Web Server Benchmarking Using Parallel WAN Emulation

November 8, 2001

Abstract

This paper describes the design, implementation, and use of a parallel discrete-event network emulator called the Internet Protocol Traffic and Network Emulator (IP-TNE). This emulator can be used for Wide-Area-Network (WAN) emulation experiments in the evaluation of Internet protocols and applications.

In this paper, Web server benchmarking is used as a case study to demonstrate the capabilities of the IP-TNE. There are two reasons for this choice of application domain. First, it demonstrates the performance capabilities of IP-TNE for Web workload generation. Second, it highlights the sensitivity of Web server performance to WAN conditions, including delays and packet losses.

The experiments in this paper demonstrate the feasibility of high-performance WAN emulation using parallel discrete-event simulation techniques on shared-memory multiprocessors. Our experiments with the Apache Web server achieve 3400 HTTP transactions per second for simple Web workloads, and 1000 HTTP transactions per second for realistic Web workloads, 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, link speeds, packet losses, packet sizes, and bandwidth asymmetry, all have significant impacts on Web server performance. WAN emulation enables stress testing and benchmarking of Web server performance in ways that may not be possible in simple LAN test scenarios.

Keywords: Network emulation, Web performance, TCP/IP, WAN emulation

1 Introduction

There are three classical approaches to the performance evaluation of Internet protocols and applications: experimental, simulation, and analytical approaches. These approaches have been the foundation of computer systems performance evaluation research methodology for many years.

The experimental approach arguably offers the most realistic environment for producing credible performance results. Examples of this approach include "hands on" operational measurements of existing systems and applications, as well as "proof of concept" implementations of new protocols and applications. This approach often requires significant investment in experimental infrastructure, either to create a dedicated test environment, or to provide adequate instrumentation for measurement of operational systems and networks. Significant effort is also required to implement, configure, test, and evaluate the systems under study.

Simulation is another widely-used methodology for performance evaluation research. Simulation modeling involves abstracting the logical components of the system being modeled, and defining the interactions, and the timing of these interactions, between components. Discrete-event simulation is often used for this purpose. The simulation approach enables the evaluation of systems that may or may not physically exist, and provides a completely controlled environment for the performance experiments. However, the abstractions used in the simulation model may sometimes compromise the fidelity of the results produced.

The analytical modeling approach uses mathematical methods to analyze a problem. Examples of this approach include queueing theory for the analysis of queueing network models, Petri nets for the modeling of systems, and optimization techniques from operations research for resource allocation problems. While valuable

for many problem domains, analytical approaches often require greater degrees of abstraction to produce tractable mathematical analyses. In some cases, these abstractions and assumptions reduce the practical applicability of the analytical results.

Network emulation is, in some sense, a hybrid performance evaluation methodology. Network emulation combines aspects of experimental implementation with aspects of simulation (or even analytical) modeling. This approach has received increasing research attention in recent years [13, 15, 23], as researchers address large, complex, and challenging Internet performance problems [3, 4, 6, 7, 8, 9].

The network emulation approach is attractive for network performance studies, for several reasons. First, an emulation environment offers a flexible, controllable, and reproducible environment for performance experiments. Second, network emulation enables controlled experimentation with end-user applications (e.g., Internet gaming, video streaming, Web servers) in their native form, without requiring source code modifications to port them into a simulator, and without facing the transient behaviours of the Internet. Third, emulation enables experimentation with a wide range of network and workload configurations. As indicated by Nahum *et al.* [23], the properties of a Wide-Area-Network (WAN) environment can have significant impact on Internet protocol behaviours and Web server performance. Only by testing end-user applications in a wide range of WAN scenarios can confidence be gained in the robustness of a server or application for Internet deployment.

In this paper, we explore the use of network emulation in the evaluation of Internet systems and applications. In particular, we consider Web server benchmarking using the Internet Protocol Traffic and Network Emulator (IP-TNE) [28]. The IP-TNE is built using a parallel discrete-event simulation (PDES) kernel, enabling high-performance WAN emulation.

The IP-TNE enables the testing of real networks and distributed systems under controlled conditions, as provided by network simulation. 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. In other words, the IP-TNE provides the means to "translate" packets between real and simulated Internet environments. This translation is accomplished through a technique similar to "IP masquerading", carefully implemented and tuned to provide high-performance packet reading and writing at Gigabit Ethernet rates.

The purpose of this paper is to demonstrate the capability of parallel network emulation using the IP-TNE. We use Web server benchmarking as an initial case study, for several reasons. First, it demonstrates the capabilities of the IP-TNE as a Web server workload generator. Second, it serves to validate prior results [23] highlighting the impacts of round-trip delays and WAN packet losses on Web server performance. Finally, our study demonstrates that a "centralized" approach to WAN emulation, if properly implemented, is adequate for Web server benchmarking.

Figure 1 provides an illustration of our approach to WAN emulation for the purposes of Web server benchmarking. Rather than following the traditional "centralized" approach in Figure 1(a), we use the approach in Figure 1(b), wherein the clients themselves are within the IP-TNE. This approach has several advantages. First, it eliminates the need for extra equipment in the experimental setup. Second, there is no need for elaborate synchronization of multiple client machines in the experiments. Third, it provides complete control over the client workload: we can model either homogeneous or heterogeneous clients, and we can completely specify their HTTP and TCP behaviours [17, 19, 20]. Finally, this approach provides a fuller demonstration of the performance capabilities of the IP-TNE. Nahum *et al.* [23] argue that the approach in Figure 1(a) is not scalable for Web server benchmarking; we demonstrate via the IP-TNE example that the (more aggressive) approach in Figure 1(b) *is* feasible (and scalable enough) for Web server benchmarking.

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.

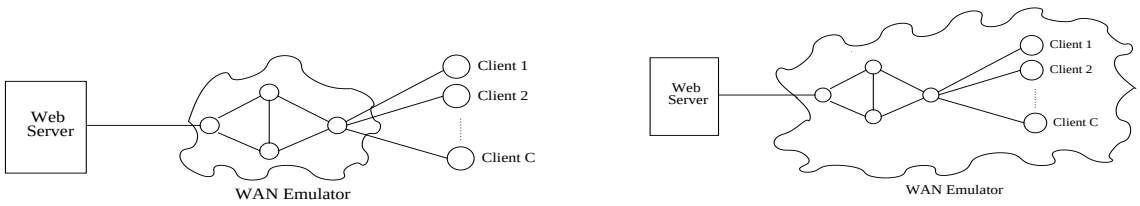


Figure 1: Possible Approaches to WAN Emulation: (a) Traditional Approach; (b) Our Approach Using IP-TNE

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 [15], DummyNet [27], and ns-2 [13] 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, since route computation can add significant overhead [25].

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 [25, 26], to the best of our knowledge these projects are aimed primarily at improving scalability of network models that can be simulated [25], and the inter-operability of multiple simultaneous simulations (i.e., distributed simulations [26]). Our focus is on the execution-time performance of simulations of modest-sized networks, which are still of practical interest to the users of an emulator.

We believe that the IP-TNE offers significant advantages for WAN emulation of any Internet application. In this paper, we demonstrate these advantages through a "case study" of Web server benchmarking.

2.2 Web Server Benchmarking

There are many tools available for Web server benchmarking: ApacheBench [1], `httperf` [22], `s-client` [7], SPECWeb [29], SURGE [8], WebBench [31], and WebStone [32], 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 experiments in a carefully controlled environment. However, the LAN environment is also somewhat biased, since it typically 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, 9, 10]. This observation highlights the need for WAN testing of Web servers for robustness and performance. Nahum *et al.* [23] applied WAN emulation in their WASP (Wide-Area Server Performance) test environment, by using a DummyNet [27] "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 [23]).

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, we support 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.

