Multimedia Tools and Applications, 14, 269–296, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

ENDE: An End-to-end Network Delay Emulator Tool for Multimedia Protocol Development

IKJUN YEOM ikjun@ee.tamu.edu
A.L. NARASIMHA REDDY reddy@ee.tamu.edu
Department of Electrical Engineering, Texas A&M University, College Station, TX 77843

Abstract. Multimedia applications and protocols are constantly being developed to run over the Internet. A new protocol or application after being developed has to be tested on the real Internet or simulated on a testbed for debugging and performance evaluation. In this paper, we present a novel tool, ENDE, that can emulate end-to-end delays between two hosts without requiring access to the second host. The tool enables the user to test new multimedia protocols realistically on a single machine. In a delay-observing mode, ENDE can generate accurate traces of one-way delays between two hosts on the network. In a delay-impacting mode, ENDE can be used to simulate the functioning of a protocol or an application as if it were running on the network. We will show that ENDE allows accurate estimation of one-way transit times and hence can be used even when the forward and reverse paths are asymmetric between the two hosts. Experimental results are also presented to show that ENDE is fairly accurate in the delay-impacting mode.

Keywords: delay emulation, network simulation, one-way transit times, multimedia protocol/application development, packet behavior, internet

1. Introduction

Various kinds of multimedia applications and protocols are being developed to run on the Internet. A new protocol or application after being developed has to be tested on the real Internet or simulated on a testbed for debugging and performance evaluation. For example, a video delivery protocol developer has to understand how the protocol reacts to various levels of congestion, delay and loss behavior in the networks so that the protocol can be tuned to work well in most cases. Since the Internet is a large and complex network, it is not practical to conduct sufficient experiments directly on the real network. The user would need access to multiple hosts across the Internet. That is why we need tools to mimic the Internet behavior. However, it is not easy to simulate the packet behavior in the Internet because of its heterogeneity and rapid change.

The difficulties of the Internet traffic characterization and simulation have been addressed by Paxson and Floyd [6]. The paper shows that the Internet has three key properties: technical and administrative heterogeneity, rapid growth over time, and immense changes over time. With these properties, it is difficult to obtain a general agreement for characterizing the Internet traffic behavior. Thus a basic problem of many Internet simulations is how to generate such traffic sources into simulations.

Network simulation is required for various reasons. Some of the popular needs include (a) for protocol/application development of point-to-point delivery (b) for protocol/application development of multicast delivery, and (c) scalability analysis of scheduling

P1: vendor, GHS
Multimedia Tools and Applications KL1280-03 April 4, 2001 19:19

algorithms/protocols. The work reported here addresses the needs of point-to-point delivery (a) and is being enhanced for multi-point delivery (b). Protocol development and testing can be done in various ways. Two of the popular techniques include the use of traffic traces and the use of real network (with an access to a remote machine). Our experience in using these techniques for developing effective video delivery protocols over Internet motivated the design of the End-to-end Network Delay Emulator (ENDE) tool reported here. We experienced the following difficulties in designing and testing the protocol: (a) traces of traffic behavior from the past are not very useful since the Internet traffic is changing rapidly over time, (b) needed access to a remote machine (c) the traffic behavior on the network was constantly changing and we could not conduct repeatable experiments (d) the results obtained were valid only for the single path between our machine and the remote machine we had access to.

ENDE is developed to address these issues:

- To obtain realistic traces of current Internet traffic.
- To enable protocol testing on a single machine.
- To enable repeatable experiments.

ENDE makes the following significant contributions. (1) We report on a method for obtaining accurate traces of one-way delays between two nodes on a network without having access to a remote node. This enables a user to generate traces whenever they are needed. Since remote access is not needed, ENDE can be used to generate as many traces as needed to obtain reasonable confidence. Generated traces enable repeatable experiments. We call this a “delay-observing” activity. (2) Trace based simulations have inherent problems since the background traffic is unaffected by changes in the protocol/application being modified. Due to this reason, it is necessary to simulate the protocol on a real network. ENDE allows this by modeling the “delay-impacting” activity of the application/protocol being developed. ENDE supports a highly accurate “delay-observing” mode and a fairly reasonable “delay-impacting” mode to make it a useful tool.

A simple model for ENDE is shown in figure 1. ENDE simulates end-to-end packet behavior on an Internet path between a local host and a destination host. As shown in the model, ENDE consists of two single server queues with finite buffers. One is used as a forward path, and the other is used as a reverse path. Packets from a client or a server are queued in each buffer and transferred to a destination (the client or the server) after experiencing certain delays.

