A Case Study of Web Server Benchmarking Using Parallel WAN Emulation

Carey Williamson, Rob Simmonds, Martin Arlitt

Department of Computer Science, University of Calgary,
2500 University Drive NW, Calgary, AB, Canada T2N1N4

Abstract

This paper describes the use of a parallel discrete-event network emulator called the Internet Protocol Traffic and Network Emulator (IP-TNE) for Web server benchmarking. The experiments in this paper demonstrate the feasibility of high-performance WAN emulation using parallel discrete-event simulation techniques on a single shared-memory multiprocessor. Our experiments with an Apache Web server achieve up to 8000 HTTP/1.1 transactions per second for static document retrieval across emulated WAN topologies with up to 4096 concurrent Web/TCP clients. The results show that WAN characteristics, including round-trip delays, packet losses, and bandwidth asymmetry, all have significant impacts on Web server performance, as do client protocol behaviours. WAN emulation using the IP-TNE enables stress testing and benchmarking of Web servers in ways that may not be possible in simple LAN test scenarios.

Key words: Network emulation, Web performance, TCP/IP, WAN emulation

1 Introduction

Network emulation is a hybrid performance evaluation methodology that combines aspects of experimental implementation with aspects of simulation (or even analytical) modeling. This approach has received increasing research attention in recent years [14,16,25], as researchers address large, complex, and challenging Internet performance problems [3,4,6–8,10].

Email address: {carey,simmonds,arlitt}@cpsc.ucalgary.ca
(Carey Williamson, Rob Simmonds, Martin Arlitt).

The network emulation approach offers a flexible, controllable, and reproducible environment for performance experiments. It enables controlled experimentation with end-user applications (e.g., Internet gaming, video streaming) without facing the transient behaviours of the Internet. More importantly, emulation enables experimentation with a wide range of network and workload configurations. As indicated by Nahum *et al.* [25], the properties of a Wide-Area-Network (WAN) environment can have significant impact on Internet protocol behaviours and Web server performance. Testing in a wide range of WAN scenarios can provide greater confidence in the robustness of a server or application prior to Internet deployment [19].

In this paper, we focus on Web server benchmarking using the Internet Protocol Traffic and Network Emulator (IP-TNE) [30]. The IP-TNE is built using a parallel discrete-event simulation (PDES) kernel, enabling high-performance WAN emulation. The IP-TNE provides a detailed simulation model of an arbitrary IP internetwork WAN topology. Internet hosts can send IP packets to other hosts, whether real (on the Internet) or virtual (within the simulated WAN), via the emulator. Similarly, virtual hosts within the emulator can send (real) IP packets to other (real) hosts on the Internet. The translation of packets between real and simulated Internet environments is accomplished through a technique similar to "IP masquerading", carefully implemented to provide high-performance packet reading and writing at Gigabit Ethernet rates [13].

The purpose of this paper is to demonstrate parallel network emulation using the IP-TNE. We use Web server benchmarking as a case study, for three reasons. First, it demonstrates the packet-handling and throughput capabilities of the IP-TNE as a Web server workload generator. Second, it serves to validate prior results [25] highlighting the impacts of WAN conditions on Web server performance. Third, our study demonstrates that WAN emulation using a single computer is feasible for Web server benchmarking.

Figure 1 provides an illustration of our approach to WAN emulation for Web server benchmarking. Rather than following the traditional "centralized" approach in Figure 1(a), or the "shim" approach [25,29] in Figure 1(b), we use the approach in Figure 1(c), wherein the clients themselves are emulated within the IP-TNE. This approach has several advantages. First, it reduces the equipment required in the experimental setup, and eliminates the need for elaborate synchronization of multiple client machines in the experiments. Second, it provides complete control over the client workload: we can model homogeneous or heterogeneous clients, and we can completely specify their HTTP and TCP behaviours [18,22,23]. Finally, this approach provides a fuller demonstration of the performance capabilities of the IP-TNE. Nahum *et al.* [25] argue that the approach in Figure 1(a) is not scalable for Web server benchmarking, and thus recommend the approach in Figure 1(b). We demonstrate via the IP-TNE example that the (more aggressive) approach in Figure 1(c) *is* feasible

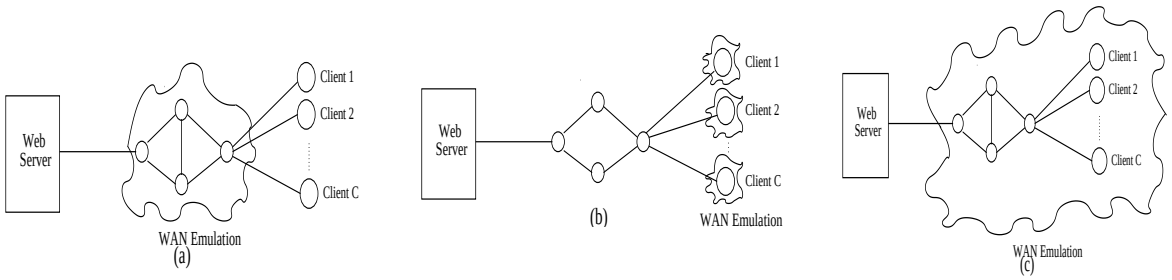


Fig. 1. Possible Approaches to WAN Emulation: (a) Traditional "Centralized" Approach; (b) "Shim" Approach; (c) Our Approach Using IP-TNE

(and scalable enough) for Web server benchmarking. This is one of the main contributions of our paper.

The remainder of this paper is organized as follows. Section 2 briefly discusses related work on Web server performance and network emulation. Section 3 describes the IP-TNE. Section 4 presents the experimental methodology for our WAN emulation experiments, and Section 5 presents the results from our experiments. Finally, Section 6 concludes the paper, and describes ongoing work.

2 Related Work

This section provides a brief discussion of related work on network emulation and Web server benchmarking.

2.1 Network Emulation

IP-TNE is not the first system to provide network emulation. NISTnet [16], DummyNet [29], and ns-2 [14] all provide network emulation functionality. NISTnet works by adding random delays, losses, or reordering to an IP packet stream en route from its source to its destination. Similar functionality is provided by DummyNet, which serves as a "shim" layer in the protocol stack to alter the structure of inbound and outbound packet streams. The ns-2 approach, like IP-TNE, models interactions between packets routed through the emulator and simulated packet events in the emulator. However, the execution-time performance of ns-2 often suffers on large networks.

The primary advantage of the IP-TNE approach to network emulation is fast execution-time performance, due to efficient implementation of the simulation kernel. Furthermore, IP-TNE can run in parallel on a shared-memory multiprocessor. While there are efforts underway to make ns-2 run in parallel [27,28], to the best of our knowledge these projects are aimed primarily at

improving scalability of network models that can be simulated [27], and the inter-operability of multiple simultaneous simulations (e.g., federated simulations [28]). Our focus is on the execution-time performance for modest-sized network models, which are still of practical interest to the users of an emulator. We believe that the IP-TNE offers significant advantages for WAN emulation, which we demonstrate through a "case study" of Web server benchmarking.

2.2 Web Server Benchmarking

There are many tools available for Web server benchmarking: ApacheBench [1], GEIST [20], `httperf` [24], `s-client` [7], SPECWeb [32], SURGE [8], WebBench [34], and WebStone [35], to name a few. These tools are often used on commodity client workstations to generate synthetic request streams to a Web server in a captive test environment. By varying the client workload, these benchmarks can stress different aspects of the Web server implementation, and identify performance bottlenecks.

Many of these Web benchmarking tools are used primarily in local-area network (LAN) environments. A LAN typically offers a dedicated test environment, enabling reproducible and controllable experiments. Such a LAN environment often consists of homogeneous clients, each with high-bandwidth, low-latency access to the Web server.

Many interesting Web server performance problems manifest themselves only in a heterogeneous wide-area network (WAN) environment [3,4,6,10,11]. This observation highlights the need for WAN testing of Web servers for robustness and performance. Nahum *et al.* [25] applied WAN emulation in their WASP (Wide-Area Server Performance) test environment, by using a DummyNet [29] "shim" layer in the protocol stack of the client machines to model WAN delays and packet losses. Their results demonstrate the effects of round-trip delays and packet losses on Web server performance, and identify performance issues that are not apparent in simple LAN test scenarios (e.g., advantages of TCP SACK [25]).

Our work complements that of Nahum *et al.* in several ways. First, we demonstrate that a "centralized" approach to WAN emulation is feasible, through parallel simulation techniques. Second, the IP-TNE provides a more detailed approach to WAN emulation that can more faithfully reproduce wide-area Internet behaviours (e.g., queueing, congestion, packet reordering, packet losses, and IP fragmentation). Third, we confirm several of their observations about the impacts of WAN conditions on Web server performance. Finally, we extend their work by considering several scenarios not explicitly studied in their paper (e.g., heterogeneous RTTs, asymmetric networks, HTTP/1.1).