3 IP-TNE: Internet Protocol Traffic and Network Emulator

The Internet Protocol Traffic and Network Emulator (IP-TNE) [28] is a computer network emulator that uses a fast parallel discrete-event simulation (PDES) kernel called TasKit [34]. This section provides an overview of the IP-TNE, and a description of several of its key architectural components.

IP-TNE is based on an IP network simulator called the Internet Protocol Traffic and Network (IP-TN) simulator, developed in prior work [30]. IP-TN uses the TasKit kernel and is designed to model 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 and write packets from a real network.

3.1 Architectural Overview

Figure 2 shows the main components of the IP-TNE. The IP-TNE is based on the IP-TN network simulator (on the left in Figure 2). This network simulator 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 (on the left) 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.

The exact configuration of the TasKit simulator component depends on whether it is executing on a single processor, or on multiple processors of a shared-memory parallel computer. When executing sequentially, the TasKit simulator runs as a single thread. When executing in parallel, a thread is allocated for each available processor.

3.2 The TasKit PDES Kernel

TasKit is a simulation kernel that implements the Critical Channel Traversing (CCT) algorithm [34]. CCT is an extension of the Chandy-Misra-Bryant (CMB) algorithm that incorporates scheduling decisions into the conservative causality maintenance calculations. It uses a dynamic load balancing scheme that has been shown to work well when used with computer network simulations with irregular workloads.

TasKit uses the logical process modeling view of discrete-event simulation. With this, the system being modeled is represented by logical processes (LPs) which communicate using timestamped event messages. TasKit employs a channel-based event messaging view, so any pair of LPs that may ever exchange event messages must have channels defined between them (*a priori*).

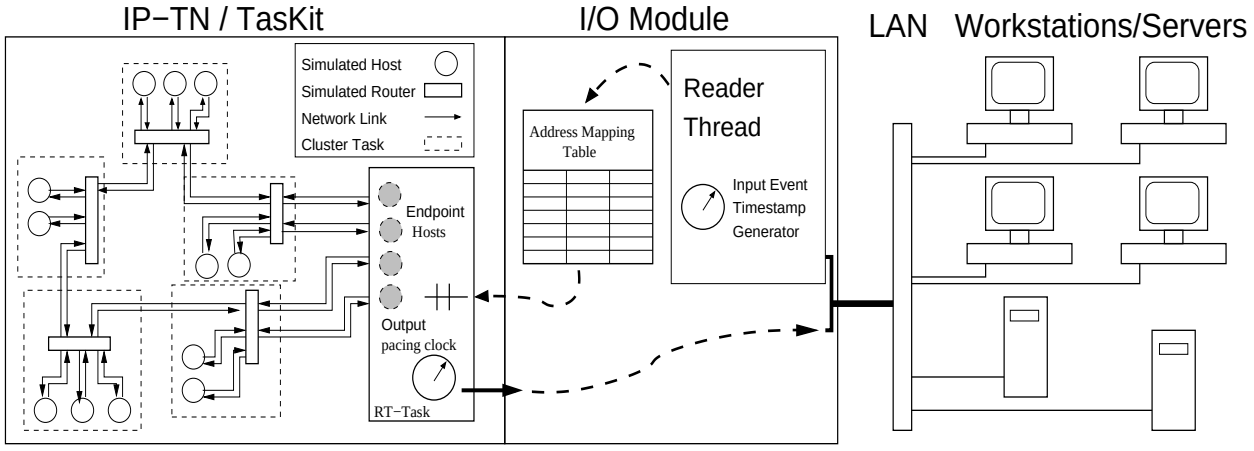


Figure 2: Architectural Overview of the Internet Protocol Traffic and Network Emulator (IP-TNE)

TasKit uses a partitioning scheme where LPs are grouped into collections referred to as tasks. A task is formed using a graph construct where vertices represent LPs in the model and edges represent channels between LPs. This task-based approach is different from most systems where load information has to be used when determining the best way to partition a model. The topological partitioning scheme used by TasKit, along with the scheduling scheme used to take advantage of this partitioning, makes TasKit less sensitive to changes in the values of input parameters than other systems that we have explored [34].

3.3 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 [24] library could be used for packet reading, though we currently use a simple packet reader that just performs the tasks required by IP-TNE [12].

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. To support the use of the fictitious MAC address the reader has to be able to read and respond to Address Resolution Protocol (ARP) requests for the IP address of a gateway to a virtual network.

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. There is no need to compute checksums within the simulator since simulated data corruption can simply be indicated by a flag in the event representing the packet, but checksums are needed on the real network. For packets that have been routed between real clients, the IP checksum needs to be recomputed, since the Time-To-Live (TTL) field of the IP packet changes when it leaves the emulator. 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 [12]. 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.

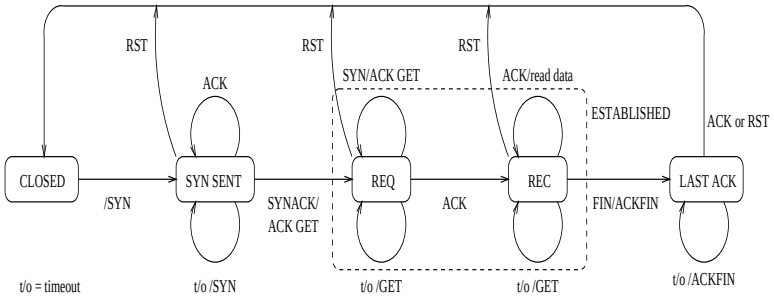


Figure 3: State Diagram of Simple TCP Model Used to Make HTTP Requests

3.4 IP Model

The IP-TNE has detailed modeling of the IPv4 protocol. Each host and network interface is assigned its own virtual IP address. 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. Note that the checksum does not need to be recomputed at each router since no data corruption can occur within the simulator, unless the links are specifically configured to model this. 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` [16].

3.5 TCP Model

The TCP clients in the IP-TNE are simplified models of the basic TCP state machine. Figure 3 shows an outline of the state machine used for the client TCP models. The actual implementation has extra transitions to deal with packets received out of order.

The clients model TCP's three-way handshake for connection setup (SYN and SYN/ACK handshake) and close (FIN and FIN/ACK). The model includes sequence numbers, acknowledgments (ACKs), and retransmit timers for the detection and retransmission of lost packets. The client models support full bi-directional exchange of data and ACKs, with an advertised receive window of 64 KB. In this paper, however, the client TCP models generate only outbound data packets carrying HTTP/1.0 "GET" requests. The bulk of the data transfer is from the HTTP server to the client. The TCP clients generate one ACK for each incoming TCP segment. The HTTP server initiates the close of the TCP connection once all data has been transferred and acknowledged. After the TCP connection has been closed and the final ACK has been received, the client enters a think time period (set to 0 seconds for the tests presented in this paper) before initiating the next TCP connection to the server. Each host uses a new (sequentially assigned) port number for each new TCP connection. Concurrent TCP connections from each client are supported in the IP-TNE, but are not used in the experiments in this paper. Rather, concurrency is achieved by increasing the number of clients. Persistent connections [20] are not currently modeled in the IP-TNE, but will be added soon.

These TCP clients generate a closed-loop HTTP request workload. Upon the successful completion of one HTTP transaction, clients initiate the next transaction. This design differs from the open-loop control of `httperf` [22], for example. Note also that our TCP model waits for the final FIN/ACK from the server before counting a connection as closed, unlike ApacheBench.

The TCP clients use a "one-shot" approach for TCP connection initiation, unlike real TCP implementations. The timeout for a lost SYN packet is 5 seconds. If no ACK has been received from the server during this period, the client abandons that TCP connection, and initiates the next one using a new SYN request with a new proposed sequence number. Late replies to a previous SYN packet result in a RESET generated by the client to abort that earlier TCP connection setup. This simplified protocol provides greater control over workload generation by avoiding TCP's backoff mechanisms during connection setup.