Each delay consists of two components, a fixed component that includes the transmission delay at a node and the propagation delay on the link, and a variable component that includes the queuing delays at each node. The variable component is determined by other Internet traffic. ENDE estimates the other Internet traffic through Internet Control Message Protocol (ICMP) probes. By using ICMP packets for collecting data representing the current state of the Internet, ENDE can emulate current Internet packet behavior without a login access to the destination host. Figure 2(a) shows a typical client-server model that is generally used for testing new applications or protocols. Using ENDE, we can test and measure the performance of new protocols with any remote host in the Internet without running any application on the remote host as shown in figure 2(b).

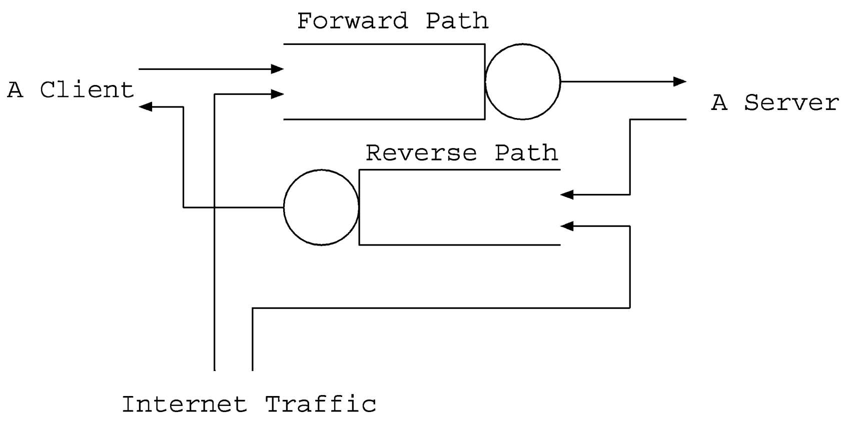


Figure 1. A simple model for ENDE.

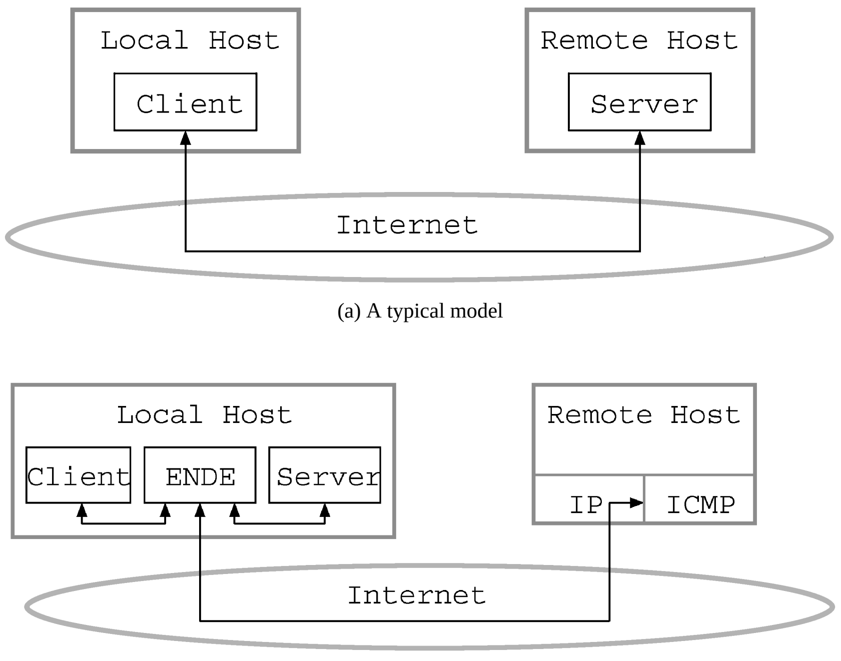


Figure 2. Client-server models in the Internet.

The rest of this paper is organized as follows. Section 2 discusses how to characterize end-to-end Internet packet behavior and introduces new tools for measuring and analyzing them. Section 3 presents a methodology to implement ENDE. Section 4 presents simulation results and compares them with experimental results. Finally, this paper is concluded in Section 5.

2. End-to-end delay emulation

In this section, we propose a new network tool for measuring the end-to-end Internet packet behavior and discuss how to estimate effective bandwidth and how to synchronize two network clocks for estimating One-way Transit Times (OTTs).

