hospital that is the center of a regional health-care network) all use a single ISP link for Internet connectivity. The user population is large (over 35,000) and diverse in their interests and how they use the web — including, for example, student "surfing" (and music downloading), access to research publications and data, business-to-consumer shopping, and business-to-business purchases by the university. In addition to the thousands of on-campus computers (the large majority of which are Intel architecture PCs running some variant of Microsoft Windows), several units of the university operate dial-in modem pools (total of about 250 ports) that are used by some students, faculty, and staff for access from home. In effect, the university is a local ISP for this population, forwarding all Internet traffic to its upstream ISP. There are only a handful of small proxy servers on campus so almost all the web traffic is generated directly by browser actions. It is important, however, to remember that all web traffic we observed represents only requests that could not be satisfied from local browser caches.

We used network monitors to capture traces of TCP/IP headers from all packets entering and leaving the campus network. The traces were collected during six one-hour sampling periods each day. The one hour sampling periods were 8:30-9:30AM, 11:00-12:00 noon, 1:30-2:30PM, 4:00-5:00PM, 7:30-8:30PM, and 10:00-11:00PM. These periods were chosen somewhat arbitrarily to produce four traces during the normal business day and two during non-business hours when traffic volumes were still reasonably high. One set of traces consists of all the TCP/IP headers collected during these sampling intervals over a seven-day period, in late September 1999. This seven-day period provided a set of six traces from each of the seven weekdays for a total of 42 one-hour traces. This set of traces will be referred to in this paper as "UNC-99." The second set of 42 traces also consists of traces taken at the same hours over a seven-day period, in late September 2000 (referred to as "UNC-00"). This allows us to compare results from traces gathered approximately one year apart.

When the UNC-99 traces were gathered, our campus was connected to the ISP by an OC-3 (155 Mbps) full-duplex, ATM link. This link carried all network traffic between the campus and the "public" Internet (traffic between the campus and Internet 2 sites was routed over a separate OC-3 link). We placed the monitor on the OC-3 link to the "public" Internet. The specific monitor used was the OC3mon developed initially at MCI for vBNS [1] and now distributed by CAIDA (CoralReef) [34]. This monitor was passively inserted in the link using a fiber splitter to divert some light from the optical signal. The signal is input to the receive port of an ATM interface card in an Intel architecture PC equipped with large, high-performance disks. Because the link was full-duplex (two fibers), the monitor required two ATM interface cards, each receiving the signal for one of the directions of transmission (*i.e.*, inbound or outbound with respect to the campus). The OC3mon software ran on FreeBSD (version 2.2.8) and produced traces of timestamped entries giving the contents of ATM cells carrying the TCP/IP headers. The trace entries for each interface were filtered into separate traces

for the inbound and outbound traffic. We also converted the OC3mon trace into the format used by *tcpdump* using a locally-modified version of a tool, *mon2dump*, originally developed at MCI.

A year later when the UNC-00 traces were taken, the ISP link had been replaced by an OC-12 (622 Mpbs) path based on Cisco-proprietary DPT technology instead of ATM. Thus we could not use the same monitoring tool we used for the 1999 traces. Fortunately, all the traffic between the campus and the Internet traversed a single full-duplex gigabit Ethernet link from the campus aggregation switch to the edge router with the DPT interface. In this configuration, both "public" Internet and Internet 2 traffic are co-mingled on the one Ethernet link (the only traffic on this link is traffic to and from the ISP edge router). We again placed a monitor on this gigabit Ethernet by passively inserting a fiber splitter to divert some light to the receive port of a gigabit Ethernet network interface card (NIC) set in "promiscuous" mode. As the link was still full-duplex, the monitor required two NICs, and each monitored one direction of transmission. The NICs were hosted in a PC running FreeBSD version 3.2. An instance of the *tcpdump* program was run on each of the interfaces to collect a trace of TCP/IP packet headers. Buffer space of 3 MB was allocated to the *bpf* devices used by *tcpdump* to buffer transient overloads in packet arrivals. The *tcpdump* program reports statistics on the number of packets dropped. We found that in many traces no packets were dropped and that the maximum number of drops in any trace was less than 0.02% (average of 0.004% drops over all traces).[3]

In this paper we use data only from the traces of packets flowing into the campus network from the ISP ("inbound" packets). Summary statistics for the inbound packets in the two sets of UNC traces is given in Table 1. The volume of Internet data increased significantly between 1999 and 2000. Some part of this can be attributed to the fact that the 2000 data includes both "public" Internet and Internet 2 traffic. However, most of it is simply growth of Internet usage by the university population. While the number of HTTP bytes flowing into the campus doubled between 1999 and 2000, web traffic actually declined as a percentage of all TCP traffic. This is attributed to the sudden popularity of the *Napster* application for downloading audio files [36].

As a "sanity check" of our data, two considerably smaller sets of traces from the repository of traces at NLANR [35] were used for comparison. This allowed us to both debug our measurement methodology and verify that other university campuses see similar patterns of web traffic. The NLANR traces were gathered using the OC3mon/Coral software described above. We selected from the NLANR repository two sets of traces collected at approximately the same time periods in 1999 and 2000 as our UNC traces. Within these time periods, we selected traces from sites that appeared to have relatively high volumes of traffic during the sampling times used by NLANR. The set of traces we refer to as "NLANR-99" is composed of eight traces from each of two sites (Merit-University of Michigan, and the San Diego Supercomputer Center "commodity connection") all taken on September 19, 1999. Our "NLANR-00" set also consists of eight traces from the same sites. These traces were all taken on September 28, 2000. Summary statistics for the NLANR traces are also given in Table 1.

[3] The OC3mon does not explicitly report dropped packets but halts with an error if the trace overruns the disk. This did not occur in any of our traces.

Table 1: Summary data for the UNC and NLANR traces (all counts in thousands).

	UNC-99	UNC-00	NLANR-99	NLANR-00
TCP Packets	525,258	1,872,964	16,919	18,656
% TCP Packets	85.08%	90.77%	84.66%	90.13%
UDP Packets	89,759	180,482	2,681	1,596
% UDP Packets	14.54%	8.75%	13.42%	7.71%
HTTP Packets	232,245	602,183	9,263	7,617
% HTTP Packets	37.62%	29.18%	46.35%	36.80%
Total Packets	617,333	2,063,351	19,985	20,699
TCP Bytes	211,610,632	721,866,693	8,374,506	9,744,043
% TCP Bytes	86.15%	89.84%	93.08%	95.85%
UDP Bytes	33,760,062	80,921,395	480,644	385,298
% UDP Bytes	13.74%	10.07%	5.34%	3.79%
HTTP Bytes	138,050,697	278,484,679	4,198,123	3,351,340
%HTTP Bytes	56.20%	34.66%	46.66%	32.97%
Total Bytes	245,636,674	803,493,236	8,996,799	10,165,498

4. Analysis of TCP Connections

Because we have only traces of TCP/IP headers, all the statistics we report here have been derived from analysis of these headers knowing their formats and the dynamic behaviors of the TCP and HTTP protocols. The primary information used from the TCP/IP headers was the IP source and destination addresses, the protocol number (to identify TCP segments), the source and destination ports, the TCP flags (to detect SYN, FIN, and Reset), the data sequence number, and the acknowledgement (ACK) number.