Clients record the start time and end time for each successful TCP connection, as well as the number of data bytes exchanged in each direction. The statistics are stored in memory [11] until the end of the emulation experiment, to avoid undue interactions with the host operating system (e.g., network traffic, disk I/O). These statistics are used to compute response time and throughput information for the experiments.

4 Experimental Methodology

This section describes the experimental methodology for the network emulation experiments, including the hardware, software, and network configuration for the experiments, the WAN network model, the 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 connected by Gigabit Ethernet on a private LAN. 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. That is, each ES40 has a Gigabit Ethernet card. The two cards are directly connected via a short multi-mode fiber. The direct connection enables testing without any contention from other network streams. The default MTU size for the network interface is 9000 bytes (the "jumbo frame" size supported by Gigabit Ethernet), though the TCP clients in our experiments negotiate a Maximum Segment Size (MSS) smaller than the MTU during TCP connection setup.

In 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.

The Web server in our experiments runs on the other ES40, and makes use of all four available processors. The Web server software[2] is Apache (version 1.3.20), which is widely used on today's Internet [1, 23].

Apache is a process-based HTTP server, which uses child processes to serve incoming HTTP requests. The number of processes is configurable, though processes are often created or destroyed dynamically based on incoming request load. In our experiments, the HTTP server 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. Preliminary experiments showed that the performance of Apache is very sensitive to the settings of some of these parameters. Some settings improve performance, some seem to limit performance, and others seem to cause anomalous behaviour. For example, with 150 child processes and 100 requests per child, we observed

[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.

[2] We have conducted preliminary experiments with the Flash Web server as well, but for time and space reasons, we restrict our discussion in this paper to the Apache experiments.

periodic delays of 300-900 milliseconds at the server, approximately every 5 seconds. We speculate that this delay is due to the (almost simultaneous) termination and creation of 150 processes after serving every multiple of 15,000 identical 1 KB requests (about 5 seconds worth) in round-robin order amongst the child processes. Setting the per-child request limit to be infinite eliminated this problem.

Other relevant Apache parameters were configured as follows. MaxClients was set to allow 64 connections to be served simultaneously. Setting this value larger tended to degrade performance due to the resource loads placed on the host operating system, and excessive virtual memory page fault traffic generated by the competing processes. MinSpareServers, MaxSpareServers and StartServers were also set to 64 to avoid process creation startup overhead at the beginning of each experiment. MaxRequestsPerChild was set to 0 to reflect the "infinity" setting mentioned previously. The experimental `mmap` module was not used in our experiments. The Web server was restarted between each experiment.

Within the operating systems of the ES40s, the size of the TCP listen queues was set to 64 KB, to reduce the likelihood of the Web server dropping incoming connection requests. The TCP send and receive socket buffer sizes were set to 600,000 bytes, and the number of IP queues at the network interface was increased[3] from 1 to 16. These settings made no difference on the emulator itself, since all the TCP state is contained within the TCP client model instances.

Parameters were also updated in the network cards to disable the interrupt thread and to reduce the time between send and receive cycles. This was done to remove some inconsistency from the test results. Finally, the TCP delayed ack timer was disabled on each machine. This modification was required to eliminate a deadlock problem observed with the directly connected network cards. This appeared to be the Nagle/delayed ACK interaction described by Mogul and Minshall [21], though it was occurring even when the NO_NAGLE TCP socket option was used.

4.2 Network Model

In this paper, we use the IP-TNE to model an arbitrary 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, using a network modeling language of our own design [18]. Other network topologies can easily be constructed in a similar fashion.

A representative example of our emulated WAN topology appears in Figure 4. The internetwork consists of N ($1 \leq N \leq 32$) subnetworks, where each subnetwork consists of H ($1 \leq H \leq 4096$) 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 4 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 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 4 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 strictly 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

[3]Statistics recorded at the ES40 running the Web server show that inbound IP packets are being lost at the network interface. This packet loss occurs regardless of the number of IP queues used, because of the server overload induced in our experiments. However, we increased the number of IP queues simply to reduce the number of packets lost at the interface.

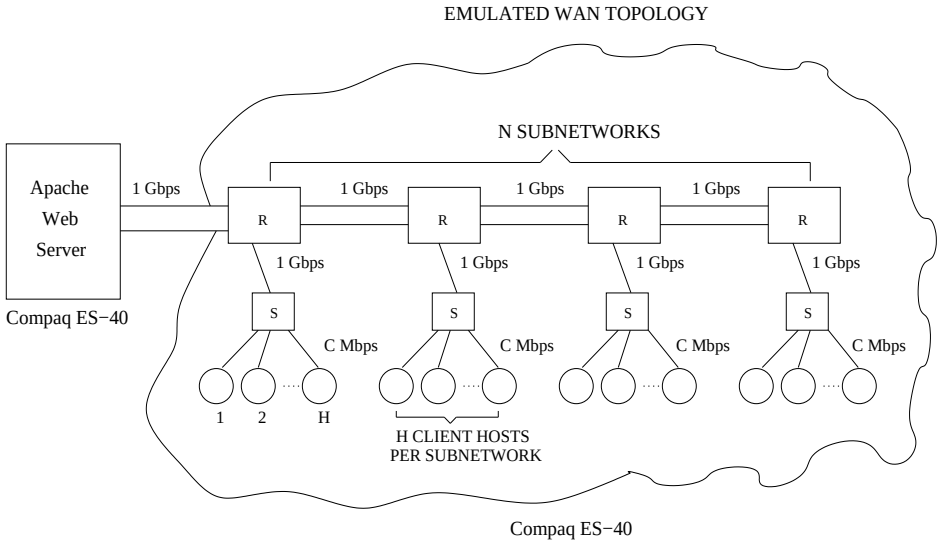


Figure 4: Emulated WAN Topology for Web Server Benchmarking Experiments

Table 1: Experimental Factors and Levels for WAN Emulation Experiments

Factor	Levels
Number of Clients H	1, 2, 4, 8 . . . 4096
Number of Subnetworks N	1, 2, 4, **8**, 16, 32
Client Access Link Capacity C (Mbps)	1, 10, **100**, 1000
Link Propagation Delay D (sec)	**0.001**, 0.002, 0.004, 0.008
Maximum Transmission Unit (MTU) Size (bytes)	296, 576, **1500**
Router Queue Size Q (KB)	8, 16, 32, 64, 128, **512**

from the Web server, and thus different round-trip times. Clients farther from the Web server are at a slight disadvantage for Web data transfer performance, because of the RTT dependencies in the TCP protocol.

4.3 Experimental Design

We consider six main factors in our WAN emulation experiments: number of clients, number of subnetworks, network link speed, propagation delay (i.e., round-trip time), MTU size, and router queue size. Table 1 summarizes the factors and levels used in the experiments. Values shown in bold font are the default values used in our WAN emulation experiments in Section 5.3.

A one-factor-at-a-time experimental design is used. The number of clients and the number of networks are used to change the workload generated by the IP-TNE. The link capacity, RTT, and MTU factors are used to assess the impacts of different WAN configurations 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

Several metrics are used to quantify the performance of the IP-TNE and the Apache Web server. These metrics fall into three main categories: server-centric metrics, network-centric metrics, and user-centric metrics. The server-centric metrics include HTTP transaction completion rate and failure 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.

Performance data is recorded at both the client-end and the server-end during our experiments. At the server, an access log (in Common Log Format) is recorded. Post-processing of this log indicates the elapsed time for the test, the number of Web transactions, the client generating each request, the URL and status (response code) for each request, and the size of each transaction.[4] Within the IP-TNE, each Web client records the start time and end time of each TCP connection, and the size (in bytes) for both the request and the response for the Web document transfer (including HTTP header overhead). Post-processing of this data is used to determine network throughput, transaction rate, failed connections, and mean and median document transfer times.