Several studies have been proposed to characterize the Internet dynamics. The end-to-end round trip delays of small UDP packets sent every 39.06 ms from a source to a destination node are examined in [8]. The measurements indicate that the IP level service provided in the Internet yields high losses, duplicates and reordering of packets, in addition that the round trip time varies over time significantly. Bolot measured round-trip-delay of UDP echo packet stream sent at fixed intervals to several Internet paths [9]. He characterized end-to-end Internet packet delay and loss, and estimated bottleneck bandwidths on the paths from the measurements with packet pair technique [10]. TReno (Traceroute Reno) was designed to measure the Internet performance using bulk transfer metric [11]. Carter at Boston University proposed new tools for server selection system [12]. The tools use ICMP packets for probing network bandwidth. Recently, pathchar was released by Jacobson [13]. It was developed from traceroute for analyzing performance (including transmission delay, loss rates and queuing delay) at individual links along a path. All of these tools use round trip times to characterize the network traffic. However, when the network paths are asymmetric, the round trip times are insufficient to present an accurate picture of the differences in the forward and the reverse paths.

Some of the well-known freely available network simulators are REAL [1], x-Sim [2] and Network Simulator (ns) [3]. REAL (REalistic And Large) was initially developed from the NEST (Network Simulation Testbed) for comparing the fair queuing gateway algorithm with first-come-first-served scheduling, and is used for various purposes. X-Sim is a network simulator based on the x-kernel, an object-based network protocol implementation framework [4]. Ns is a simulation tool developed by the Network Research Group at the Lawrence Berkeley National Laboratory. It is an event-driven simulation engine and intended to explore the behavior inherent to the underlying congestion control algorithms [3].

Paxson reports on a large-scale experiment to study end-to-end Internet packet dynamics based on measurements of TCP bulk transfers conducted between 35 sites [14]. The measurements are based on One-way Transit Times (OTTs) instead of RTTs in order to allow asymmetries along forward and reverse paths. Clock synchronization between local and remote hosts is necessary to measure OTTs accurately. Paxson also proposed algorithms for measurements of packet transit times [15]. His algorithms are for detecting and deleting clock adjustment and relative skew in addition to clock synchronization.

ENDE uses ICMP packets to obtain information about other Internet traffic and loss rates. Since ICMP packets are generated and sent by kernel automatically, we do not need

P1: vendor, GHS
Multimedia Tools and Applications KL1280-03 April 4, 2001 19:19

to run any other application program on the destination host. The following issues need to be addressed in emulating delay behavior using ICMP probes:

- How to obtain accurate OTTs-this requires taking care of issues such as clock synchronization, clock offset, clock shifts and limited resolution of ICMP timers.
- Do ICMP probes accurately model the delay behavior of UDP and TCP packets? Since ICMP packets are used for transferring control messages, there may be some differences between ICMP and other packet behavior.

In this section, we address these two issues. First, we describe techniques used for getting accurate estimates of OTTs. We observed previously unreported packet behavior with ICMP probes and this required special processing to obtain accurate OTTs. Then, we compare the ICMP based results with UDP and TCP based results to validate the obtained results.

Measurement tools, called PBAT (Packet Behavior Analyzing Tool), are implemented for three protocols, TCP, UDP and ICMP. TCP or UDP version of PBAT consists of a client and a server, and ICMP version has only a client since ICMP packets are echoed by kernel automatically. The client running on a local host consists of two threads, sender and receiver. The user can specify the number of packets to be sent, the size of the packets, and the time interval between successive packets in the command line. The sender generates probe packets that contain a sequence number and a time stamp, and sends them at regular intervals to the destination host. For UDP and TCP, a server on the destination host immediately echoes the received packets to the client with echoed time stamps. For ICMP, echoed timestamp messages are sent by the kernel on the destination host. The receiver receives packets from the destination, and attaches time stamp to the packets. After finishing sending and receiving packets, the client calculates average RTT and its standard deviation, the number of lost packets, and the number of duplications. Then, average forward and reverse delay and their standard deviations are estimated with algorithms described in the following sections.

2.1. *An algorithm for effective bandwidth estimation*

The fundamental idea is that if we send two different size packets (small one is P_s, and large one is P_l) alternatively enough times at a regular interval, then we can assume that a minimum RTT of each size packet contains minimum queuing delay. Since queuing and propagation delay are independent of packet size, the difference between minimum RTTs of different size packets is caused by the transmission delay. Thus, we can estimate the effective bandwidth on a path with the following formula:

$$\text{effective bandwidth} = \frac{\text{length of } P_l - \text{length of } P_s}{\min RTT \text{ of } P_l - \min RTT \text{ of } P_s} \tag{1}$$

Figure 3 shows the relation between minimum RTT and the packet size. We can find that the minimum RTT increases linearly as the packet size increases.