The OC3mon tool used to gather the UNC-99 and all the NLANR traces timestamps the arrival of each trace entry using a clock local to the ATM interface card that receives the data. Because the trace entries from each direction of data flow on the full-duplex OC-3 link are timestamped with a different clock, it is difficult to create a merged bi-directional trace in the correct order. Even though the Coral software attempts to initialize the clocks on both interface cards to the same value, there is usually a random initial offset between the clocks and the clocks may drift at different rates. Our attempts to produce merged bi-directional traces by sorting on timestamps resulted in a large number of TCP connections with obviously incorrect orderings (*e.g.*, SYN+ACK before SYN).[4] Fortunately, it is not necessary to merge (or even use) both directions of data flow in order to reconstruct important elements of the internal dynamics within a TCP connection and infer parts of the HTTP protocol.

Consider a trace consisting only of TCP/IP packet headers captured on the interface receiving IP packets arriving on the inbound path (to the university from its ISP). We first filter this trace to pull out only those IP packets where the protocol field in the IP header designates TCP and the *source* port field in the TCP header contains the value 80 (the normal HTTP server port). This produces a trace of TCP/IP packet headers that were sent from Web servers somewhere in the Internet to Web clients (browsers) located at the university. This filtered trace is then sorted on three keys in the following order: source IP address and port, destination IP address and port, and timestamp. This produces a time ordered trace of TCP segments within each TCP connection (actually within unique TCP connection ad-

[4] Later versions of the Coral software have improved the bounds on clock synchronization.

dress 4-tuples; port reuse occasionally produces multiple TCP connections using the same 4-tuple within a trace). The HTTP protocol is asymmetric — the client always initiates the connection (sends the TCP initial SYN segment). The server normally continues the connection establishment protocol by responding with a SYN+ACK segment. This SYN+ACK segment should appear in our trace and its timestamp is used for the beginning time of a TCP connection. Similarly the connection is considered to end when a FIN or Reset segment from the server is found in the trace.

For the HTTP protocol, we are interested in the exchanges of data between the browser and server that occur within the TCP connection. Specifically, we want to identify the first and last bytes of browser requests and the first and last byte of server responses. For those cases where the TCP connection is used for more than one request/response pair (when both the browser and server support persistent connections), we need to identify the beginning and end of requests and responses for multiple exchanges between the browser and server. Fortunately the TCP protocol allows us to infer this information from examination of only the TCP segments flowing from the server to the browser. Consider the common case where the first TCP segments that flow on an established TCP connection are the HTTP-request protocol elements sent by the browser. As the TCP protocol stack on the server receives the segment(s) comprising the request, it will send TCP acknowledgment sequence numbers (ACKs) indicating the in-order byte sequence it has received. These may be sent in segments containing only an ACK or in segments containing an ACK along with HTTP-response protocol elements and perhaps object data. The ACK may be sent immediately, delayed by up to 200 milliseconds, or be sent on the next outbound data segment on that connection. The important observations are that the ACK value will advance by an amount equal to the size of the request protocol elements and that all of the request message will be ACKed *no later than* the first segment carrying any data for the corresponding response. The size of the response is indicated by the amount the data sequence number advances in segments from the server. Note that this method of computing the response size means that it is the sum of the HTTP headers and the size of the referenced object. Since the HTTP response headers are non-trivial in size (around 200-300 bytes is typical), the reported response sizes are larger than actual object sizes.

In the case of persistent connections with more than one request/response exchange, we will see an alternating pattern of advancing ACK values followed by advancing data sequence numbers from the server. Pipelining could complicate the identification of request/response exchanges in persistent connections and is discussed in Section 8.

Fundamental to all of this is the observation that a server should not be sending data in response to a request unless that data is accompanied or preceded by an advance in the ACK sequence covering receipt of the request segments. Similarly, any new data segment sent by a server that follows, or is accompanied by, an advance in the ACK sequence number is assumed to be a response to the request that caused the ACK sequence to advance. Put another way, response segments (a sequence number advance) mark the end of a request and ACK advances mark the end of a response. Of course other events such as FIN or Reset can mark ends also. A request's start time is the timestamp on the trace entry containing the first advance in the ACK field following the connection establishment or a sequence of response segments. A response's start time is the timestamp on the trace entry containing the first advance in the sequence number field following a sequence of request segments. A response's ending time is the timestamp on the trace entry that last advanced the data sequence number before the response ended (a new request starts, a FIN is sent, *etc.*). Similarly, a request is considered to end at the timestamp on the last trace entry of the request.

All of this would be quite straightforward if it were not for all the ways real TCP connections deviate from such well-behaved traces.[5] Retransmissions and segment reordering in the network disturb this use of advancing ACK/data sequence numbers to mark requests and responses. In traces of TCP segments, the data sequence numbers may not be monotonically increasing. In some cases, such as re-orderings or retransmissions of data-only segments in a response, this presents no problem since only the highest sequence number seen is used. The length of a response in a persistent connection can be computed as the difference between the (largest) sequence at the end of one response and the (largest) sequence at the end of the subsequent response. ACKs should be monotonically increasing so the length of a request in a persistent connection can be computed as the difference between the ACK value marking the end of one request and the ACK value marking the end of the next request. Unfortunately, out-of-order segments can cause ACKs to appear to "go backward." For segments without data (ACK only), simply ignoring the "backward" ACK is correct. Reordering of segments with changes in both data sequence numbers and ACK values presents problems since boundaries between requests and responses may be missed which can result in overstating or understating request and response sizes. Issues related to analysis of TCP segments are discussed in Section 8.

5. User and Web Content Characterizations

Our traces do not include any part of the user data carried in TCP segments and hence we do not have access to any of the HTTP protocol headers. We must, therefore, use heuristics to infer characteristics of user and browser behavior from the analysis of TCP connections. The first step in this process is to sort the summary descriptions of the TCP connections in each of the individual traces as produced by the analysis program described above. These summary traces are sorted on two keys; first by unique client IP addresses (the unique IP destination addresses found in a trace filtered for source port 80) and then by time. This creates a time-sorted summary of the TCP connection activity between each individual client and the server(s) that client used during the one-hour trace. The time-sorted summary of TCP connections contains the connection start time, the client IP address and port number, the server IP address, the beginning time and size in bytes of each request, the beginning and ending times of each response along with its size, and the ending time of the connection. We then use this time-ordered information to infer certain characteristics of activity by the user or the browser.