A 2-minute run-length is used for the emulation experiments, with a warmup period of 5 seconds. Preliminary experiments indicated that this run-length was adequate to provide stable estimates of the performance metrics of interest (see Figures 5(a) and (b)).

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 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[5] during experiments. 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 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 focuses on the performance for simple Web workloads with fixed size documents. Section 5.2 focuses on the performance for a more realistic Web workload with variable-size documents. Section 5.3 focuses on the impacts of WAN conditions on Web server performance. Section 5.4 summarizes our results.

[4]The server access log actually provides redundant information, relative to the information recorded at the clients. After our preliminary validation experiments, logging at the server was disabled to maximize Web server performance.

[5]The `tcpdump` utility was also vital in verifying the correct operation of the TCP client models (e.g., three-way handshake, MSS negotiation, TCP options processing, slow-start) in our experiments. Both `tcpdump` and `traceroute` were used to verify the correct operation of the IP protocol models (e.g., addressing, routing, fragmentation, ICMP).

5.1 Performance Results for Fixed-Size Requests

The first set of experiments considers a simple Web workload model where all document transfers are the same size. This workload model is used to assess the HTTP transaction rate and network throughput for small (1 KB) Web document transfers. In this experiment, the number of clients (H) and the number of subnetworks (N) in the emulated WAN is varied, to understand the relationship between the WAN model and the generated workload.

Figure 5 presents the performance results from the experiments with 1 KB transfers. Figure 5(a) and (b) are time series plots showing the network throughput achieved for different numbers of clients and subnetworks in the emulation experiment. Figure 5(c) shows the average HTTP transaction rate sustained over the duration of the emulation experiment, as a function of the total number of clients in the emulated WAN. Figure 5(d) shows the corresponding average network throughput for each experiment. Figure 5(e) shows the mean response time at the server as a function of client load. Figure 5(f) shows the percentage of HTTP requests that failed (i.e., SYN packet lost, or connection request rejected by server) as a function of offered load. Each line on the graphs presents results for a different number of subnetworks or clients in the emulated WAN topology.

Figure 5(a) shows the network throughput achieved using 1 KB transfers when each of the N subnetworks ($1 \leq N \leq 32$) has a single client. The throughput is sampled and reported for each one second interval of the experiment. The graph shows that the warmup period required to reach steady-state is brief (about 5 seconds), and that throughput is fairly[6] constant. The average network throughput increases steadily as the number of subnetworks is increased from $N = 1$ to $N = 32$, since the total number of clients increases. The results in this graph represent very light load on the Web server.

Figure 5(b) shows how the throughput 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 throughput increases from 1.1 Mbps to 35 Mbps. The throughput grows almost linearly with the number of clients, up to $H = 32$, but grows slowly thereafter as the load begins to saturate the HTTP server. Several anomalous throughput dips are evident when the offered load is generated by $H = 32$ clients or more. These throughput drops appear to be periodic, with a period[7] of 30 seconds.

Figures 5(c) and (d) summarize how the Web workload generated by the IP-TNE changes with the number of clients and networks in the modeled WAN topology. Figure 5(c) shows the transaction completion rate, while Figure 5(d) shows the network throughput. Each data point in Figure 5(d) is computed as the average throughput for the steady-state portion (i.e., ignoring the warmup period) of each experiment, such as those illustrated in Figure 5(a) and (b). The results in Figure 5(c) are computed similarly, from the report of completed HTTP transactions during each one second interval of the experiment. The rest of this paper uses this graphical format to present performance results; additional time series plots are omitted for space reasons.

Figure 5(c) shows that a single client ($H = 1$, $N = 1$) can generate a (rather modest) sustained rate of 112 transactions per second to the Web server. This workload represents a network throughput of approximately 1.14 Mbps, including HTTP protocol overhead (see Figure 5(d)). The elapsed time for each 1 KB transfer is approximately 9 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 2.4 milliseconds each, on our emulated WAN topology). Each 1 KB transfer involves only one TCP data packet sent by the Web server. The server does not piggyback its FIN on the final data segment. The TCP client waits for the final FIN/ACK before closing the connection.

Figure 5(d) shows the corresponding network throughput results for the same experiments. Throughput behaviour is consistent with the transaction rate behaviour, since all Web document transfers are the same size in this experiment.

For a single subnetwork ($N = 1$) in the emulated WAN topology, the transaction rate and the throughput roughly double as the number of clients is doubled (note the logarithmic scale on the horizontal axis in these

[6]The anomalous throughput drop observed at time 45 seconds for $N = 1$ subnetworks is due to a response time of 986 milliseconds for one transaction, while each other transaction required about 9 milliseconds. Analysis with `tcpdump` suggests that the cause is the loss of a client ACK packet at the server's network card, requiring a TCP timeout and retransmission at the server.

[7]These anomalies seem to be caused by disk I/O, likely to flush to disk some buffered file system writes containing inode or superblock information. In our experiments, the `smoothsync_age` parameter in Tru64 was set to 30 seconds.