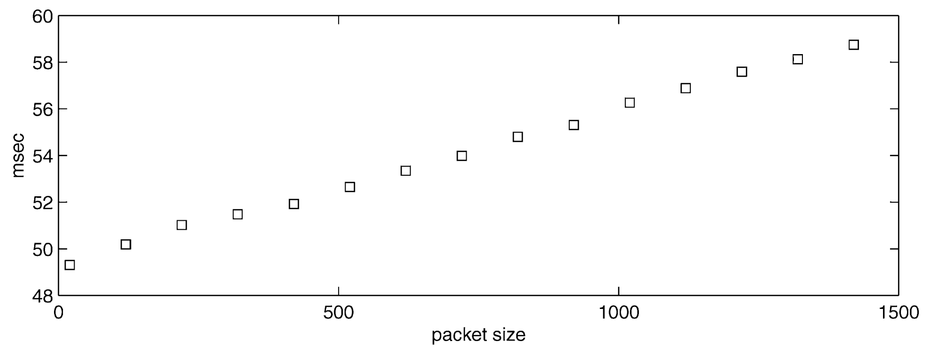


Figure 3. Minimum RTTs of various packet sizes between Texas A&M University and Syracuse University.

2.2. *Network clock synchronization*

In this section, we discuss how to synchronize network clocks and estimate one-way transit time (OTT). Since the clock of each host is not synchronized, we can only estimate OTT, not calculate accurately. We can estimate OTT by dividing RTT in half. However, when the forward and reverse paths are asymmetric, half of RTT cannot be used as OTT [14]. Paxson proposed algorithms for clock synchronization and OTT estimation based on TCP packets [15]. In this paper we apply his algorithms to UDP packets and develop our own algorithms for ICMP packets because some errors are found in ICMP echoed timestamps and so his algorithms cannot be directly applied to ICMP packets.

In the following sections, we use two UDP datasets, U_1 and U_2, and two ICMP datasets, I_1 and I_2 collected using UPBAT and IPBAT. Each of them consists of 5000 UDP or ICMP packet traces. U_1 and I_1 are collected between Texas A&M University and Syracuse university, and U_2 and I_2 are between Texas A&M University and University of California at San Diego. Packets are generated at 50 millisecond intervals with a packet size of 30 bytes. For each trace, a time of packet arrival or departure were recorded at both the sender and the receiver. The clocks used at the remote hosts were not necessarily synchronized.

2.2.1. Removing relative clock offset. In this section, an algorithm, based on [15], for detecting and removing relative clock offset between two hosts is discussed. A relative clock offset means that there is a difference between the two clocks at any moment. Figure 4(a) shows forward and reverse OTTs of U_1 before two network clocks are synchronized.

The forward OTTs are calculated by subtracting sent-time from echoed-time, and the reverse OTTs are calculated by subtracting echoed-time from received-time. To estimate relative clock offset between the two network clocks, we assume that the minimum forward OTT is same as the minimum reverse OTT.

Suppose that a packet, P_1 was sent at time s_1 and arrived at the remote host at time e_1, and that another packet, P_2 was echoed back at time e_2 and arrived at the local host at time r_1.