We assume that in the vast majority of cases a client IP address identifies a single human user running one or more browser instances on a personal computer or workstation. Although we know that there are times when multiple users concurrently run browsers on a shared compute server (single client IP address),

[5] We describe here only the "expected" exceptional behaviors related to loss, retransmissions, and reordering; the truly bizarre sequences that our analysis tools unearthed are fortunately so rare that completely discarding those connections does not effect the results.

we believe this to be rare on our campus where there is basically one computer for each Internet user. Furthermore, even though the vast majority of computers on our campus have IP addresses assigned by DHCP, we believe the reuse of a given IP address on different machines during a single one hour trace is rare because leases last eight hours or more. Further, many of the larger DHCP servers maintain a fixed mapping of IP address assignments to Ethernet MAC addresses.

Using heuristics similar to those developed originally by Mah [20] and Barford and Crovella [3], we attempted to identify points in each client's activity that are likely to mark a request for a new (or refreshed) page. We use the term "page" as a convenient label for a web object referenced in a "top-level" sense, *i.e.*, it is not referenced through interpreting references found internal to some other object (*e.g.*, embedded references in HTML). We also use the term "object" synonymously with a response from a web server. Server responses that are error reports (*e.g.*, "404 – Not found") are counted as objects (or pages) in this discussion. We assume that page references normally occur after some period of idle or "think" time at the client, *e.g.*, the time a user spends digesting the contents of one browser display before selecting (or entering) a link to a new page. This same model of a page request following an idle period also captures the behavior of periodically refreshed pages. User actions that complicate this simple view of behaviors such as clicking the browser "stop" or "reload" buttons while a page is loading or a "quick click" on a link before the page loads completely, are discussed in Section 8.

We define an idle period heuristically by examining the time-ordered set of TCP connections used by a client. We identify periods in which the client either has no established TCP connections or where no established connection has an active request/response exchange in progress. We consider a request/response exchange to be active from time the request begins until the corresponding response ends. If any period with no activity persists for longer than a time threshold, it is classified as an idle period. We found empirically that a threshold of 1 second works well for distinguishing idle periods (as did Mah and Barford and Crovella). It is important to note that this approach works only on traces for which we can be reasonably certain that *all* the TCP connections for a given browser appear in the traces. Since the NLANR traces have no information about where clients are located relative to the monitoring point, we perform this analysis only on the UNC traces where we know the clients are located on (or dialed into) the campus network and the servers are located somewhere in the Internet. Figure 1 shows the distribution of idle periods greater than one second observed in the UNC data. There are no appreciable differences between the 1999 and 2000 results; 60% of idle periods are between 1 and 10 seconds and approximately 90% of idle periods are less than 60 seconds.

We consider the initial request/response exchange following an idle period to be for the "top-level" page object (typically HTML) and all the subsequent request/response exchanges before the next idle period to be for the "embedded" object references (if any) within the initial page object. This classification heuristic implies that responses consisting of error status are treated as page objects. Note that top-level or embedded objects that can be used from the browser's local cache are not visible in our traces and Conditional-GET request/response exchanges for cache validation found in the traces are treated as normal page or embedded object references (see Section 8). The server IP address involved in the request/response exchange for the top-level page object is considered to be the *primary* server for the page. All server IP addresses not equal to the primary IP address involved in subsequent request/response exchanges for objects related to that page are considered to be *non-primary* servers. Embedded objects may come from either the primary server or from non-primary servers.

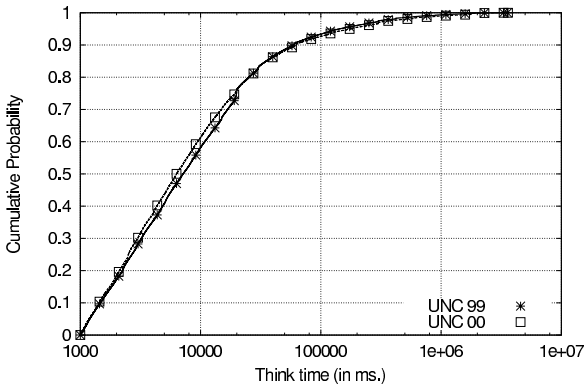

Figure 1: Cumulative idle ("think") time distribution.

6. TCP connection usage

We have implemented a suite of tools for processing *tcpdump* formatted traces that produce statistical data to characterize TCP connection behaviors. We first present results concerning the usage of persistent and non-persistent connections. Our classification of a TCP connection used in HTTP as persistent reflects actual usage characteristics, not whether a browser and server have used the HTTP protocol to establish a persistent connection. Our definition is thus of *effective persistence*, that is, whether or not a TCP connection is actually used for multiple request/response exchanges. A TCP connection is considered persistent if it is actually used for two or more request/response exchanges. All TCP connections used for one request/response exchange are considered non-persistent. This means that TCP connections in which the browser and server have enabled a persistent connection but make only one request/response exchange are considered non-persistent. Partial connections (normally at the beginning and end of traces) and those terminated without any exchanges are not classified.

Table 2 gives summary information about how TCP connections are used in the web. While 15% or fewer of all TCP connections are effectively persistent, they are now used for 40-50% of all object references representing about 40% of all bytes transferred. This means that persistent connections now have a significant influence on the dynamics of TCP connections for the web. Put another way, the number of TCP connections required for web traffic is now approximately 50% lower than it would have been with the original HTTP protocol. Figure 2 shows the distributions of the number of request/response exchanges for persistent connections. Over 60% of persistent connections are used for three or more request/response exchanges and 10% carry more than ten.

Figure 3 shows the distribution of unique TCP connections used in requesting all the objects for a page. Around 55% of all pages are fetched using two or more unique TCP connections (which may be any mix of persistent and non-persistent connections depending on the capabilities of the servers and browsers). Around 50% of all pages required 2-10 unique TCP connections. The number of unique TCP connections used for a

Table 2: Summary data for TCP connections used in the web (all counts in thousands).

	UNC		NLANR	
	99	00	99	00
Top-level Objects	3,722	6,586	N/A	N/A
Embedded Objects	14,799	30,836	N/A	N/A
% Embedded Objects	79.90%	82.40%	N/A	N/A
Non-Persistent Connections	9,620	17,672	261	187
Persistent Connections	1,551	3,152	34	29
% Persistent Connections	13.89%	15.14%	11.60%	13.77%
Unclassified Connections	577	1,513	58	64
% Unclassified	4.91%	6.78%	16.51%	22.79%
Objects on Non-Persistent	9,620	17,672	261	187
Objects on Persistent	8,280	17,497	153	140
% Objects on Persistent	46.26%	49.75%	36.97%	42.83%
Bytes on Non-Persistent	66,522,598	124,665,042	1,490,813	991,967
Bytes on Persistent	45,471,488	84,205,017	680,123	550,485
% Bytes on Persistent	40.60%	40.41%	31.33%	35.69%

page is a result of complex factors including the number of objects in the page, the number of servers holding the objects for the page, the number of concurrent TCP connections the browser opens to each server, whether the client and server support persistent connections, and how aggressive they are about keeping persistent connections active over multiple pages. Figure 4 gives the distribution of unique server IP addresses per page. While about 65% of all pages can be obtained from a single server IP address, about 35% require connections to 2-10 different IP addresses (and rarely as many as 100 IP addresses). We believe these results are a reflection of the ways page content is obtained dynamically from a number of sources including banner ads from agency sites and content that has been explicitly distributed to content servers (*e.g.*, Akamai).