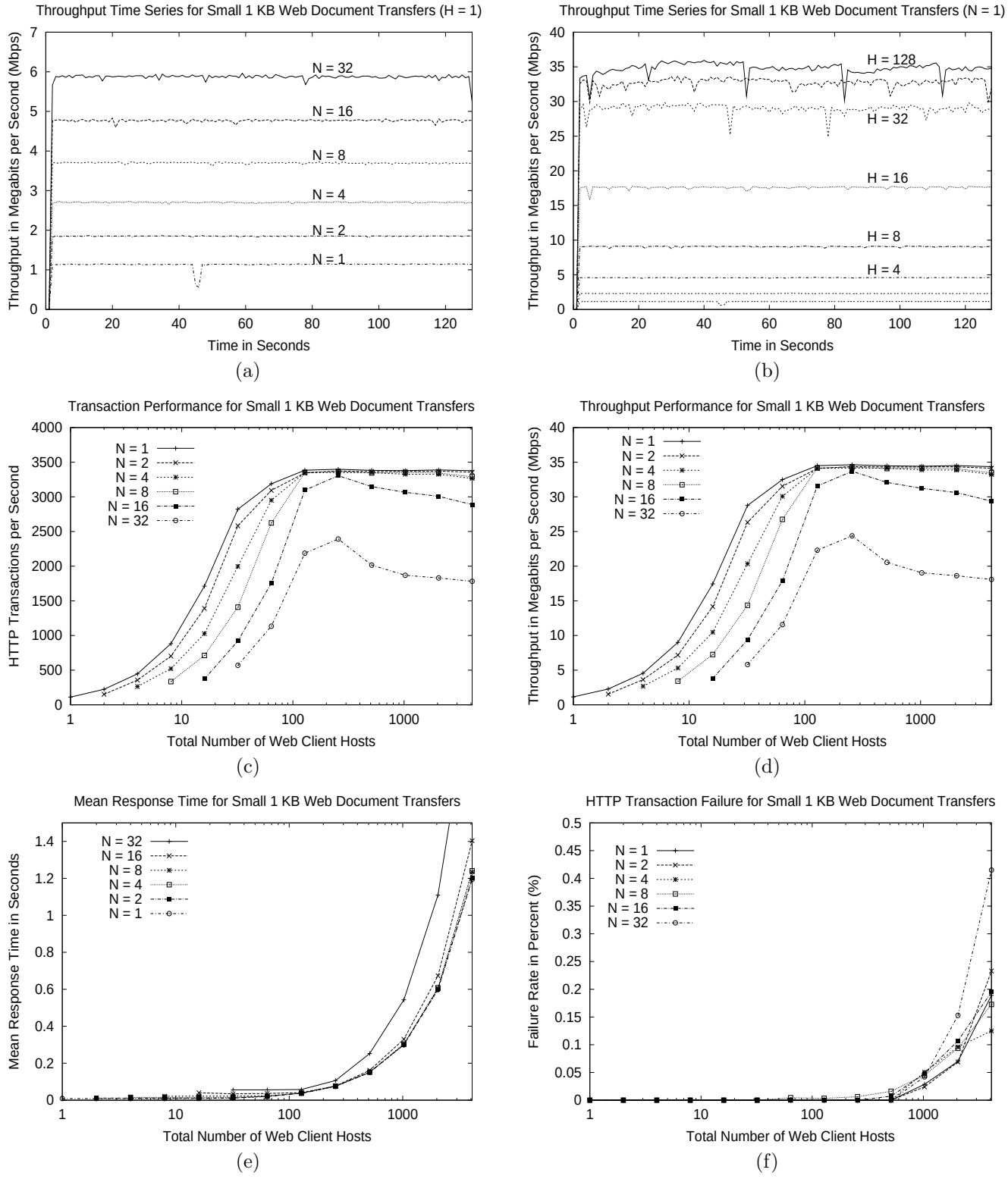


Figure 5: IP-TNE Performance Results for 1 KB Web Document Transfers: (a) Throughput vs. Time (H = 1); (b) Throughput vs. Time (N = 1); (c) Transaction Rate; (d) Network Throughput; (e) Response Time; (f) TCP Connection Failure Rate

figures), up to about $H = 32$ clients. Beyond this point, the transaction rate and the throughput are rather flat, reflecting a saturation of the Web server at a rate of approximately 3400 transactions per second. For larger numbers of emulated subnetworks, a similar trend occurs, with a performance plateau of approximately 3400 transactions per second (for small N) once there are 128 or more total clients.

One additional observation from Figure 5(d) is that, for a given total number of clients, the Web workload generated from multiple ($N > 1$) emulated subnetworks (each with fewer clients) is lower than that generated by the single subnetwork case ($N = 1$). For example, 16 clients on 1 network produce a throughput of 17.45 Mbps, while 16 networks each with 1 client produce a throughput of 3.81 Mbps.

The reason for this "dropoff" in server and network throughput is round-trip time effects. The linear topology of the emulated WAN (see Figure 4) 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.

The dropoff in throughput is most pronounced for $N = 32$ subnetworks, where the clients farthest from the Web server have an RTT of 64 milliseconds. Because of the closed-loop nature of our workload generator, these far-away clients contribute little to the workload, in terms of transactions and throughput. At best, they can complete a 1 KB transfer every 192 milliseconds (3 RTTs), ignoring transmission and queueing delays, for a load of about 5 transactions per second, while occupying precious TCP state resources at the server throughout each transaction. Furthermore, in the $N = 32$ subnetwork topology, 50% of the clients are *at least* 16 routing hops away from the server, with RTTs exceeding 32 milliseconds. This phenomenon contributes to the dropoff in server and network throughput.

Figure 5(e) presents the mean response time results for the experiments with 1 KB transfers. This graph shows the expected increase in delay as the load on the server is increased. The shape of the curve is similar for each WAN topology considered, especially when the number of subnetworks is small (e.g., $N < 16$). Queueing delay begins to dominate the response time once the total number of clients is 256 or more. The response time curve is steeper for $N = 16$ and $N = 32$ subnetworks, since the round-trip times for these topologies are much larger than for topologies with smaller N.

Finally, Figure 5(f) summarizes the percentage of TCP connection failures during[8] our experiments, for different numbers of clients and subnetworks generating the load. Connection failures are recorded at the client when no SYN/ACK is received within 5 seconds of the client's TCP SYN packet. Few connection failures are observed when the total number of clients is less than 128, though the failure rate increases noticably as the load is increased further. The highest failure rate observed is 0.42% of the requests, for $N = 32$ subnetworks. Many of the connection failures happen in the warmup stages of each experiment, due to synchronized arrivals of initial requests from many clients.

The results in Figure 5(e) and Figure 5(f) indicate that the client workload generated by the IP-TNE is capable of saturating the Apache Web server, with fixed-size 1 KB requests. The next section studies Web server performance under a more realistic Web workload model.

5.2 Performance Results for a Realistic Web Workload

This section studies the performance of the Apache Web server when presented with a "realistic" (synthetic) Web workload generated using the IP-TNE for WAN emulation. The experiments focus on the transaction rate supported by the server, the network throughput achieved, and the mean and median response times for Web clients. These performance characteristics are then studied later in Section 5.3 as different aspects of the emulated WAN topology are changed.

The workload for these Web server benchmarking experiments approximates that observed in real Web server and Web proxy workloads. In our baseline experiments, we model a Web site with a total of 3000 Web pages, 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. This static Web content is created once, and used repeatedly in all remaining experiments.

[8]For these results only, we include connection failures that occur during the warmup period of the experiments.

The synthetic workload is generated using the ProWGen proxy workload generation tool obtained from the University of Saskatchewan [14], 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. Approximately 25% of the documents are so-called "one-timer" documents [2]. Temporal locality in the aggregate workload is modeled using an LRU stack reference model, with a maximum stack depth of 1000 documents. There is zero correlation between document size and document popularity. Each document is independent. Only static Web content is modeled.

The Web content is allocated in the server file system using a hierarchical directory structure. Each file is assigned a unique integer document identifier, in order of document popularity (from most popular to least popular). A unique URL is then constructed that includes the document identifier. A three-level directory hierarchy is built, with 10 items (files or subdirectories) in each directory. Files are assigned to directories based on each successive digit of the file identifier (least significant to most significant). This assignment of documents to directories avoids creating file system "hot spots" with many popular documents in the same directory. However, preliminary experiments indicate that changing the assignment of files to directories has relatively little impact on the experimental results.

Each client in the emulated WAN generates requests based on information from the aggregate workload file. Clients are mapped to requests using a modulo arithmetic technique. Each client cycles through its assigned requests continuously during an emulation experiment, with zero think time between requests. The clients generate requests as fast as possible, in a sequential fashion, using HTTP/1.0.

Figure 6 presents the performance results from the experiments with the realistic Web workload. Figure 6(a) shows HTTP transaction rates. Figure 6(b) shows network throughputs. Figure 6(c) shows the mean response times for Web document transfers, while Figure 6(d) shows TCP connection failure rates.

Figure 6(a) has the same general shape[9] as Figure 5(c) for 1 KB transfers, in terms of HTTP transaction completions per second. For small numbers of subnetworks, the HTTP transaction rate increases proportionally with the number of clients in the network, until reaching a plateau where the Web server is presumably saturated. The primary difference from Figure 5(c) is that the plateau occurs near 1000 HTTP transactions per second, rather than 3400 transactions per second for 1 KB transfers. The explanation for this lower plateau 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.

Three other observations are evident from Figure 6(a). First, the transaction performance tends to decrease when a fixed number of clients is distributed across a larger number of subnetworks. For example, one subnetwork with 8 clients achieves 531 transactions per second, while 8 subnetworks each with 1 client produces 239 transactions per second. 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 $N = 32$ subnetwork topology is not nearly as pronounced here as it was for the experiment with 1 KB transfers. In other words, randomized transfer sizes seem to produce more robust server performance. This result makes sense intuitively since far-away clients tend to have larger transfer sizes (on average) than in the 1 KB scenario, and synchronization effects among client requests are significantly reduced. Third, for each number of subnetworks considered in Figure 6(a), the peak transaction rate is achieved at a slightly different number of total clients. For example, the $N = 8$ topology peaks at 1003 transactions per second when there are 64 clients, while the $N = 16$ topology peaks at 972 transactions per second with 256 clients, and the $N = 1$ topology peaks at 1012 transactions per second with $H = 4096$ clients. This phenomenon (as noted by Nahum *et al.* [23]) is due in part to the changes in client round-trip times as the WAN topologies are changed; as the round-trip delay increases, additional clients are needed to sustain full load on the Web server. We attribute the peak behaviour evident in some of the plots to a tradeoff between increasing the number of concurrent client TCP connections (which generally improves performance) and the competition of these connections for scarce server or network resources (which generally degrades performance). These subtle behaviours were not evident with fixed-size transfers.