3 IP-TNE: Internet Protocol Traffic and Network Emulator

The Internet Protocol Traffic and Network Emulator (IP-TNE) [30] is a computer network emulator that uses a fast parallel discrete-event simulation (PDES) kernel called TasKit [37]. This section provides a very brief overview of the IP-TNE and its key components. Additional details regarding IP-TNE and TasKit are provided in earlier papers [13,30,33,37].

3.1 Architectural Overview

IP-TNE is based on an IP network simulator called the Internet Protocol Traffic and Network (IP-TN) simulator, which uses the TasKit parallel simulation kernel [33,37]. IP-TN models network events at the IP packet level. For use with IP-TNE, methods have been added to IP-TN to handle real packet data within the modeled network. Also, TasKit was extended to enable real-time interaction, and an I/O module added to read packets from and write packets to a real network.

Figure 2 shows the main components of the IP-TNE. The IP-TN network simulator (on the left in Figure 2) supports parallel execution on a shared-memory multiprocessor. The middle part of the diagram in Figure 2 shows the I/O module. This module handles the translation of IP packets between the simulated network within IP-TN and the real network (on the right in Figure 2). An address mapping table is used to convert between physical IP addresses on the real network and the internal IP addresses used for endpoint hosts. Each endpoint host represents one physical host on the real network. Only the IP packets sent to and from the endpoint hosts require handling by the emulator; packets exchanged between simulated hosts stay within IP-TN, and packets exchanged between real hosts stay on the real network. A separate thread is used for reading packets from the physical network. On computers with multiple network interfaces a separate thread can be used to read packets from each interface.

3.2 IP Packet Handling

In order to read packets into the simulator, a socket needs to be opened for each interface connecting IP-TNE's host computer to a network. This may be a packet socket or a packet filter socket, depending on the operating system, and requires superuser access privileges on Unix systems. The portable PCAP [26] library could be used for packet reading, though we currently use a simple packet reader that just performs the tasks required by IP-TNE [13].

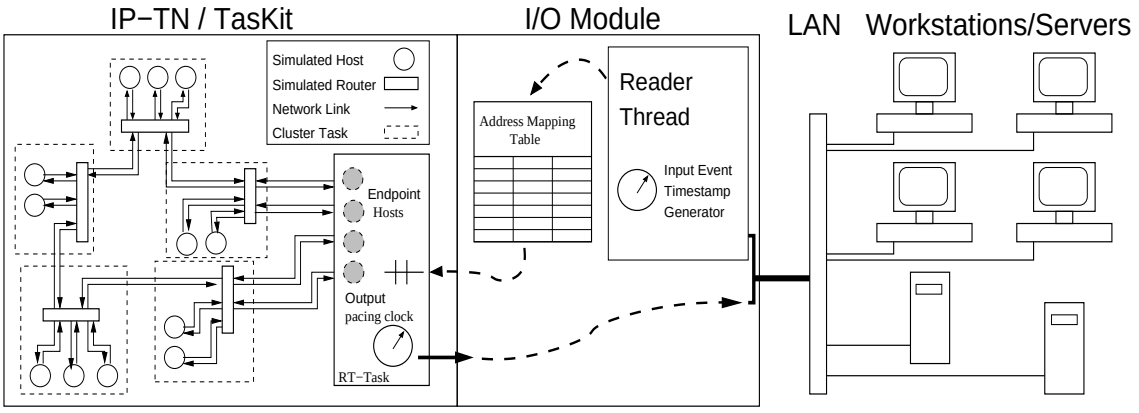


Fig. 2. Architecture of Internet Protocol Traffic and Network Emulator (IP-TNE)

The packet reader can read from interfaces set to promiscuous or non-promiscuous mode. One advantage of running with the interface in promiscuous mode is that a virtual router address and fictitious MAC address can be used. Filtering packets based on the fictitious MAC address can be faster than filtering for packets routed to particular networks. This also avoids the host operating system attempting to route packets if routing is enabled.

Writing packets back to the network requires the use of a raw socket. This allows packet headers to be written directly by IP-TNE rather than being added by the operating system. This is important since IP-TNE needs to make packets appear as if they were from real hosts other than the computer running the emulator. Before a packet is written out, a checksum has to be computed and inserted into the header. For packets that are generated within the emulator, additional checksums may have to be computed and inserted into the header. For example, a TCP packet generated within the emulator requires a TCP checksum to be added to the TCP header.

Further details on the implementation of packet reading appear in an earlier paper [13]. That paper also includes a performance comparison of four possible approaches to packet reading. For the experiments performed for this paper, a kernel packet filter socket was used with the network interface running in promiscuous mode.

3.3 IP Model

The IP-TNE has detailed modeling of the IPv4 protocol. Each host and network interface is assigned its own virtual IP address, to provide the most efficient routing. Within the IP-TNE, the IP packet headers carry virtual addresses in both the IP source address and the IP destination address fields. The endpoint object provides the translation between real IP addresses and virtual IP addresses, using the IP address mapping table. At each router, the

TTL field in the IP packet header is decremented. Packets that traverse networks with different Maximum Transmission Unit (MTU) sizes undergo IP fragmentation, if necessary, at the router between these networks. Checksums are computed for each new fragment generated.

The IP-TNE also supports ICMP (Internet Control Message Protocol) generation. The ICMP error messages (e.g., "host unreachable", "network unreachable") are useful for debugging an emulated WAN topology. The "echo reply", "TTL timeout", and "port unreachable" messages enable the use of `ping` and `traceroute`, which are useful for verifying that the emulated WAN topology is defined and working properly. Also, the "fragmentation required but DF set" message enables path MTU discovery, such as performed by `tracepath`, as well as more complex analysis of the modeled network using `pathchar` [17].

3.4 TCP Model

The TCP clients in the IP-TNE use a socket-based BSD TCP model. The model includes TCP's three-way handshake for connection setup (SYN and SYN/ACK handshake) and close (FIN and FIN/ACK). The client models support full bi-directional exchange of data and ACKs, using TCP's sliding window flow control and congestion control mechanisms (e.g., slow start, congestion avoidance), sequence numbers, acknowledgments (ACKs), and retransmit timers for the detection and retransmission of lost packets. Each host uses a new (sequentially assigned) port number for each new TCP connection. Up to 8 concurrent (parallel) TCP connections from each client are currently supported.

3.5 HTTP Model

The HTTP model in IP-TNE supports both HTTP/1.0 and HTTP/1.1 [9,23]. The model is flexible to allow either open-loop (as in `httperf` [24]) or closed-loop workload generation. In this paper, we use the closed-loop version: upon the successful completion of one HTTP "GET" transaction, clients initiate the next transaction. A random think time (set to 0 in most of our experiments) can be used before initiating the next transaction with the server.

The HTTP/1.1 model provides additional functionality beyond the basic HTTP/1.0 protocol model, such as persistent connections and pipelining [23]. Multiple HTTP transfers are permitted over the same TCP connection, either serially (persistent connection) or in a pipelined fashion (pipelined persistent connection). A parameter to the model specifies the maximum number of HTTP

requests allowed before the TCP connection is closed. Either endpoint can initiate closing of the persistent connection.

For space reasons, the experiments in this paper focus on HTTP/1.1 only. Experiments with a simplified HTTP/1.0 model are described in an earlier (unpublished) technical report [31].

4 Experimental Methodology

This section describes the methodology for the network emulation experiments, including the experimental setup, network model, experimental design, performance metrics, and validation of the IP-TNE.

4.1 Experimental Setup

The experiments in this paper were conducted using two Compaq ES40 enterprise servers. Each ES40 has four Alpha 667 MHz Ev67 processors. Each computer is configured with 4 GB RAM and an 18 GB disk (though the disk is not central to the experiments). The host operating system is Compaq's Tru64 (version 5.1A).

The two Compaq ES40's are directly connected by a dedicated 1 Gbps [1] Ethernet link. The default MTU size for the physical network interface is 9000 bytes (the "jumbo frame" size supported by Gigabit Ethernet), though we set the MTU to 1500 bytes. The TCP clients in IP-TNE negotiate a smaller Maximum Segment Size (MSS) to use as part of TCP connection setup. An MSS of 536 bytes is used in most of our experiments.

The Internet Protocol Traffic and Network Emulator (IP-TNE) runs on one of the Compaq ES40 computers. The IP-TNE is configured to run using between 2 and 4 threads depending on the workload model. One thread is used to read packets from the network interface. The network simulator runs using 1 to 3 threads, with more threads being employed for larger models. These threads handle all packet-level events (e.g., client request generation, packet creation, checksumming, transmission, packet arrival, queueing, routing, ACK generation) as well as handling the job of writing packets out of the emulator.

[1] In this paper, network capacity and network throughput are expressed in bits per second (bps). Note that 1 Kbps = 10^3 bps, 1 Mbps = 10^6 bps, and 1 Gbps = 10^9 bps. Storage sizes, on the other hand, are expressed in bytes (B). For example, 1 kilobyte (KB) = 1024 bytes, 1 Megabyte (MB) = 1,048,576 bytes, and 1 Gigabtye (GB) = 1,073,741,824 bytes.