7. Request and Response Data Sizes

We now consider the sizes of individual request and response elements of the HTTP protocol as identified by our analysis tools. Figure 5 shows the distribution of request sizes. Request sizes are defined in our analysis as the number of bytes sent by the browser in a single request/response exchange as explained above. As expected, over 90% of the requests are between 100 and 1,000 bytes in size. The most surprising feature of this distribution, however, is the tail which indicates the presence of some very large requests, especially in the 2000 traces. (Note that the CCDF plot is approximately linear over three orders of magnitude in request size on log scales which is consistent with a heavy-tailed distribution.) We have looked more closely at how these very large requests arise (which are larger than one would expect from submitting forms with web browsers). An interesting example we found was the use of "web email," specifically users of Yahoo email, that send email messages with large attachments. In one trace, a user sent email with an approximately 500K attachment producing a request of that size but eliciting a response from the Yahoo server of only 2.2K bytes. We speculate that as browsers become more widely used as interfaces for web-enabled applications such as email, large request elements will become more significant.

The requests over 1,000 bytes in size already represent a non-trivial contribution to the total bytes transmitted as requests. Figures 6 and 7 give two views of the cumulative percentage of total bytes in requests as a function of request size. These plots clearly show the growing contribution of large requests to the

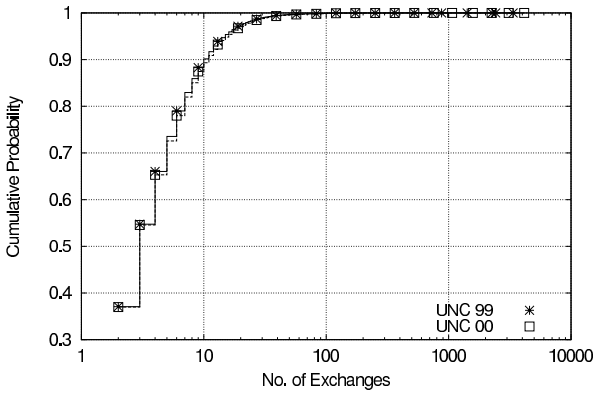

Figure 2: Cumulative distribution of request/ response exchanges per persistent connection.

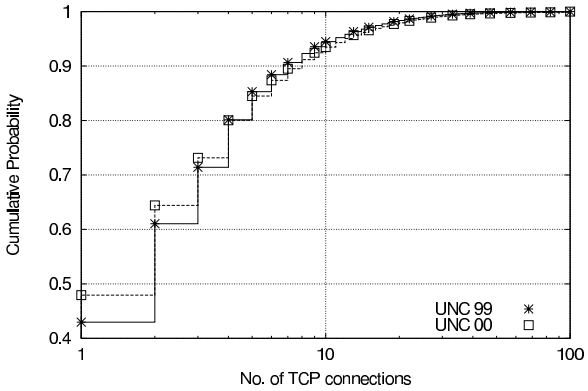

Figure 3: Cumulative distribution of unique TCP connections per page.

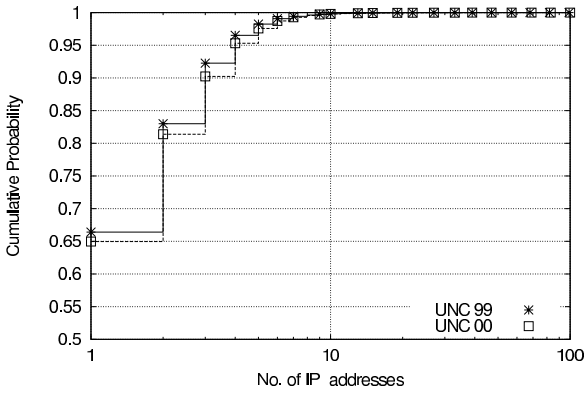

Figure 4: Cumulative distribution of unique server IP addresses visited per page.

total volume of request bytes in the UNC and NLANR traces (*e.g.*, about 20% of the request bytes come from requests larger than 1,000 bytes in the UNC-00 traces). There has been a noticeable change from 1999 to 2000 (5% of the bytes were in requests larger than 10,000 bytes compared to 3% in 1999).

Response sizes are defined in our analysis as the number of bytes sent by the server in a single request/response exchange as explained above. Figures 8 and 9 give two views of the distribution of response sizes. Overall we find that about 85% of the responses observed were 10,000 bytes or less. These dis-

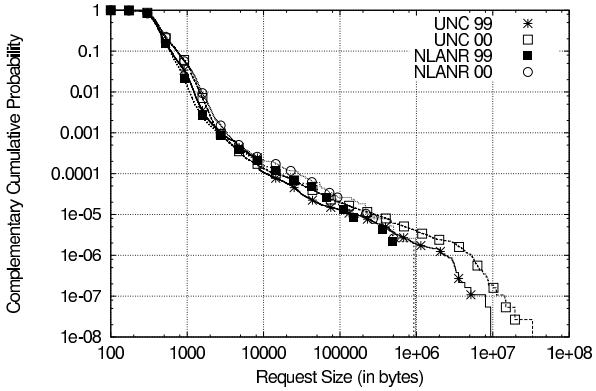

Figure 5: Complementary cumulative distribution of request data sizes greater than 100 bytes.

tributions also indicate some potentially interesting shifts between 1999 and 2000 in the proportions of responses in the range of 200 bytes to 10,000 bytes. In both the UNC and NLANR data, we see some shift to a greater proportion of smaller response objects. For example, in the UNC-99 traces about 47% of responses were 1000 bytes or smaller while in the 2000 traces, about 53% of the responses were 1000 bytes or less. Data from future years will be needed before determining if there is a definite shift in the relative proportions of object sizes in the 200-10,000 byte range.

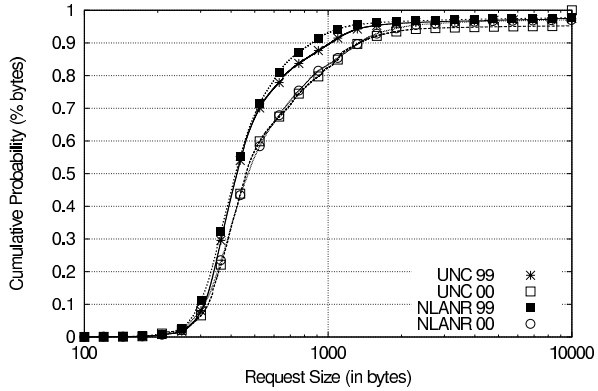

Figure 6: Cumulative distribution of request bytes transmitted weighted by request size less than 10,000 bytes.