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,

[9]Note however that the vertical scales are different in Figure 6 than in Figure 5.

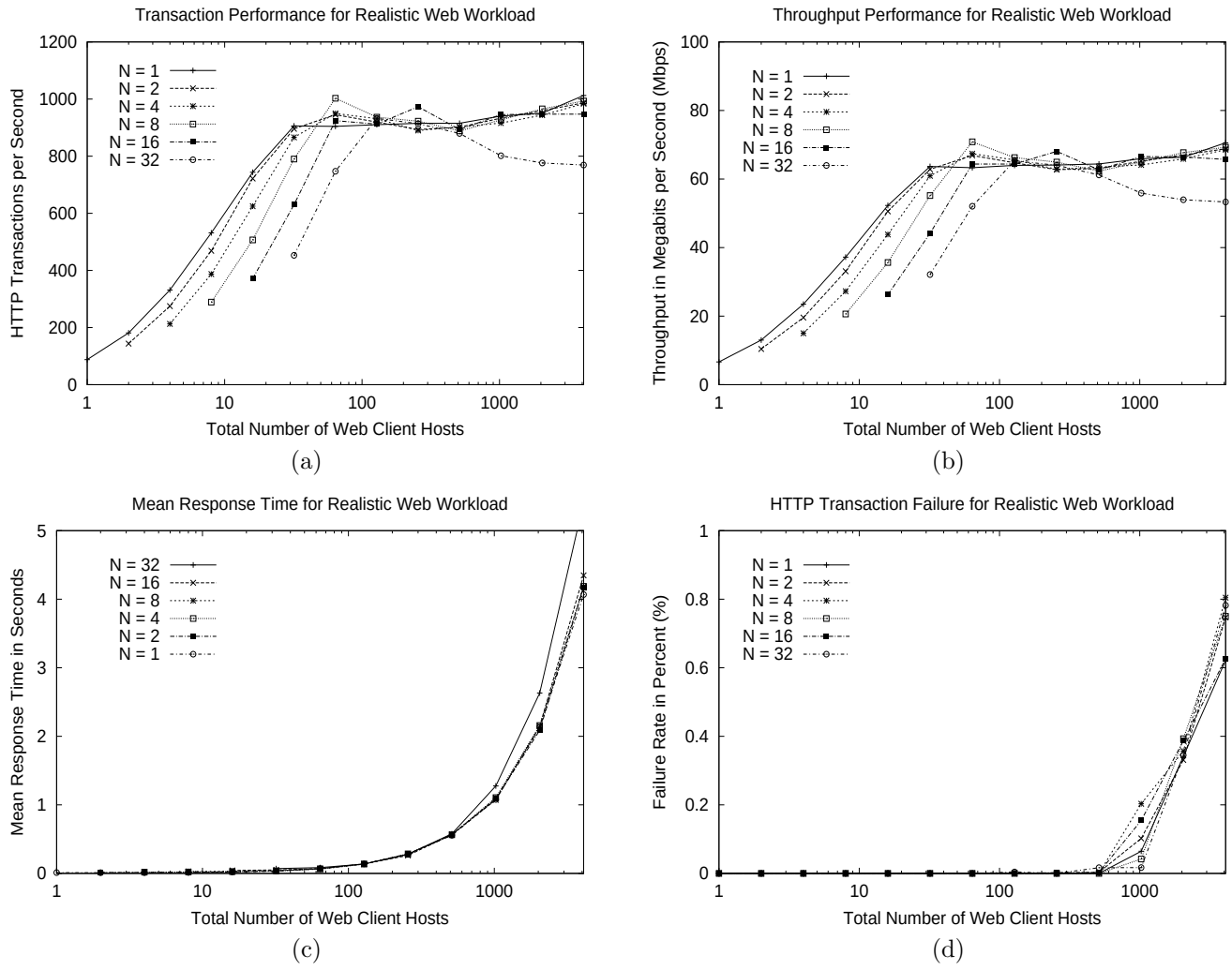


Figure 6: IP-TNE Performance Results for Variable-Size Web Document Transfers: (a) Transaction Rate; (b) Network Throughput; (c) Mean Response Time; (d) TCP Connection Failure Rate

and reaches a plateau of approximately 65 Mbps. This throughput is almost 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 long-lived transfers.

Figure 6(c) shows the mean response times for the experiments with the realistic workload. Saturation of the server begins for workloads with 128 or more clients. Mean response times are higher than median response times (not shown here, for space reasons) because of the heavy-tailed file size distribution in the realistic workload.

Figure 6(d) shows the connection failure rates observed during the experiment. Failure rates become noticable once there are 512 or more clients in the experiment. The highest connection failure rate observed is approximately 0.8%. Connection failure rates are slightly higher here than for the 1 KB experiments, since TCP connection state is held longer (on average) at the server. However, the additional randomization in the workload (due to variable document sizes) reduces the synchronization of connection failures that was evident for 1 KB transfers.

The foregoing experiment provides the baseline scenario for the remaining experiments, which assess the impacts of WAN conditions on Web server performance. Response time plots and connection failure plots are excluded from the remaining experiments to conserve space in the submitted version of the paper.

5.3 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.

5.3.1 Effect of Round-Trip Time

The first WAN experiment focuses on the impact of round-trip time (RTT) on Web server performance. We choose a single WAN topology with $N = 8$ subnetworks as a representative example, and vary the propagation delay between subnetworks to study the impact on performance. Four values of link propagation delay are considered: 1, 2, 4, and 8 milliseconds. For the modeled WAN topology with $N = 8$ subnetworks, this results in client RTT values ranging from 2 milliseconds to 128 milliseconds. This range covers typical RTT values observed in the Internet.

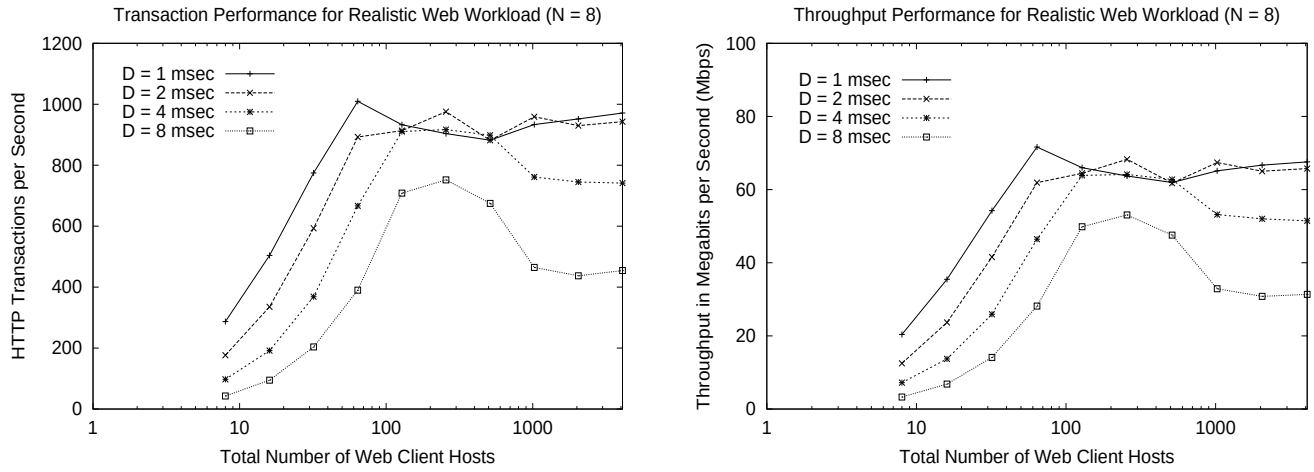
Figure 7 presents the results from this experiment. As the link propagation delay is increased, the curves for transaction rate and network throughput tend to move downward, with the peak of the curve becoming more pronounced, and often moving slightly to the right. Again, the results show that the peak 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. For example, the results for $D = 1$ millisecond link delays peak at 1010 transactions per second for 64 clients, while the results for link delays of $D = 8$ milliseconds peak at 751 transactions per second with 256 clients. These results are consistent with those reported by Nahum *et al.* [23], and are not discussed further here.