The Web server in our experiments runs on the other ES40, and makes use of all four available processors. The Web server software is Apache (version 1.3.23), which is widely used on today's Internet [1,25]. We have conducted preliminary experiments with the Flash and TUX Web servers on Linux as well, but for space reasons, we restrict our discussion to the Apache experiments in this paper.

Apache is a process-based HTTP server, which uses child processes to serve incoming HTTP requests. The number of processes is configurable, though processes can be created or destroyed dynamically based on incoming request load. The parent HTTP server process runs as a root-owned process, so that it is not subject to the per-user constraints on the maximum number of processes created or file descriptors used by the server.

There are many configuration parameters for the Apache Web server. These parameters include the minimum and maximum number of processes to create, and the maximum number of requests to dispatch to each child process. MaxClients was set to allow 128 connections to be served simultaneously. MinSpareServers, MaxSpareServers and StartServers were also set to 128 to avoid dynamic process creation during each experiment. MaxRequestsPerChild was set to 0, representing "infinity". These configuration choices were made simply to make results as repeatable as possible; we have not yet had the chance to evaluate the ability of Apache to react dynamically to sudden changes in load. Support for memory-mapped files (`mmap`) was enabled. The Web server was restarted between each experiment.

4.2 Network Model

In this paper, we use the IP-TNE to model an internetwork of Web clients accessing an Apache Web server. For simplicity, we focus on a regular topology that is easy to parameterize and use for workload generation. The network topology definition is generated automatically using a simple script written in the ANML network modeling language [21]. Other network topologies can easily be constructed in a similar fashion.

A representative example of our emulated WAN topology appears in Figure 3. The internetwork consists of N ($1 \leq N \leq 16$) subnetworks, where each subnetwork consists of H ($1 \leq H \leq 1024$) hosts (clients) connected to a switch (S) via a 100 Mbps link. The switch in turn is connected to a router (R) on the backbone network.

The routers along the backbone network are connected in series, as shown in Figure 3 for $N = 4$. The emulated backbone network capacity is 1 Gbps, the same as the physical link capacity to and from the Web server. The leftmost

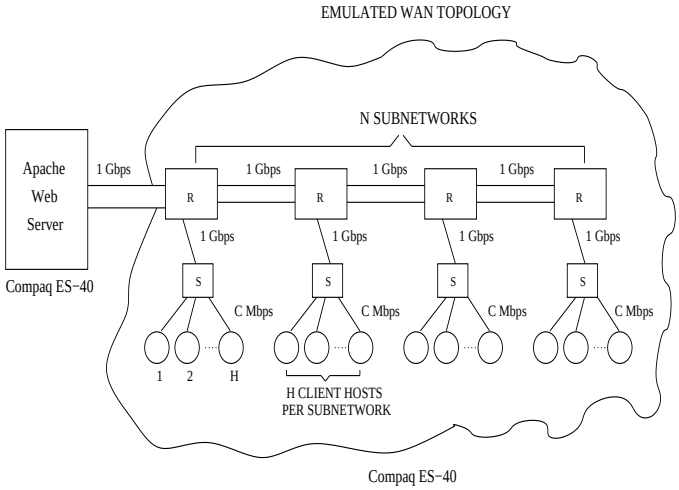


Fig. 3. Emulated WAN Topology for Web Server Benchmarking Experiments

router in the modeled backbone network connects to the external Internet (i.e., the Web server, in our experiments). An endpoint object (not shown in the diagram) is used in IP-TNE to represent this point of attachment to the external network.

The simple, linear topology in Figure 3 makes it easy to control the round-trip time (RTT) for each client in the network. The RTT value has an important influence on the TCP protocol, and thus on Web server performance. The regular topology also makes it easy to control the number of hops between each client and the Web server. Each data packet received by the IP-TNE traverses one or more of the simulated routers on the way to the destination client, and each ACK generated by the client traverses the reverse path on its way to the Web server. The number of simulation events per IP packet sent or received thus increases with the number of subnetworks in the modeled topology.

In all the experiments in this paper, each subnetwork is identical (though the IP-TNE does not require this). That is, each subnetwork has the same number of clients, the same access link capacity, the same low propagation delay (0.1 millisecond) on local links, and the same MTU size. All clients within a subnetwork have the same network access configuration. However, clients in different subnetworks may have different distances from the Web server, and thus different round-trip times. Clients farther from the Web server are at a disadvantage for Web data transfer performance, because of the RTT dependencies in the TCP protocol.

4.3 Experimental Design

We consider seven main factors in our WAN emulation experiments: number of clients, number of subnetworks, link capacity, propagation delay (i.e., round-

Table 1
Experimental Factors and Levels for WAN Emulation Experiments

Factor	Levels
Number of Clients H	1, 2, 4, 8 ... 1024
Number of Subnetworks N	1, 2, **4**, 8, 16
Client Access Link Capacity C (Mbps)	1, 10, **100**, 1000
Link Propagation Delay D (sec)	**0.001**, 0.002, 0.004 ... 0.032
MTU Size (bytes)	296, **576**, 1500
Router Queue Size Q (KB)	2, 4, 8 ... **256**
HTTP Protocol Version	HTTP/1.0, **HTTP/1.1**
Concurrent TCP Connections	**1**, 2, 4, 8
HTTP Requests per Connection	**1**, 2, 4, 8
Pipelining	**no**, yes

trip time), MTU size, router queue size, and HTTP protocol model. Table 1 summarizes the factors and levels used in the experiments. Values shown in bold font are the default values used.

A multi-factor experimental design is used, though only a subset of the experiments are presented in this paper. The number of clients and the number of networks are used to change the workload generated by the IP-TNE. The RTT, link capacity, and MTU factors are used to assess the impacts of different WAN configurations and protocol models on Web server performance. Finally, the router queue size factor is used to assess the impact of packet losses on Web server performance.

4.4 Performance Metrics

The metrics used to quantify the performance of the Apache Web server fall into three categories: server-centric metrics, network-centric metrics, and user-centric metrics. The server-centric metric is HTTP transaction completion rate. The network-centric metrics include network throughput and packet loss rate. The user-centric metrics include mean and median response times for Web document transfers.

Within the IP-TNE, the client models are instrumented to record the timestamps for significant events at the TCP (e.g., SYN, SYN/ACK, FIN) and HTTP layers (e.g., request issued, first byte of response received, last byte of response received), as well as the sizes of requests and responses, including

HTTP header overhead. We also record information regarding the success or failure of TCP and HTTP transactions (e.g., client close, server close, server reset).

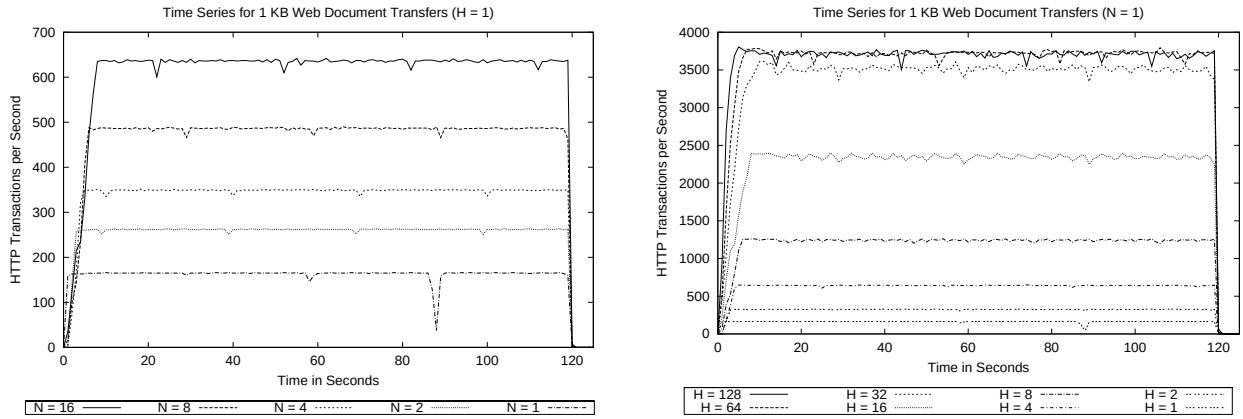
The statistics are stored in memory until the end of the emulation experiment, to avoid undue interactions with the host operating system (e.g., network traffic, disk I/O) [12]. Post-processing of this data is used to determine network throughput, transaction rate, and mean and median document transfer times. A subset of this information is usually available from the Web server access log, but we disabled logging at the server (after our preliminary validation experiments) to maximize Web server performance.

4.5 Validation

The validation of the IP-TNE in our Web benchmarking experiments focused on the functional validation of the Apache Web server, the IP-TNE emulator, and the Gigabit Ethernet network in between them. A primary focus was on validating the performance statistics reported by the IP-TNE. Validation of the IP-TN simulator itself is a separate ongoing process.