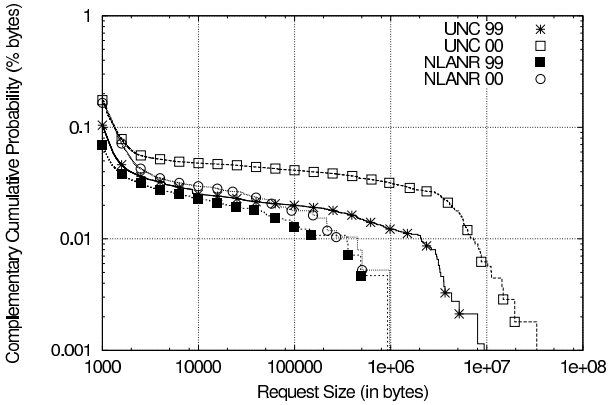

Figure 7: Complementary cumulative distribution of request bytes transmitted, weighted by request size >1,000 bytes.

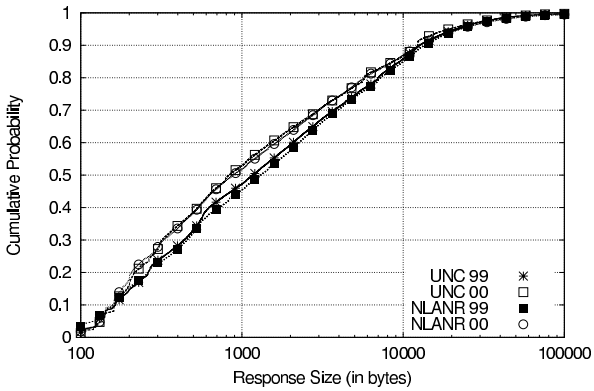

Figure 8: Cumulative distribution of response data sizes (100–100,000 bytes).

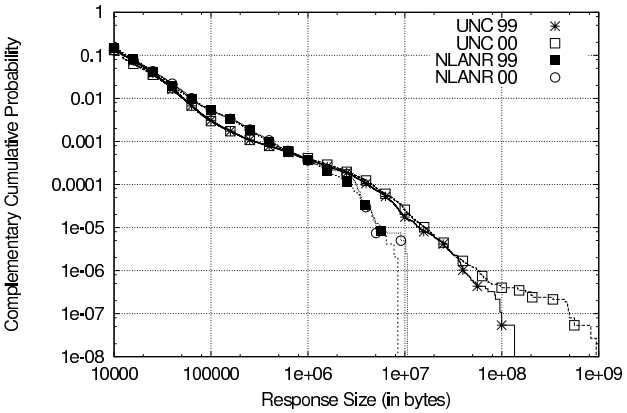

Figure 9: Complementary cumulative distribution of response data sizes greater than 10,000 bytes.

For response sizes greater than 10,000 bytes, we found that the very largest objects increased significantly in size. In the UNC traces the CCDF of response sizes is linear over four orders of magnitude on log scales and is consistent with earlier observations of heavy-tailed distributions [3, 5]. This probably reflects the increasing use of HTTP instead of FTP to distribute large files.[6] The sizes of the very largest responses in the UNC traces are bigger in 2000 by a factor of 4-7 while there appears to be no change for the NLANR traces. The sizes in the NLANR traces are smaller by almost two orders of magnitude. We believe that much of this difference in the NLANR data is not due to real changes in the sizes of responses but is an artifact of the tracing environment. This illustrates an important methodological point — the ability to collect data on very large response objects is influenced by the trace interval and link speeds. In the case of the NLANR traces, the nominal trace interval of 90 seconds is just too short to capture large responses entirely. For the UNC traces, it is likely that tracing for one hour on a gigabit-speed link results in a greater chance of observing larger responses than tracing for one hour on an OC-3 link. It will be interesting to see if the sizes of the very largest responses increase when we trace for eight hours or more on gigabit-speed links.

We found significant differences in the distributions of response sizes for top-level and embedded objects. Figures 10

[6] One specific example is CD images for software distribution (*e.g.*, Linux releases)

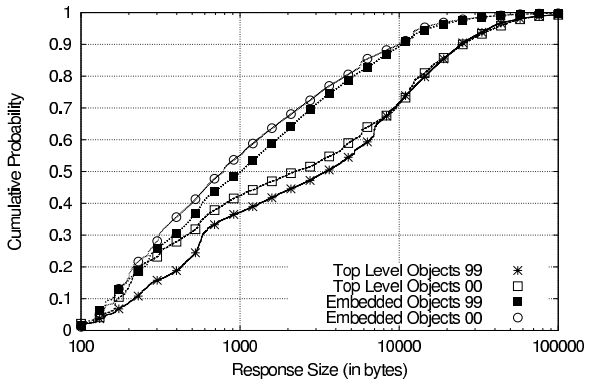

Figure 10: Cumulative distribution of response data sizes for top-level and embedded object sizes (100–100,000 bytes).

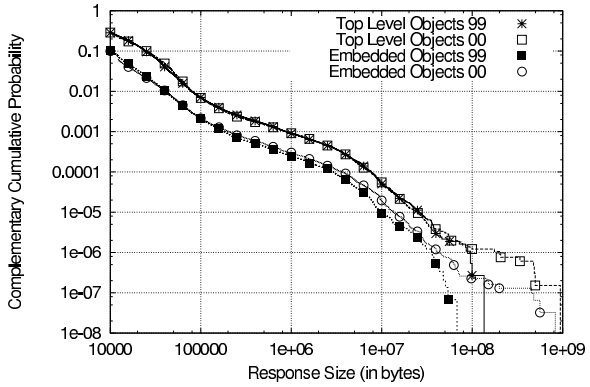

Figure 11: Cumulative distribution of response data sizes for top-level and embedded object sizes greater than 10,000 bytes.

and 11 shows the overall distributions of top-level and embedded object sizes from the UNC traces. We see clear indications that top-level objects tend to be larger than embedded objects. For example, about 30% of top-level objects are larger than 10,000 bytes while only 10% of embedded objects are. About 50% of top-level objects are smaller than 2,000 bytes while 70% of embedded objects are smaller than 2,000 bytes. This observation is consistent with the number of complex web pages we all see that are composed with embedded objects for icons or advertisements. Figure 10 shows indications of a year-to-year trend toward larger proportions of smaller objects in the range of sizes between 100 and 10,000 bytes.

While we have found that 85% of all responses are 10,000 bytes or less, these responses account for only about 20% of the bytes actually transferred from servers to clients, and responses larger than 100,000 bytes represent over 35% of the bytes. Figures 12, 13, and 14 give different views of the cumulative percentage of total bytes in responses as a function of response size. Figure 12 gives the overall contribution of different response sizes to the total bytes returned by servers. The NLANR traces are quite similar to the UNC traces except above 100,000 bytes where the truncated trace intervals limit the observation of large responses. There is a clear influence of the larger response sizes in the 2000 UNC traces when compared to 1999. In Figure 13 we see evidence that larger objects account for a higher proportion of the total bytes transmitted on non-persistent connections. Figure 14 confirms our observation that top-level objects are larger than embedded objects and also indicates embedded objects larger than 10,000 bytes may be

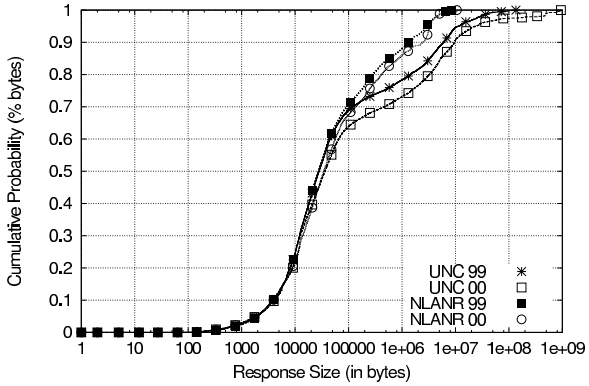

Figure 12: Cumulative distribution of response bytes transmitted weighted by response size.