5.3.2 Effect of Network Link Capacity

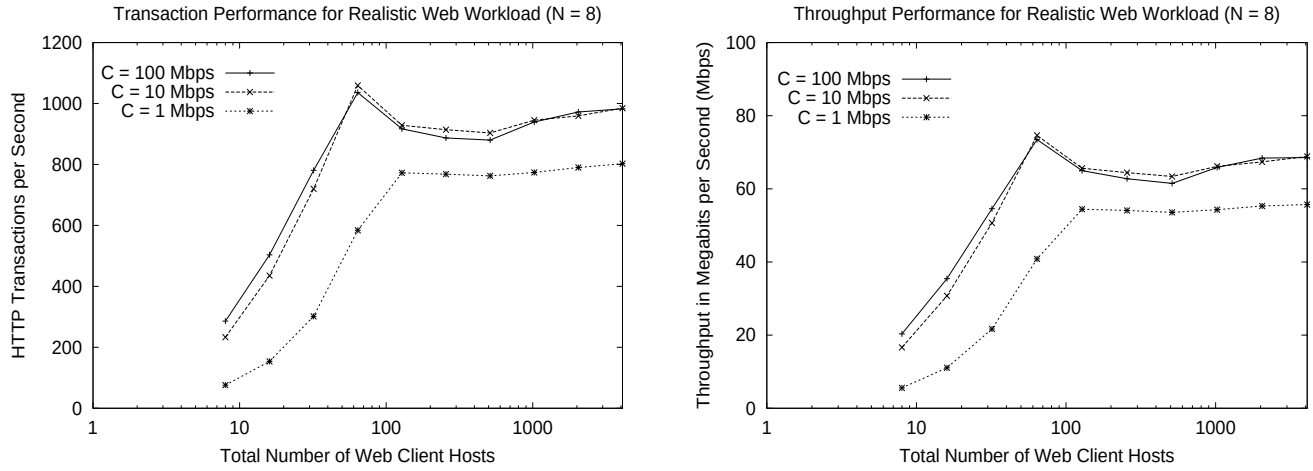
The next experiment focuses on the impact of link capacity on Web server performance. On the simple $N = 8$ subnetwork topology, the capacity of the client access links is varied, from 100 Mbps in the baseline scenario down to 10 Mbps and 1 Mbps. The backbone link capacity in the emulated WAN remains 1 Gbps.

Figure 8 presents the results from this experiment. As the link speed is decreased, the overall load processed by the Web server tends to decrease. This decrease is minimal for the change from 100 Mbps to 10 Mbps links, suggesting that the (dedicated) 10 Mbps per client is adequate to handle typical document downloads. When the link speed is reduced to 1 Mbps, a noticable drop in the HTTP transaction rate is evident, as is a drop in throughput. The 1 Mbps link becomes a bottleneck for many of the clients, leading to longer transfer times for the clients, and longer periods of TCP state consumption at the server. The mean and median client response times (not shown here) increase when the client link capacity is decreased, since the greater transmission time and queueing delay contribute to higher document transfer times.



(a) (b)

Figure 7: Effect of WAN Propagation Delay (Realistic Workload): (a) Transaction Rate; (b) Network Throughput



(a) (b)

Figure 8: Effect of Link Capacity (Realistic Workload): (a) Transaction Rate; (b) Network Throughput

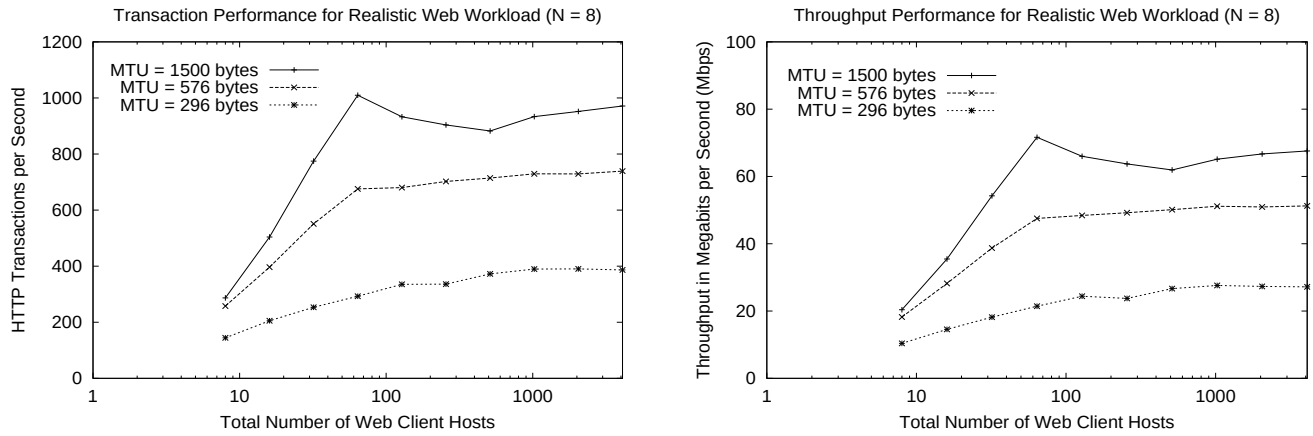


Figure 9: Effect of MTU Size (Realistic Workload): (a) Transaction Rate; (b) Network Throughput

5.3.3 Effect of MTU Size

The third experiment focuses on the impact of network MTU (Maximum Transmission Unit) size on Web server performance. On the simple $N = 8$ subnetwork topology, the MTU size is varied on the baseline scenario with 100 Mbps access links and 1 millisecond propagation delays. Three MTU size values are considered: 1500 bytes, 576 bytes, and 296 bytes. These values are common in today's LAN and WAN environments.

Figure 9 shows the results from this experiment. As the MTU size is decreased, the overall server and network throughput decrease. In fact, small MTU sizes can reduce Web server performance by a factor of two or more.

The mean and median client response times (not shown here) also increase when the MTU size is decreased, since the number of RTTs incurred by the TCP protocol for document transfers increases. That is, since the TCP congestion control algorithm advances the flow control window in units of segments, the smaller segment size (negotiated between the client and the server during TCP connection setup) results in slower window growth (in terms of bytes) and more RTTs. As a result, transfers take longer. In addition, the smaller MTU size means that the number of network packets increases, and the relative header overhead per network packet increases. These effects all contribute to lower transaction rates, lower network throughput, and higher response times.

5.3.4 Effect of Network Asymmetry

The next experiment focuses on the impact of bandwidth asymmetry [5] at the client network access point, for the simple $N = 8$ 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 1 Mbps down to 64 Kbps.

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"). In our scenario with more frequent per-packet ACKs, the upstream ACK channel will become the bottleneck for Web document transfers.