Our validation effort involved several steps. First, the `netperf` tool was used to determine the network performance capabilities of the computers in our experimental setup. The TCP_STREAM tests with `netperf` showed that user-user throughputs of 990 Mbps were possible using one processor on each ES40, for large transfers with a 9000-byte MTU and 1 MB send and receive socket buffer sizes. The REQUEST_RESPONSE tests showed that 3900 transactions per second were possible (100-byte request, 1024-byte response, 9000-byte MTU, 600,000-byte send and receive socket buffer sizes, one processor on each ES40). Second, the `ApacheBench` tool was used to assess the Web server's performance. These experiments showed that sustained loads approaching 4000 HTTP/1.0 transactions per second were achievable, for fixed-size 1 KB Web document transfers, using all 4 processors. Finally, the `tcpdump` tool was used, in concert with the IP-TNE, to study the network packet workload generated to and from the emulator. This utility was vital in verifying the correct operation of the TCP and HTTP client models (e.g., three-way handshake, MSS negotiation, TCP options processing, slow-start). Traces were collected for fixed-size 1 KB Web document transfers. These results confirmed the maximum transaction rates identified by the `httperf` and ApacheBench experiments. The network throughput results computed from the IP-TNE client data were consistent with those calculated from the `tcpdump` trace.

These preliminary experiments establish confidence in the operation of the IP-TNE, its Web workload generation process, and the client-side statistics



(a) (b)

Fig. 4. Illustration of Warmup and Steady-State Period for Emulation Experiments: (a) Transaction Rate vs. Time (H = 1); (b) Transaction Rate vs. Time (N = 1)

reported by the IP-TNE.

5 Experimental Results

This section presents the results from our Web server benchmarking experiments using WAN emulation. Section 5.1 discusses preliminary issues regarding the warmup period and run-length for our emulation experiments. Section 5.2 focuses on the impacts of the Web workload model on server performance. Section 5.3 focuses on the impacts of the client model, including concurrent and persistent connections. Section 5.4 focuses on the impacts of WAN conditions on Web server performance. Section 5.5 summarizes our results.

5.1 Preliminaries: Warmup and Run-Length Issues

A preliminary set of experiments was conducted to determine appropriate choices for warmup period and run-length for emulation experiments. These experiments used a simple Web workload model with fixed-size 1 KB document transfers.

Figure 4 presents the performance results for these experiments, using time series plots of the HTTP transaction rate achieved in each one second interval of the experiment. Each line on the graphs presents results for a different number of subnetworks (N) or clients (H) in the emulated WAN topology.

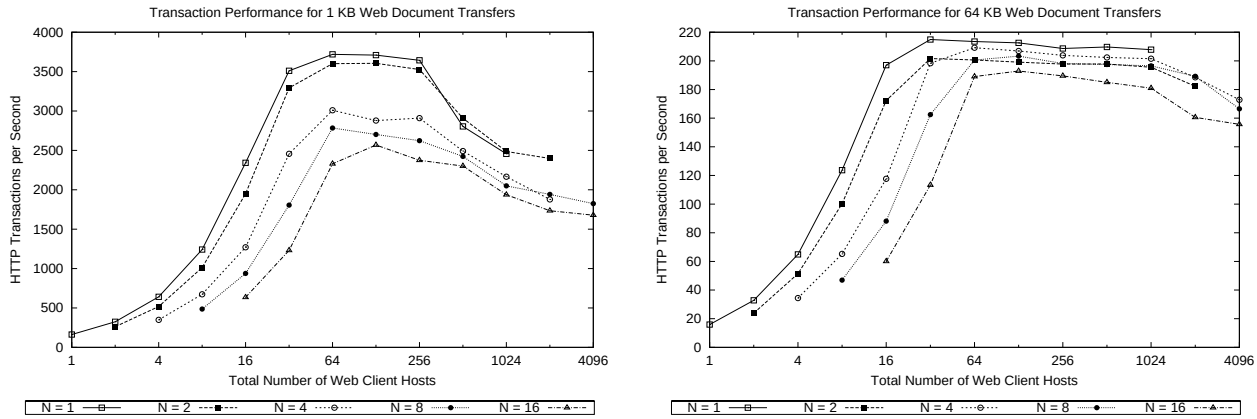
Figure 4(a) shows the HTTP transaction rate achieved using 1 KB transfers when each of the N subnetworks ($1 \leq N \leq 16$) has a single client. The results in this graph represent very light load on the Web server. The transaction rate increases steadily from 160 per second to 640 per second as the number of subnetworks is increased from $N = 1$ to $N = 16$, since the total number of clients increases.

Figure 4(b) shows how the transaction rate changes as the number of clients is increased, for a WAN topology with a single subnetwork ($N = 1$). As the number of clients is increased from $H = 1$ to $H = 128$, the transaction rate increases from 160 per second to 3800 per second. The transaction rate grows linearly with the number of clients, up to $H = 32$, but grows slowly thereafter as the load begins to saturate the HTTP server. The transaction rate is similar for $H = 64$ and $H = 128$, representing server saturation.

The graphs in Figure 4 show that the warmup period required to reach steady-state is brief (about 10 seconds), and that performance is fairly constant [2] beyond this point. These preliminary experiments suggest that a 2-minute duration for emulation experiments is adequate [3] for assessing Web server performance. The remaining experiments in this paper use a 20-second warmup period and 100 seconds for measurement data collection.

[2] Two anomalies are evident in Figure 4(a). The first is the sharp dip at time 86 seconds in the plot for a single host on a single subnetwork. We have traced this anomaly to an occasional packet reordering problem in a particular vendor's network interface card. The packet reordering confuses the TCP client, causing a TCP timeout and retransmission after 800-900 milliseconds, and thus the drop in transaction rate. The throughput dip is only evident for the single client case, since it is unlikely that multiple clients encounter this problem at the same time. The second anomaly is a periodic dip in transaction rates every 30 seconds. Analysis with `tcpdump` shows unexplained delays of 30-40 milliseconds in packet activity every 30 seconds. We suspect either the network card or the `smoothsync_age` parameter in the Tru64 operating system, but have not yet been able to pinpoint this. These anomalies highlight the importance of running experiments long enough (e.g., more than 30 seconds) to get a clear picture of server behaviour. They also highlight the importance of fine-grain measurement of server behaviour (e.g., every 1 second interval).

[3] A shorter run length would eliminate the periodicity anomaly, but could miss other phenomena such as memory leaks or running out of socket file descripters (neither of which we have observed). The current choice of run length allows us to generate all the data points for a graph like Figure 5(a) in about 2 hours.



(a) (b)

Fig. 5. Results for Fixed-Size Transfers: (a) 1 KB Transfers; (b) 64 KB Transfers

5.2 Effect of Web Workload Model

The experiments in this section focus on the performance impacts of the Web workload model used. In particular, the experiments consider fixed-size and variable-size Web document transfers. For simplicity, the experiments in this section allow only a single HTTP transaction on each TCP connection. The discussion of persistent connections is deferred to Section 5.3.

5.2.1 Fixed-Size Transfers

The first set of experiments assesses the HTTP transaction rate for fixed-size Web document transfers. The number of clients (H) and the number of subnetworks (N) are varied, to understand the relationship between the emulated WAN model and the Web server workload.

Figure 5 presents the performance results from these experiments. Figure 5(a) shows the average HTTP transaction rate sustained over the duration of the emulation experiment for 1 KB document transfers, as a function of the total number of clients in the emulated WAN. Each data point in Figure 5(a) is computed as the average transaction rate for the steady-state portion (i.e., ignoring the warmup period) of each experiment, as illustrated in Figure 4. The results in Figure 5(b) are computed similarly, but for 64 KB document transfers. Each line on the graphs presents results for a different number of subnetworks in the emulated WAN topology.

Figure 5(a) shows that a single client ($H = 1$, $N = 1$) can generate a (rather modest) sustained rate of 160 transactions per second to the Web server, for 1 KB transfers. This workload represents a network throughput of approximately 1.7 Mbps, including HTTP protocol overhead. The elapsed time for

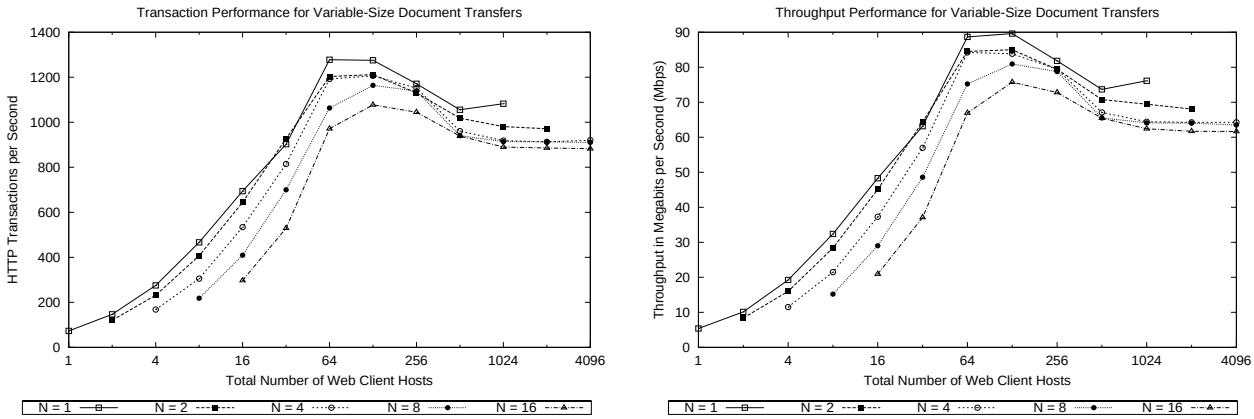
each 1 KB transfer is approximately 6 milliseconds. This value includes the overhead of TCP connection setup and termination for each transfer, plus the request-response HTTP exchange (i.e., three round-trip times of approximately 2 milliseconds each, on our emulated WAN topology). Figure 5(b) shows the corresponding results for 64 KB transfers.