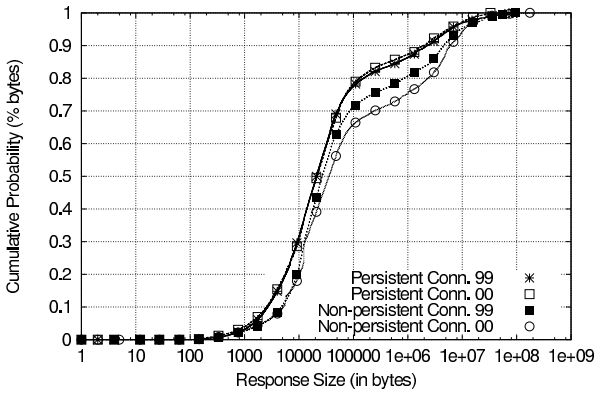

Figure 13: Cumulative distribution of response bytes transmitted weighted by response size for persistent and non-persistent connections.

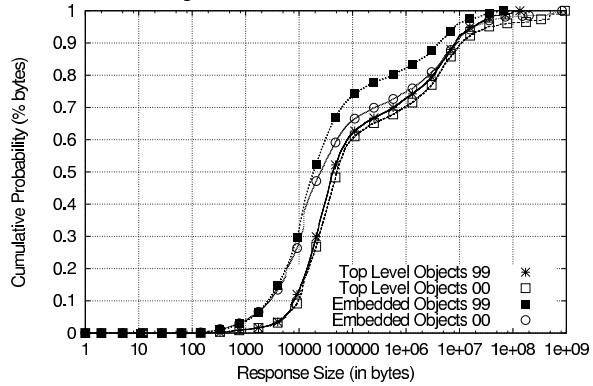

Figure 14: Cumulative distribution of response bytes transmitted weighted by response size for top-level and embedded objects

increasing in size year-to-year. For example, only 10% of the total bytes from top-level objects are from objects smaller than 10,000 bytes while about 30% of the bytes from embedded objects are from those smaller than 10,000 bytes.

8. Limitations of the Methodology

Our methodology is based on making inferences from the limited information available in the TCP/IP protocol header for one direction of a TCP connection (from off-campus server to on-campus browser in our case). There are a number of inherent

limitations and uncertainties that arise when making these inferences.[7] However, the degree of uncertainty in the results is not uniform. For characterizations of TCP connection-level properties such as the number of connections, the sizes and numbers of request/response exchanges, *etc*., the methodology should produce very good results. For other characterizations of the Web, especially those that depend on classifying exchanges as belonging to top-level or embedded references, there is greater uncertainty. We have identified four classes of issues that contribute to uncertainty in the results: pipelined exchanges, user/browser interactions, caches, and TCP segment processing. Each of these is discussed separately below.

Pipelining

Pipelining introduces the possibility of errors in determining the number and lengths of request/response exchanges in TCP connections. In the simple case where a server TCP stack received all the request segments generated in a pipeline before it sent the first response segment, the entire pipeline of requests would be treated as one (larger) request. Similarly, if a server sent all the response segments for a pipelined response before it received segments for a new client request, the entire pipeline of responses would be treated as one (larger) response. This would inflate the sizes of requests and responses and deflate the number of exchanges per connection. In more complicated cases where a pipeline of requests overlapped with (one or more) responses or *vice versa*, the analysis tool would infer the sizes of individual requests and responses incorrectly.

A somewhat dated (1998) study found that pipelining was not implemented in popular browsers [31]. Our observations show that browsers still do not appear to employ pipelining. This was determined through two separate investigations of browser behavior using the current (late 2000) releases of the two browsers that dominate usage on our campus: Netscape Navigator (version 4.7) and Microsoft Internet Explorer (version 5.5). For the first study we configured an Apache web server to support persistent connections and pipelining. A moderately complex web page consisting of a base HTML file with 20 embedded references to objects of various sizes was placed on the server. Using an HTTP server test program, we generated known pipelined requests for the elements of this page and verified that the server did in fact implement persistent connections and pipelining (verified using a *tcpdump* of entire packets flowing in both directions). We then requested the test page with both Navigator and IE. Both browsers opened multiple persistent connections (4 by Navigator, 2 by IE) but neither constructed any pipelined requests (verified by examining the resulting *tcpdump* trace).

To confirm this behavior in more realistic situations with a variety of page structures and server implementations, we did a second study. For this study we requested the site home page and a page one link off the home page from a web server at each of the top-twenty web properties as reported by MediaMetrix [37]. We made these requests with both Navigator and IE and recorded tcpdumps of *entire* packets flowing in *both* directions. We then analyzed the traces using the captured HTTP headers and saw that *no* instances of pipelining could be found even though most pages had numerous embedded objects and at most sites one or more servers supported persistent connections. We also processed the one-way, TCP/IP headers-only subset of these traces with our analysis tools to reconfirm that they correctly identified all request/response exchanges and produced correct results for request and response sizes. Based on these two investigations, we believe it is unlikely that pipelining has any influence on the results reported here.

User/Browser Interactions

User actions at the browser level can effectively interrupt the request/response exchanges for fetching page objects. These interrupting actions include clicking the browser "stop" or "reload" buttons while a page is loading, or a "quick click" on a link displayed before the page loads completely. The Windows versions of Navigator and IE respond to these actions with a *close(abort)* operation for its TCP connections (thus sending a segment with the Reset flag set to the server). When the user does either a "reload" or "quick click" while a page is loading, the browser immediately begins the process of loading a new page. As a result, our page-detection heuristic (based on an idle interval greater than one second) misidentifies the two pages involved in one of these interrupting actions as only one page. The result is that some (presumably larger) top-level objects are counted as embedded objects in the distributions of object sizes by type, and the distributions of unique TCP connections per page and unique server IP addresses visited per page may be somewhat skewed toward larger values. For uses of the "stop" button, however, the latter two distributions may be somewhat skewed toward smaller values since object references are artificially curtailed. Analysis of Reset segments in the TCP/IP headers from our client-to-server traces could potentially quantify the frequency of these user actions and provide a better characterization of the effect on these distributions.