Figure 10 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

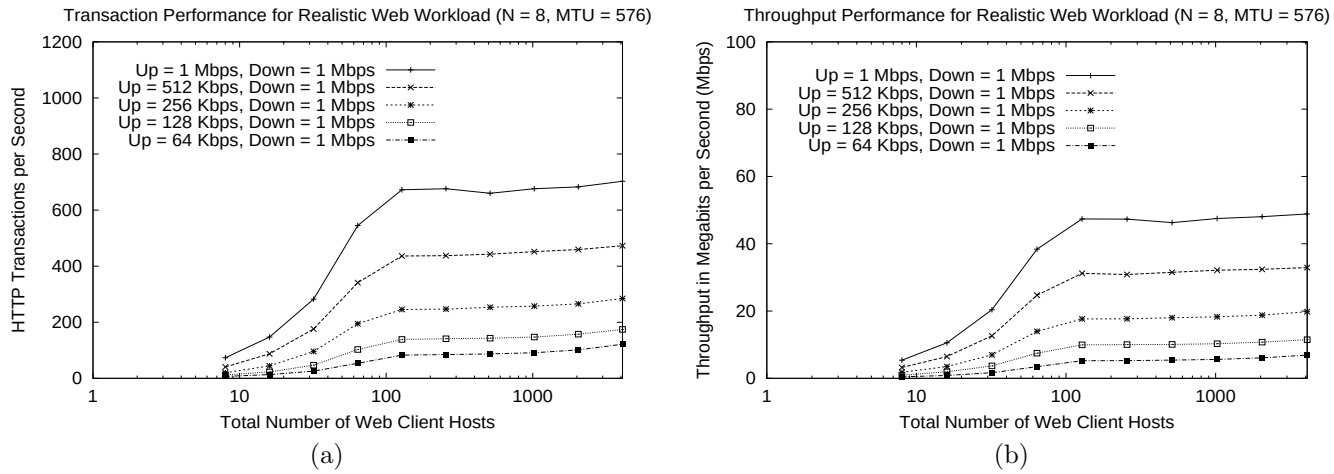


Figure 10: Effect of Network Asymmetry (Realistic Workload): (a) Transaction Rate; (b) Network Throughput

Table 2: Packet Loss Results for IP-TNE WAN Emulation Experiments (Realistic Workload, $N = 8$)

Total Number of Client Hosts	Router Queue Size			
	8 KB	32 KB	128 KB	512 KB
8	4.43%	0.0095%	0%	0%
32	5.90%	0.13%	0%	0%
128	7.84%	0.68%	0.037%	0.00067%
512	5.57%	0.69%	0.083%	0%
1024	2.34%	0.32%	0.086%	0%
2048	1.98%	0.32%	0.091%	0.0069%
4096	2.88%	0.38%	0.10%	0.0016%

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.

5.3.5 Effect of Packet Losses

The final experiment focuses on the impact of packet loss on Web server performance, for the WAN topology with $N = 8$ 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 [33].

Table 2 summarizes the packet loss results observed for the different router queue sizes considered in our experiments (8 KB to 512 KB). Packet loss rates range from 0-8%.

Figure 11 presents the performance results from the WAN packet loss experiment. As the router queue size is decreased from 512 KB to 8 KB, 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 results in lower server and network throughput.

Packet losses have a dramatic impact on TCP performance, and thus on the workload seen by the Web server.

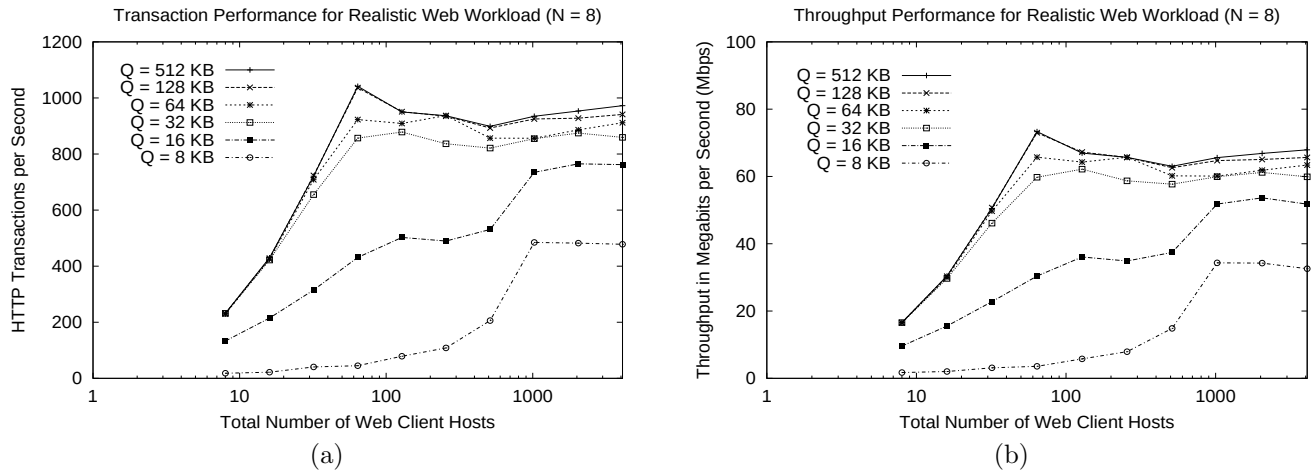


Figure 11: Effect of WAN Packet Loss (Realistic Workload): (a) Transaction Rate; (b) Network Throughput

These observations are consistent with those of Nahum *et al.* [23], 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 [23].

5.4 Summary of Results

This section presented our WAN emulation experiments for Web server benchmarking, with both simple and realistic Web workloads. The WAN emulation experiments demonstrate the importance of WAN characteristics, such as round-trip times, link speeds, MTU sizes, network asymmetry, Internet congestion, and packet losses on Web server performance. Several of our results coincide with those reported by Nahum *et al.* [23], even though our results were produced with a completely different experimental approach. We thus can 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 link speeds, MTU sizes, and network bandwidth asymmetry. All of these factors can have significant impacts on Web server performance in a wide-area network.

6 Conclusions

This paper has presented the design, implementation, and use of a parallel discrete-event IP network emulator, the Internet Protocol Traffic and Network Emulator (IP-TNE), for the purpose of Web server benchmarking. The experiments focus on two main issues: the performance of the IP-TNE itself, and the performance of a Web server (Apache) when subjected to workloads from clients in an emulated WAN environment.

This work demonstrates that a "centralized" approach to WAN emulation is feasible, at least in the context of Web server performance testing. Our 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 approach to WAN emulation 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 [23]. We demonstrate the impacts of link speeds, propagation delays, MTU sizes, bandwidth asymmetry, and packet losses on Web server performance. We quantify these results using server-centric, network-centric, and user-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. WAN emulation experiments with the Flash Web server are underway, as are experiments with background traffic on the emulated WAN. Experiments studying the effects of persistent HTTP connections are planned for the near future.

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. Its use for robustness testing (e.g., Denial-Of-Service attacks) is also being considered.

References

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

[2] M. Arlitt and C. Williamson, "Web Server Workload Characterization: The Search for Invariants", *Proceedings of ACM SIGMETRICS Conference*, Philadelphia, PA, pp. 126-137, May 1996.

[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, "Measuring Web Performance in the Wide Area", *ACM Performance Evaluation Review*, Vol. 27, No. 2, pp. 35-46, September 1999.

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

[11] 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.

[12] Reference withheld for blind review purposes.

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

[14] M. Busari and C. Williamson, "On the Sensitivity of Web Proxy Cache Performance to Workload Characteristics", *Proceedings of IEEE INFOCOM Conference*, Anchorage, AL, pp. 1225-1234, April 2001.

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

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

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

[18] Reference withheld for blind review purposes.

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

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

[21] J. Mogul and G. Minshall, "Rethinking the TCP Nagle Algorithm", *ACM Computer Communication Review*, Vol. 31, No. 1, January 2001.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] Reference withheld for blind review purposes.

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

[30] Reference withheld for blind review purposes.

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

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

[33] 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.

[34] Reference withheld for blind review purposes.