For a single subnetwork ($N = 1$) in the emulated WAN topology, the transaction rate roughly doubles as the number of clients is doubled (note the logarithmic scale on the horizontal axis in these figures), up to about $H = 32$ clients. Beyond this point, the transaction rate flattens, reflecting a saturation of the Web server at a rate of approximately 3800 transactions per second for 1 KB transfers. As the number of clients is increased further, the transaction rate tends to decrease. For larger numbers of emulated subnetworks, a similar trend occurs, with a performance plateau near 64 total clients, and a dropoff in transaction rate after that point. Queueing delay at the server is the main reason for the dropoff in transaction rate: the queueing delay increases the response time for each transaction, which reduces the per-client transaction rate in our closed-loop model. Similar observations apply for 64 KB transfers, where the peak transaction rate achieved is 220 transactions per second, representing a network throughput of 115 Mbps. The dropoff in performance beyond 64 clients is less severe, since queueing delay is a small component of the total transfer time for 64 KB documents.

One additional observation from Figure 5 is that the HTTP transaction rate decreases when a given number of clients is distributed across more and more emulated subnetworks. For example, 16 clients on 1 network produce 2400 transactions per second for 1 KB transfers, while 16 networks each with 1 client produce 600 transactions per second. The reason for this decrease in server and network throughput is a combination of queueing delay and round-trip time effects. The linear topology of the emulated WAN (see Figure 3) means that the average client round-trip time increases as N grows larger. Larger round-trip times increase the latency of a Web transaction; thus there are fewer transactions completed per second by the closed-loop client models. In addition, the linear topology of the emulated WAN tends to produce a bursty request arrival process, particularly if the clients are synchronized initially, and have similar transfer sizes. This burstiness contributes to queueing delays at the server.

5.2.2 Variable-Size Transfers

This section studies the performance of the Apache Web server when presented with a more general Web workload, with variable-size document transfers. The workload for these experiments approximates that observed in real Web server workloads [2]. We model a rather modest Web site with a total of 3000



(a) (b)

Fig. 6. Results for Variable-Size Transfers: (a) Transaction Rate; (b) Network Throughput

Web objects, totalling approximately 30 MB of document storage space. The median document size is 3600 bytes, the mean document size is 9963 bytes, and the largest document size is 1.4 MB. The Web content is allocated in the server file system using a hierarchical directory structure. The Web content is created once, and used repeatedly in all experiments.

The synthetic workload is generated using the ProWGen proxy workload generation tool [15], but appropriately parameterized to model Web server workloads [2] rather than Web proxy workloads. The document sizes are modeled using a log-normal body for the distribution, and a Pareto heavy tail ($\alpha = 1.2$, $k = 10,000$ bytes). File popularity follows a Zipf-like distribution, with a Zipf slope of -0.8. Only static Web content is modeled.

Each client in the emulated WAN generates requests from the aggregate workload file, with clients mapped to requests using a simple modulo arithmetic technique. Each client cycles through its assigned requests continuously during an emulation experiment, with zero think time between requests.

Figure 6 presents the performance results from the experiments with variable-size transfers. Figure 6(a) shows HTTP transaction rates, while Figure 6(b) shows the corresponding network throughput results.

Figure 6(a) has the same general shape as Figure 5, though the vertical scales are different. When the number of subnetworks is small, the HTTP transaction rate increases proportionally with the number of clients in the network, until the Web server is saturated. The primary difference from Figure 5(a) is that the peak occurs near 1200 HTTP transactions per second, rather than 3800 transactions per second for 1 KB transfers. The explanation for this lower rate is the larger average document size in this workload. As a result,

HTTP transactions take longer to complete (on average), and there are fewer completed transactions per second.

Two other observations are evident from Figure 6(a). First, the transaction performance tends to decrease when clients are distributed across more subnetworks. As observed earlier, this effect is due primarily to increasing average round-trip times as the WAN topology grows with N. Second, the dropoff in server and network throughput for the $N = 16$ subnetwork topology is less pronounced than it was for 1 KB transfers. The larger average transfer size is the primary reason for this. In addition, the randomness of transfer sizes reduces the synchronization effects among client requests, leading to more consistent transaction performance as load is increased.

Figure 6(b) shows the corresponding throughput results for the variable-transfer-size experiment. For each number of subnetworks considered, the average network throughput increases initially with the number of clients, and reaches a peak of 90 Mbps. This throughput is roughly double that observed for the experiments with 1 KB transfers, since the average transfer size is larger. TCP can increase its congestion window to achieve higher throughput for larger transfers.

5.3 Effects of Client Model

The experiments in this section focus on the impacts of the client workload model on Web server performance. In particular, we consider the impacts of client think times, concurrent (parallel) TCP connections, persistent connections, and pipelining.

5.3.1 Think Time

A simple experiment was conducted to assess the impact of client think times on the HTTP request workload. Intuitively, the use of a random think time between HTTP requests will simply reduce the effective workload generation rate from each client, thus requiring a larger number of clients to drive the Web server to full load.

Figure 7 confirms this intuition. This experiment uses 1 KB transfers for clients with a single HTTP transaction per TCP connection, on a WAN topology with $N = 4$ subnetworks. Exponentially distributed think times are used, with a mean of 1 second. For comparison, the results for the default (zero think time) model are also shown. These results demonstrate that the use of a non-zero think time between requests dramatically reduces the client request generation rate. For this reason, we use zero think time in remaining experiments to

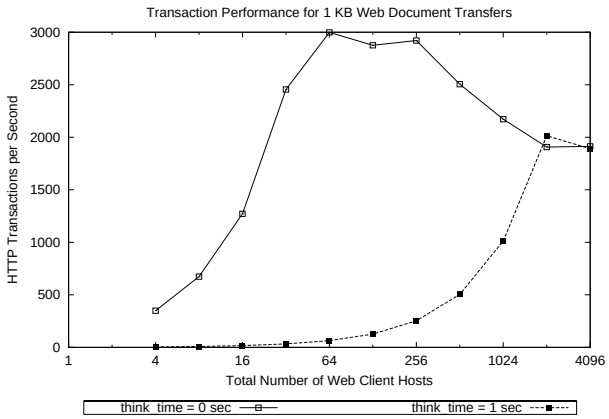


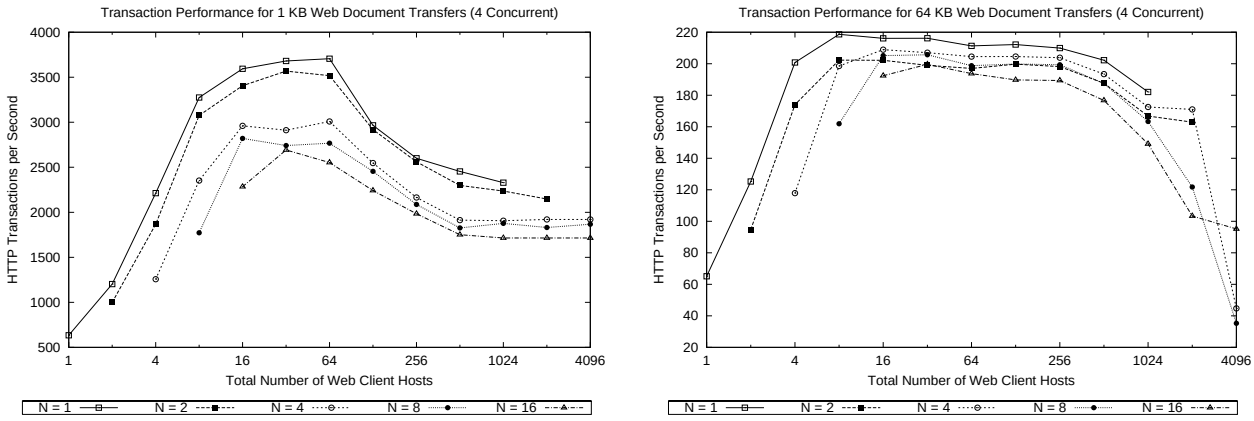
Fig. 7. Effect of Think Time in Client Model

maximize the workload generated per client. This choice means that we may overestimate server performance in some scenarios, particularly with persistent connections. However, this design choice simplifies our presentation of results.