Another user-related source of uncertainty is the somewhat common case of a user having two or more browser windows open at the same time. If the user happens to invoke page activity concurrently in multiple windows (this is expected to be rare), our methodology will not correctly identify top-level versus embedded objects and associate request/response exchanges with a page. This same source of uncertainty applies to cases where multiple users are running browsers concurrently on a compute server (multiple users per IP address).

Caches

Our traces contain information only about pages or embedded objects that could not be satisfied from local browser cache. If any components of a page are in the cache, the inferences derived from the trace have uncertainties that depend on how the browser implements cache validation. For example, if objects are in the cache, our trace may contain HTTP Conditional-GET request/response exchanges for those components. The sizes of these conditional-GET exchanges somewhat obscure actual sizes of objects just as response headers do. (We do, however, produce an accurate measurement of the bytes actually transferred on the TCP connection.)

If the cache contains an expiration or MAX_AGE value for an object (or implements some form of adaptive time-to-live), there may be no activity in the trace for many browser references to objects. This can cause incorrect inferences for identifying the top-level page (*e.g*., if the top-level page is taken from the cache but some embedded object is not, the traced fetch of the embedded object may be inferred to be for a top-level object). While browser caches have no effect on our results for actual TCP connection-level characterizations, they do introduce uncertainty in page-level characterizations that de-

[7] Using TCP/IP headers from both directions of flow in a TCP connection would not substantially improve these uncertainties because they arise from a lack of information (*e.g*. HTTP headers, cache contents, or user actions) not available at this level.

pend on identifying the top-level page versus embedded objects (*e.g.*, distributions of sizes for top-level objects may contain more small objects than they should).

Proxy caches are not in significant use on our campus and do not contribute materially to the uncertainties described for browser caches. Traces from other locations where proxy caches are deployed would be very interesting for comparison purposes.

TCP Segment Processing

As we mentioned in Section 4, retransmissions and segment reordering in the network do not cause problems for the analysis as long as the segments do not contain changes in both data sequence numbers and ACK values. For persistent connections, out-of-order segments carrying both response data and new ACK values can lead to incorrect results. For example, consider two consecutive segments that have advances in both data sequence numbers and ACK values. If these are reordered in the network, the analysis tool, ignoring the smaller values that arrive late, will potentially miss a request/response exchange and report larger values of sizes. Each case of such segment reordering that might lead to erroneous request or response lengths was recorded for off-line investigation. In the UNC traces, we found about 4,900 connections with segment reordering where the analysis tool could not make a correct choice.[8] We examined a random sample of several hundred of these cases and found that less than half were analyzed incorrectly. Thus we believe that a very small percentage (less than 0.005%) of incorrect size values are included in the data and have no appreciable effect on the results.

A final uncertainty arises from the fact that our traces cover specific intervals of time. This means that at the beginnings and ends of traces we find incomplete TCP connections (5-7% of the total connections in the UNC traces[9]). We have excluded from our analysis any request or response size that was made ambiguous by missing the beginning or end of the TCP connection while retaining data about other unambiguous request/response exchanges in the same connection. We also excluded incomplete connections from results concerning persistent versus non-persistent connections. We made no attempt, however, to exclude other connections related to the incomplete connections (*e.g.*, those within the same top-level page). This probably has a small effect on results for the number of exchanges in persistent connections or in characterizations that depend on identifying top-level or embedded objects.

To summarize, in the absence of pipelining, the methodology produces accurate results for TCP connection-level characterizations. For making inferences about the web that rely on classifying request/response exchanges as belonging to top-level or embedded objects, there is greater uncertainty (arising mostly from caching and user/browser interactions). Additional study is needed to evaluate the impact of these effects.

9. Summary and Conclusions

By any measure, web traffic is the single largest identifiable consumer of bandwidth on the Internet. A contemporaneous characterization of web traffic is therefore important for driving network simulations and live testing of network components such as congestion control mechanisms. While previous models of web traffic have been presented in the literature, it is important to continually re-populate these models with new data and update the models to account for the evolution of protocols and their use. We demonstrate the use of a lightweight measurement methodology for gathering these data based on capturing only TCP packet headers flowing from servers to clients. The method balances the tradeoff between ensuring the privacy of users and gathering sufficient data to capture the HTTP protocol dynamics. We have developed a sufficient set of tools for processing multi-gigabyte *tcpdump* files to create distributions of the major structural elements of an HTTP connection. We have conducted two measurement studies of HTTP traffic arriving at the ingress router on the UNC campus during September and October in 1999 and 2000. We also compare our traces to set of similar (but smaller) traces acquired by NLANR during the same periods as our traces.

From the analysis of our trace data, and the NLANR data, we draw a number of conclusions about the evolving nature of web traffic, the use of the web, the structure of web pages and the organization of web servers that provide their content, and the use of packet tracing as a method of understanding the dynamics of web traffic. From a methodology standpoint, we conclude that a substantial and detailed analysis of HTTP request/response exchanges is possible given only the TCP/IP headers of packets sent from servers to clients. Given only a unidirectional trace, one can discern transport protocol phenomena and the effects of higher-level protocol usage such as the use of persistent HTTP connections, as well as application-level phenomena such as the number of embedded objects per web page and the distribution of servers delivering this content. We believe that future traces of web traffic must be considerably longer (on the order of hours) in order to fully capture the tail of the distribution for measures such as the size of responses from servers to clients.

From a protocol usage standpoint, we observed that 15% of all the TCP connections carrying HTTP request/response exchanges are persistent in the sense that they actually carry more than one request/response exchange. These persistent connections deliver 40-50% of all the web objects requested and these objects account for approximately 40% of all the bytes transferred. Thus, although the fraction of connections that are persistent is small, their use represents a 50% reduction in the total number of TCP connections required to deliver web content (compared to a similar environment in which persistent connections are not used).

From an analysis of HTTP responses we see that 65% of all web pages are constructed entirely by responses from a single server while 35% of the pages requested receive content from two or more servers. Close to 70% of the consecutive top-level page references go to an IP address that is different from the address used for the previous top-level page reference possibly representing the impact of large organizations managing their web sites with a "server farm" for load balancing [29]. We also see an increase in the number of embedded objects per web page but a decrease in frequently occurring sizes. This is possibly due to the pervasive use of "banner adds" and icons to decorate pages [29]. Overall the number of bytes transferred to deliver embedded objects is increasing.

[8] The analysis tools were designed for a simple one-pass processing of segments with no internal stack to save segments for analysis of reordering. A more sophisticated stack-based program could deal with these situations.

[9] We found about 20% of the connections in the NLANR traces to be incomplete because of the short (90 second) trace durations.

Overall we see a slight decrease in the frequently occurring sizes of response objects but a marked increase in the size of the largest objects transferred. The largest 15% of object sizes account for 80% of the bytes transferred from servers. From an analysis of HTTP requests we see novel uses of web requests to implement applications such as "web email" that result in a shift in the distribution of request sizes because files are being transferred (*e.g.*, email attachments) in the context of an HTTP request. This also gives rise to HTTP request/response exchanges in which the request is orders of magnitude larger than the corresponding response.

10. References

[1] J. Apisdorf, K Claffy, K. Thompson, and R. Wilder, *OC3mon: Flexible, Affordable, High-Performance Statistics Collection*, Proceedings of INET '97, June 1997. (*http://www.isoc.org/isoc/whatis/conferences/inet/97/proceedings/F1/F1_2.HTM*)

[2] H. Balakrishnan, M. Stemm, S. Seshan, V. Padmanabhan, R. H. Katz, *TCP Behavior of a Busy Internet Server: Analysis and Solutions*, Proceedings of IEEE INFOCOMM '98, March 1998, pp. 252-262.

[3] P. Barford and M. E. Crovella, *Generating Representative Web Workloads for Network and Server Performance Evaluation*, Proceedings of ACM SIGMETRICS '98, 1998, pp. 151-160.

[4] P. Barford and M. E. Crovella, *A Performance Evaluation of HyperText Transfer Protocols*, Proceedings of ACM SIGMETRICS '99, May 1999, pp. 188-197.

[5] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella, *Changes in Web Client Access Patterns: Characteristics and Caching Implications*, World Wide Web, Special Issue on Characterization and Performance Evaluation, Vol. 2, 1999, pp. 15-28.

[6] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, *Advances in Network Simulation*, IEEE Computer, vol. 33 no. 5, May 2000, pp. 59-67.

[7] R. Caceres, P. Danzig, S. Jamin, and D. Mitzel, *Characteristics of Wide-Area TCP/IP Conversations*, Proceedings of ACM SIGCOMM '91, pp. 101-112.

[8] M. Christiansen, K. Jeffay, D. Ott, and F. D. Smith, *Tuning RED for Web Traffic*, Proceedings of ACM SIGCOMM 2000, September 2000, pp. 139-150.

[9] K Claffy, G. Miller, and K. Thompson. *The nature of the beast: recent traffic measurements from an Internet backbone*, Proceedings of INET '98, (*http://www.isoc.org/inet98/proceedings/6g/6g_3.htm*).

[10] W. S. Cleveland, D. Lin, D. X. Sun, *IP Packet Generation: Statistical Models for TCP Start Times Based on Connection-Rate Superposition*, Proceedings of ACM SIGMETRICS 2000, June 2000, pp.166-177.

[11] Crovella, M. and A. Bestavros, *Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes*, IEEE/ACM Transactions on Networking, vol. 5, no. 6, December 1997, pp. 835-846.

[12] C. R. Cunha, A. Bestavros, and M. E. Crovella, *Characteristics of WWW Client-based Traces*, Technical Report TR-95-010, Boston University Computer Science Department, June 1995.

[13] P. Danzig and S. Jamin, *tcplib: A Library of TCP Internetwork Traffic Characteristics*, USC Technical Report USC-CS-91-495, 1991.

[14] P. Danzig, S. Jamin, R. Caceres, D. Mitzel, and D. Estrin, *An Empirical Workload Model for Driving Wide-Area TCP/IP Network Simulations*, Internetworking: Research and Experience, vol. 3, no. 1, 1992, pp. 1-26.

[15] B. M. Duska, D. Marwood, and M. J. Feeley, *The Measured Access Characteristics of World-Wide-Web Client Proxy Caches*, Proceedings of the USENIX Symposium on Internet Technologies and Systems, December 1997, pp. 23-36.

[16] L. Fan, P. Cao, J. Almeida and A. Broder, *Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol*, Proceedings of ACM SIGCOMM '98, pp. 254-265.

[17] A.Feldmann, *BLT: Bi-Layer Tracing of HTTP and TCP/IP*, Proceedings of WWW-9, May 2000. *(http://www.research.att.com/~anja/feldmann/blt_httptrace.abs.html)*

[18] W. Feng, D. Kandlur, D. Saha, K. Shin, *Blue: A New Class of Active Queue Management Algorithms*, University of Michigan Technical Report CSE-TR-387-99, April 1999.

[19] S.D. Gribble and E.A. Brewer, *System Design Issues for Internet Middleware Services*, Proceedings of the USENIX Symposium on Internet Technologies and Systems, December 1997.

[20] B. Mah. *An Empirical Model of HTTP Network Traffic,* Proceedings of IEEE INFOCOM '97, April 1997. (An extended version is at *http://www.ca.sandia.gov/~bmah/Software/HttpModel/*)

[21] G. R. Malan and F. Jahanian, *An Extensible Probe Architecture for Network Protocol Performance Measurement*, Proceedings of ACM SIGCOMM '98, September 1998, pp. 215-227.

[22] S. McCreary, K. C. Claffy, *Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange*, CAIDA Technical Report (*http://www.caida.org/outreach/papers/AIX0005/*).

[23] J. Mogul, *The Case for Persistent-Connection HTTP*, Proceedings of ACM SIGCOMM '95, pp. 299-313.

[24] H. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie, and C. Lilley, *Network Performance Effects of HTTP/1.1, CSS1, and PNG*, Proceedings of ACM SIGCOMM '97, September 1997, pp. 155-166.

[25] T. Ott, T. Lakshman, and L. Wong, *SRED: Stabilized RED*, Proceedings IEEE INFOCOM '99, March 1999, pp. 1346-1355.

[26] V. Paxson, *Empirically Derived Analytic Models of Wide-Area TCP Connections*, IEEE/ACM Transactions on Networking, vol. 2, no. 4, August 1994, pp. 316-336.

[27] V. Paxson, and S. Floyd, *Wide-Area Traffic: The Failure of Poisson Modeling*, IEEE/ACM Transactions on Networking, vol. 3, no. 3, June 1995, pp. 226-244.

[28] V. Paxson, and S. Floyd, *Why We Don't Know How To Simulate The Internet*, Proceedings of the 1997 Winter Simulation Conference, December 1997.

[29] F.D. Smith, F. Hernandez Campos, K. Jeffay, D. Ott, *What TCP/IP Protocol Headers Can Tell Us About the Web (Extended Version)*, *http://www.cs.unc.edu/Research/dirt*.

[30] K. Thompson, G. Miller, and R. Wilder, *Wide-Area Internet Traffic Patterns and Characteristics*, IEEE Network, vol. 11 no. 6, November/December 1997.

[31] Z. Wang and P. Cao, *Persistent Connection Behavior of Popular Browsers*, December 1998. (*http://www.cs.wisc.edu/~cao/papers/persistent-connection.html*)

[32] A. Wolman, *et al.*, *On the Scale and Performance of Cooperative Web Proxy Caching*, Proceedings of ACM SOSP '99, December 1999, pp. 16-31.

[33] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, M. Brown, T. Landray, D. Pinnel, A. Karlin, and H. Levy, *Organization-Based Analysis of Web-Object Sharing and Caching*, Proceedings of the 2nd USENIX Conference on Internet Technologies and Systems, October 1999.

[34] *http://www.caida.org/tools/measurement/coralreef/*

[35] *http://moat.nlanr/net/Traces/*

[36] *http://www.napster.com*

[37] *http://us.mediametrix.com/data/thetop.jsp*