5.3.2 Concurrent Connections

The TCP client models in IP-TNE allow multiple TCP connections to be used simultaneously between the same client and server. This support for concurrent (parallel) TCP connections is similar to that provided in most Web browsers. A parameter in our model determines the maximum number of concurrent TCP connections allowed per client.

Figure 8 shows the performance results for 4 concurrent connections per client, for fixed-size transfers. Compared to the baseline case with a single TCP connection per client (Figure 5), the results in Figure 8 show that there is a general leftward shift of the performance curves. In other words, the peak transaction rate is achieved with fewer clients than previously, since each client has multiple concurrent TCP connections. For a single host on a single network ($H = 1$, $N = 1$), the transaction rate has increased by about a factor of 4 (e.g., from 160 transactions per second to 600 transactions per second, for 1 KB transfers). However, the peak throughput for the server is the same as before: about 3800 transactions per second for 1 KB transfers, and 220 transactions per second for 64 KB transfers. These results show (as expected) that adding concurrent TCP connections to our client models is essentially equivalent to increasing the number of clients per subnetwork in the emulated WAN topology.



(a) (b)

Fig. 8. Results for Concurrent TCP Connections: (a) 1 KB Transfers; (b) 64 KB Transfers

5.3.3 Persistent Connections

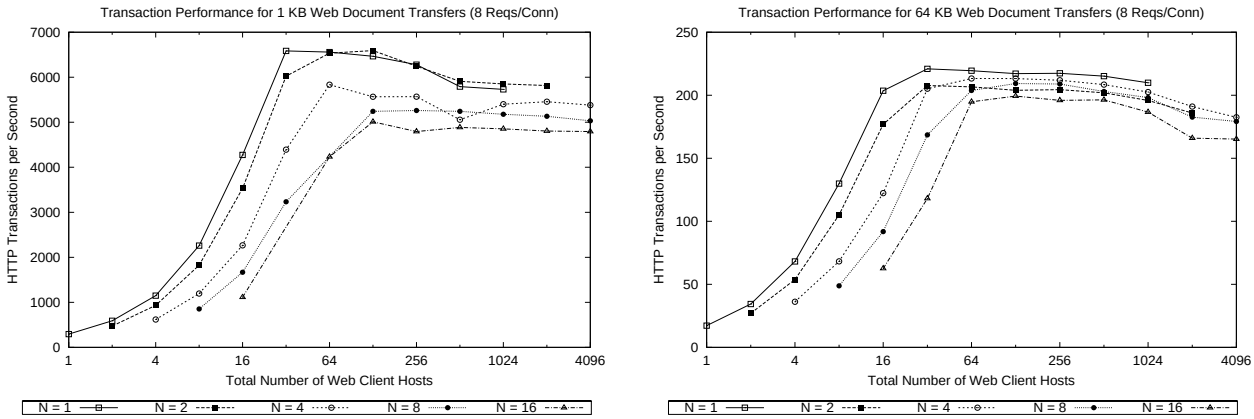
The next experiment varies the number of HTTP requests that can be sent on a TCP connection before it is closed. The default in prior experiments was one HTTP transaction per TCP connection. We now increase this limit, though we maintain the restriction that the requests are sent serially (i.e., one after the other, as HTTP transactions complete). For simplicity, we consider only fixed-size transfers.

Figure 9 presents the results when 8 requests are allowed on each persistent connection. The results here show a dramatic increase in performance for 1 KB transfers: transaction rates and network throughputs are 70% higher than in the baseline case in Figure 5. The peak transaction rate is approximately 6500 transactions per second, and performance degrades slowly as the number of clients continues to increase. Because the use of persistent connections avoids the repeated setup and close of TCP connections, clients are able to achieve much higher transaction rates for 1 KB transfers. There is no significant performance advantage to persistent connections for 64 KB transfers, since TCP connection setup time is a small component of the total transfer time.

5.3.4 Pipelining

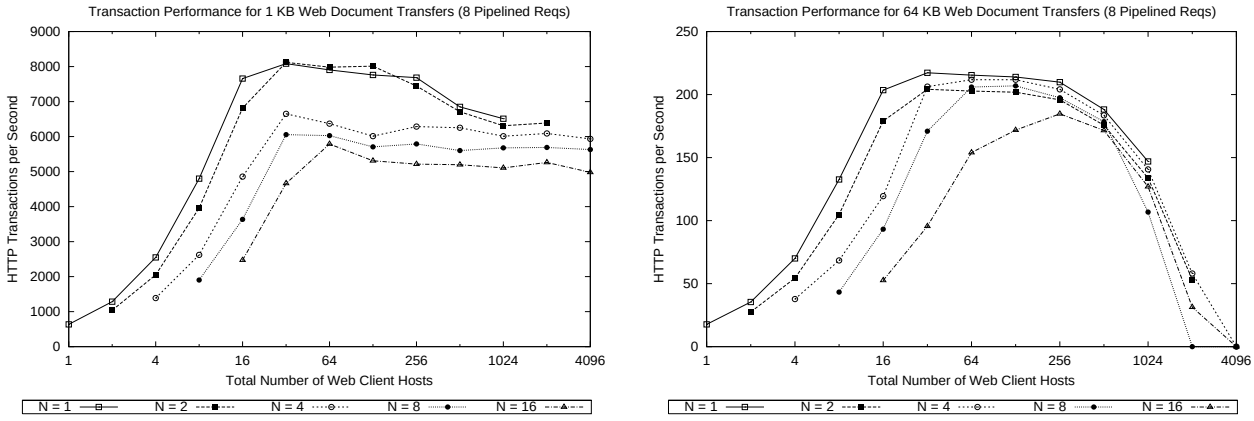
The next experiment considers the pipelining feature of HTTP/1.1. Multiple requests can be sent on the same TCP connection before it is closed, and the requests can be sent asynchronously with respect to the responses. Again, we only consider fixed-size transfers.

Figure 10 presents the results for 8 pipelined requests on each persistent connection. The results here show a further increase in performance for 1 KB



(a) (b)

Fig. 9. Results for Persistent Connections: (a) 1 KB Transfers; (b) 64 KB Transfers

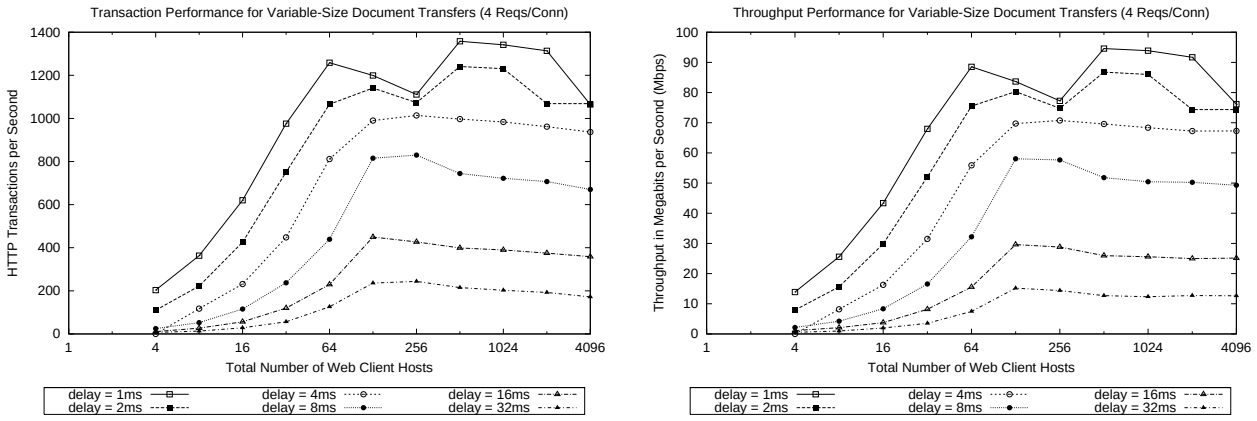


(a) (b)

Fig. 10. Results for Pipelined Persistent Connections: (a) 1 KB Transfers; (b) 64 KB Transfers

transfers: transaction rates and network throughputs are roughly double those from the baseline case in Figure 5. While the qualitative shape of the curves in Figure 10 is the same as before, the peak transaction rate for 1 KB transfers is approximately 8000 transactions per second. The corresponding network throughput is 80 Mbps.

These results show that the pipelining feature of HTTP/1.1 offers significant performance advantages, particularly for short transfers across large WAN topologies. The pipelining feature lessens the RTT dependencies in the closed-loop workload model, ameliorating the dropoff in server and network throughput for 1 KB transfers as the number of clients is increased. The performance benefits of pipelined persistent connections for 64 KB transfers are negligible. In fact, pipelined connections degrade performance when the number of clients is large.



(a) (b)

Fig. 11. Effect of WAN Propagation Delay: (a) Transaction Rate; (b) Network Throughput

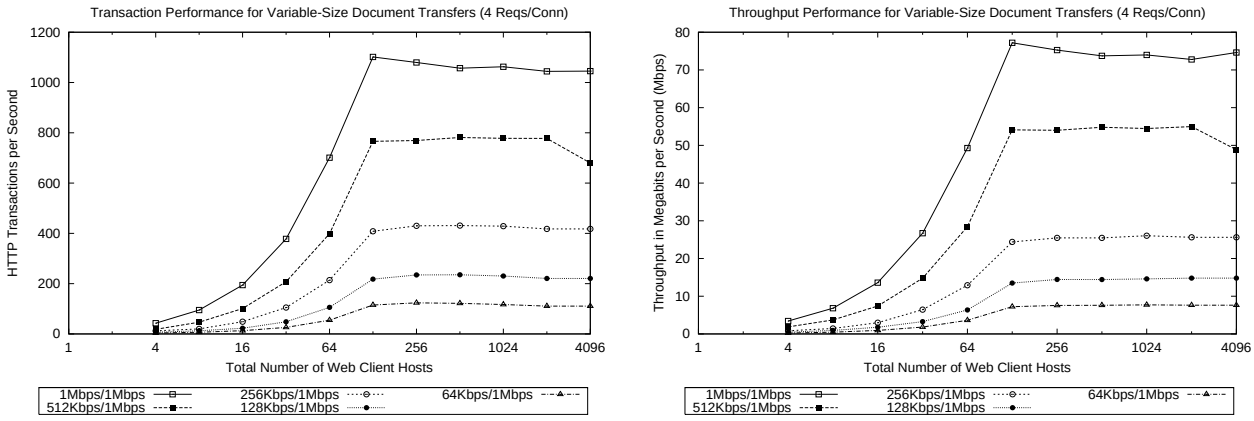
5.4 Effects of WAN Characteristics on Web Server Performance

This section studies the performance of the Apache Web server as different characteristics of the emulated WAN topology are changed. In these experiments, clients use persistent connections with 4 HTTP requests per TCP connection. The Web workload with variable-size transfers is used.

5.4.1 Effect of Round-Trip Time

The first WAN experiment focuses on the impact of round-trip time (RTT) on Web server performance. We choose the WAN topology with $N = 4$ subnetworks (Figure 3) as a representative example, and vary the propagation delay between subnetworks to study the impact on performance. Six values of link propagation delay are considered: 1, 2, 4, 8, 16, and 32 milliseconds. For the modeled WAN topology with $N = 4$ subnetworks, this results in client base RTT values ranging from 2 milliseconds to 256 milliseconds.

Figure 11 presents the results from this experiment. As the link propagation delay is increased, the curves for transaction rate and network throughput tend to move downward, as expected. Again, the results show that the peak transaction rate achieved depends on the round-trip time in the emulated WAN, and that the number of clients at which the transaction rate is maximized depends on the RTT as well. In general, more clients are needed to drive the Web server to its capacity as the RTT increases. These results are similar to those reported by Nahum *et al.* [25], and are not discussed further here.



(a) (b)

Fig. 12. Effect of Network Asymmetry: (a) Transaction Rate; (b) Network Throughput

5.4.2 Effect of Network Asymmetry

The next experiment focuses on the impact of bandwidth asymmetry [5] at the client network access point, for the simple $N = 4$ subnetwork topology. This scenario is constructed to model an ADSL (Asymmetric Digital Subscriber Line) network. The downstream link capacity (to the client) is 1 Mbps, while the upstream link capacity (from the client) is varied from 64 Kbps to 1 Mbps.

The asymmetric configuration is of interest since in some scenarios the upstream link can limit TCP performance [5]. While the packet transmissions by our TCP clients are simply HTTP requests and TCP ACKs, which should not stress the upstream link capacity, the delay or loss of an ACK on the reverse channel can impede TCP congestion window growth for the server [5]. For example, with an upstream link capacity of 64 Kbps, the *normalized bandwidth ratio* [5] is 2.4 (i.e., if one 40-byte TCP ACK is sent for every 2.4 1500-byte TCP data packets received, then the upstream and downstream links will be "equally busy").

Figure 12 presents the results from this experiment. The results show that network asymmetry can limit Web server performance. For the asymmetric network cases, the overall throughput of the Web server decreases. The throughput drop is most pronounced for the asymmetric scenario with the 64 Kbps upstream link: the transmission and queueing delays for ACKs on this path impede TCP transfer performance, leading to longer transfer times and fewer transaction completions per second. These results indicate that bandwidth asymmetry in a WAN can have a substantial impact on Web server performance.

Table 2
Packet Loss Results for IP-TNE WAN Emulation Experiments ($N = 4$)

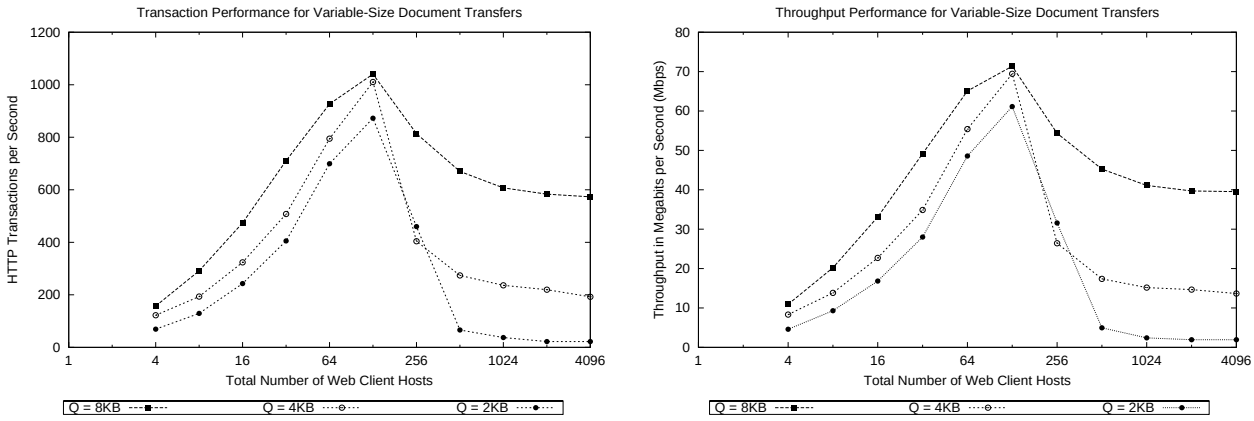
Total Number of Client Hosts	Router Queue Size		
	2 KB	4 KB	8 KB
4	0.2413%	0.0666%	0.0013%
8	0.2218%	0.1444%	0.0241%
16	0.2375%	0.2245%	0.0448%
32	0.2662%	0.2769%	0.1316%
64	0.2920%	0.3400%	0.2850%
128	0.4113%	0.5295%	0.6459%
256	0.9506%	1.9817%	0.9182%
512	6.3393%	2.2179%	1.1224%
1024	10.0216%	2.3915%	1.1355%
2048	12.4610%	2.4845%	1.0854%
4096	18.3911%	2.4829%	1.1111%

5.4.3 Effect of Packet Losses

The final experiment focuses on the impacts of packet losses on Web server performance, for the WAN topology with $N = 4$ subnetworks. Rather than define a link-level packet error model, we control packet loss by changing the router queue size between the Web server and the emulated WAN topology. For this experiment only, we set the backbone link capacity in the WAN topology to 100 Mbps, and study packet losses at the outbound interface of the router between the 1 Gbps physical link and the emulated 100 Mbps WAN backbone link. This scenario produces temporally correlated packet drops at the router, similar to the packet loss patterns observed in the Internet [36].

Table 2 summarizes the packet loss results observed for the different router queue sizes considered in our experiments (2 KB to 8 KB). Packet loss rates range from 0-18%.

Figure 13 presents the performance results from the WAN packet loss experiment. As the router queue size is decreased, the overall level of packet loss increases (see Table 2), and the HTTP transaction rate drops, as does the network throughput. The mean and median client response times increase when packet loss occurs, since timeouts and retransmissions are required at the TCP layer to recover from lost packets, increasing document transfer time. As a result, transfers take longer, and the closed-loop client model produces lower server and network throughput.



(a) (b)

Fig. 13. Effect of WAN Packet Losses: (a) Transaction Rate; (b) Network Throughput

Packet losses have a dramatic impact on TCP performance, and thus on the workload seen by the Web server. These observations are consistent with those of Nahum *et al.* [25], though the packet losses in our experiments are generated differently (i.e., router queue overflows instead of stochastic packet loss models). Our model produces temporally correlated losses both within and across TCP connections, improving upon the loss models in [25].

5.5 Summary of Results

This section presented our WAN emulation experiments for Web server benchmarking. The WAN emulation experiments demonstrate the importance of WAN characteristics, such as round-trip times and packet losses on Web server performance. Several of our results confirm those reported by Nahum *et al.* [25], though our results were produced using a different experimental approach. We thus provide independent validation of several of their observations about wide-area Web server performance. These observations are augmented with our new results regarding the impacts of HTTP/1.1, and the asymmetric network case. All of these factors can have significant impacts on Web server performance in a wide-area network.

6 Conclusions

This paper has demonstrated the use of IP-TNE, a parallel discrete-event IP network emulator, for Web server benchmarking. This work demonstrates that a "centralized" approach to WAN emulation is feasible for Web server perfor-

mance testing. The IP-TNE emulator, using a single physical machine and network interface, can generate adequate client workload to stress a production-quality Web server, while also modeling the packet-level events required for high-fidelity WAN emulation. The performance capabilities of the IP-TNE come from a simulation kernel design that is optimized for parallel execution on shared-memory multiprocessors, and from efficient mechanisms for packet reading and writing at Gigabit Ethernet rates.

This work also reinforces prior observations that wide-area network conditions have an important impact on Internet server performance [25]. We demonstrate the impacts of propagation delays, bandwidth asymmetry, and packet losses on Web server performance. We quantify these results using server-centric and network-centric metrics. The results highlight the importance of WAN testing, or at least WAN emulation, in Web server benchmarking.

Ongoing work focuses on larger-scale validation of the IP-TNE, and on its use in a geographically-distributed experimental Internet testbed. We believe that the IP-TNE offers immense potential for WAN emulation and evaluation of network applications, including Web servers, Web proxies, media streaming, wireless services, and Internet gaming applications. Our "to do" list of future work includes testing with higher performance Web servers (e.g., Apache 2.0, Flash, TUX) to find the upper bound of IP-TNE's capability, and conducting experiments with open-loop workloads, dynamic content, and SURGE. Use of the IP-TNE for robustness testing (e.g., Denial-Of-Service attacks) is also being considered.

Acknowledgements

Financial support for this research was provided by iCORE (Informatics Circle of Research Excellence) and ASRA (Alberta Science and Research Authority) in the Province of Alberta, and by the Natural Sciences and Engineering Research Council of Canada, through NSERC Research Grant OGP0121969. The authors are grateful to the anonymous reviewers for providing constructive feedback on earlier versions of this paper.

The network emulation experiments described in this paper would not have been possible without the Internet Protocol Traffic and Network Emulator (IP-TNE) developed by the TeleSim research group at the University of Calgary. Many people contributed to the design and implementation of this emulator, including Rob Simmonds, Russell Bradford, Brian Unger, Roger Curry, Cam Kiddle, Mark Fox, Kitty Wong, and others. For these contributions, we are grateful.

References

[1] Apache Software Foundation, `www.apache.org`

[2] M. Arlitt and C. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications", *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5, pp. 631-645, October 1997.

[3] M. Aron and P. Druschel, "TCP Implementation Enhancements for Improving Web Server Performance", Technical Report TR99-335, Rice University, July 1999.

[4] H. Balakrishnan, S. Seshan, M. Stemm, and R. Katz, "Analyzing Stability in Wide-Area Network Performance", *Proceedings of ACM SIGMETRICS Conference*, Seattle, WA, pp. 2-12, June 1997.

[5] H. Balakrishnan, V. Padmanabhan, and R. Katz, "The Effect of Asymmetry on TCP Performance", *ACM Journal of Mobile Networks and Applications* (MONET), Vol. 4, No. 3, pp. 219-241, 1999.

[6] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements", *Proceedings of IEEE INFOCOMM Conference*, pp. 252-262, San Francisco, CA, March 1998.

[7] G. Banga and P. Druschel, "Measuring the Capacity of a Web Server Under Realistic Loads", *World Wide Web Journal*, Vol. 2, No. 1, pp. 69-83, May 1999.

[8] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", *Proceedings of ACM SIGMETRICS Conference*, Madison, WI, pp. 151-160, June 1998.

[9] P. Barford and M. Crovella, "A Performance Evaluation of Hyper Text Transfer Protocols", *Proceedings of ACM SIGMETRICS Conference*, Atlanta, GA, pp. 188-197, May 1999.

[10] P. Barford and M. Crovella, "Measuring Web Performance in the Wide Area", *ACM Performance Evaluation Review*, Vol. 27, No. 2, pp. 35-46, September 1999.

[11] P. Barford and M. Crovella, "Critical Path Analysis of TCP Transactions", *Proceedings of ACM SIGCOMM Conference*, Stockholm, Sweden, pp. 127-138, September 2000.

[12] S. Bellenot and M. DiLoreto, "Tools for Measuring the Performance and Diagnosing the Behavior of Distributed Simulations Using TimeWarp", *Proceedings of the SCS Multi-Conference on Distributed Simulation*, pp. 145-149, 1989.

[13] R. Bradford, R. Simmonds, and B. Unger, "Packet Reading for Network Emulation", *Proceedings of IEEE MASCOTS Conference*, Cincinnati, OH, pp. 150-157, August 2001.

[14] L. Breslau *et al.*, "Advances in Network Simulation", *IEEE Computer*, Vol. 33, No. 5, pp. 59-67, May 2000.

[15] M. Busari and C. Williamson, "ProWGen: A Synthetic Workload Generation Tool for Simulation Evaluation of Web Proxy Caches", *Computer Networks Journal*, Vol. 38, No. 6, pp. 779-794, June 2002.

[16] M. Carson, NISTnet. Available at http://snad.ncsl.nist.gov/

[17] A. Downey, "Using pathchar to Estimate Internet Link Characteristics", *Proceedings of ACM SIGCOMM Conference*, Cambridge, MA, pp. 241-250, August 1999.

[18] S. Floyd, "A Report on Recent Developments in TCP Congestion Control", *IEEE Communications*, Vol. 39, No. 4, pp. 84-90, April 2001.

[19] X. Huang, R. Sharma, and S. Keshav, "The ENTRAPID Protocol Development Environment", *Proceedings of IEEE INFOCOMM*, March 1999.

[20] K. Kant, V. Tewari, and R. Iyer, "GEIST: A Generator of E-Commerce and Internet Server Traffic", *Proceedings of IEEE ISPASS*, Tucson, AZ, November 2001.

[21] C. Kiddle, R. Simmonds, D. Wilson, and B. Unger, "ANML: A Language for Describing Networks", *Proceedings of IEEE MASCOTS Conference*, Cincinnati, OH, pp. 135-141, August 2001.

[22] B. Mah, "An Empirical Model of HTTP Network Traffic" *Proceedings of IEEE INFOCOM*, April 1997.

[23] J. Mogul, "The Case for Persistent Connection HTTP", *Proceedings of ACM SIGCOMM Conference*, Cambridge, MA, August 1995.

[24] D. Mosberger and T. Jin, "httperf: A Tool for Measuring Web Server Performance", *ACM Performance Evaluation Review*, Vol. 26, No. 3, pp. 31-37, December 1998.

[25] E. Nahum, M. Rosu, S. Seshan, and J. Almeida, "The Effects of Wide-Area Conditions on WWW Server Performance", *Proceedings of ACM SIGMETRICS Conference*, Cambridge, MA, pp. 257-267, June 2001.

[26] PCAP Packet Capture Library, `www.tcpdump.org`

[27] G. Riley, M. Ammar, and R. Fujimoto, "Stateless Routing in Network Simulations", *Proceedings of IEEE MASCOTS Conference*, San Francisco, CA, pp. 524-531, August 2000.

[28] G. Riley, R. Fujimoto, and M. Ammar, "A Generic Framework for Parallelization of Network Simulations", *Proceedings of IEEE MASCOTS Conference*, College Park, MD, pp. 128-135, October 1999.

[29] L. Rizzo, "Dummynet: A Simple Approach to the Evaluation of Network Protocols", *ACM Computer Communication Review*, Vol. 27, No. 1, pp. 31-41, January 1997.

[30] R. Simmonds, R. Bradford, and B. Unger, "Applying Parallel Discrete Event Simulation to Network Emulation", *Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS)*, Bologna, Italy, pp. 15-22, May 2000.

[31] R. Simmonds, C. Williamson, R. Bradford, M. Arlitt, and B. Unger, "Web Server Benchmarking Using Parallel WAN Emulation", http://www.cpsc.ucalgary.ca/~carey/papers/iptne.pdf
(A short 2-page poster version of this paper is to appear in ACM SIGMETRICS 2002).

[32] Standard Performance Evaluation Corporation, `www.spec.org`

[33] B. Unger, Z. Xiao, J. Cleary, J. Tsai, and C. Williamson, "Parallel Shared-Memory Simulator Performance for ATM Network Scenarios", *Transactions on Modeling and Computer Simulation (TOMACS)*, Vol. 10, No. 4, pp. 358-391, October 2000.

[34] WebBench, `www.etestinglabs.com/benchmarks/webbench/webbench.asp`

[35] WebStone, `www.mindcraft.com/webstone`

[36] M. Yajnik, S. Moon, J. Kurose, and D. Towsley, "Measurement and Modeling of the Temporal Dependence in Packet Loss", *Proceedings of IEEE INFOCOM*, New York, NY, March 1999.

[37] Z. Xiao, B. Unger, R. Simmonds, and J. Cleary, "Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation", *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS)*, Atlanta, GA, pp. 20-28, May 1999.