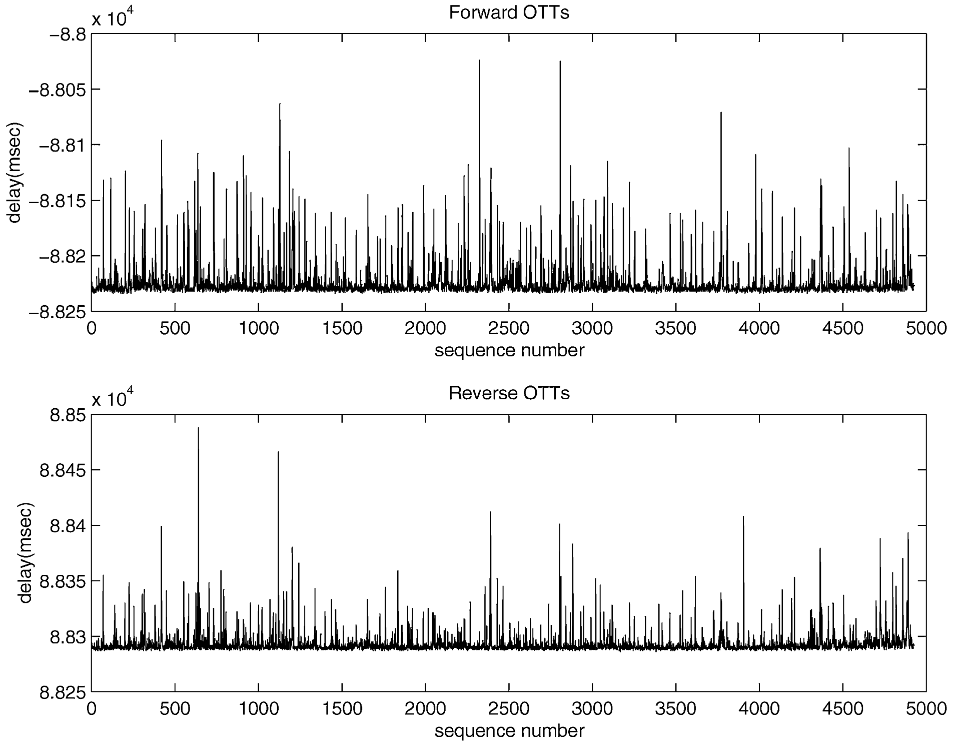
If the clock offset between local and remote clocks is ΔT, and the OTTs of P_1 and P_2 are the minimum forward OTT and the minimum reverse OTT of a dataset, respectively, then from the assumption we have:

$$(e_1 - \Delta T) - s_1 = r_1 - (e_2 - \Delta T) \tag{2}$$

$$\Delta T = \frac{e_1 + e_2 - s_1 - r_1}{2} \tag{3}$$

Using (3) we can calculate ΔT, and then, synchronized times are calculated by subtracting ΔT from each echoed time. Figure 4(b) shows forward and reverse OTTs of U_1 after removing clock offset.

2.2.2. Removing relative clock skew. In the dataset, U_2, we can detect a relative clock skew. A clock's skew at a particular moment is the frequency difference between the two clocks [15]. A clock's skew, even if it is very small, can cause fatal errors on the estimation

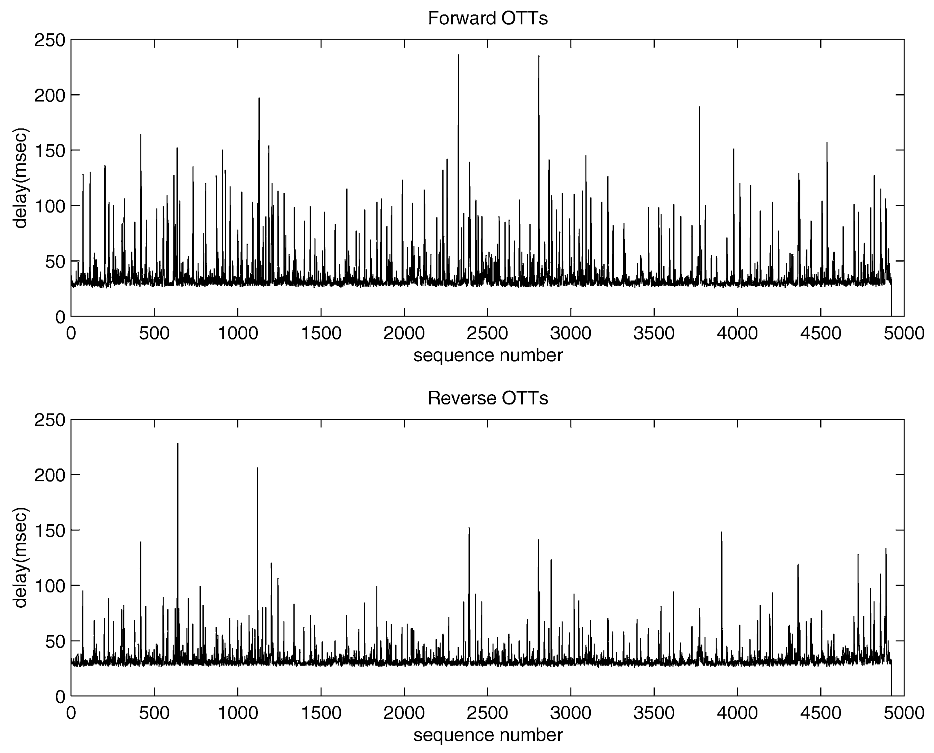


(a) OTTs with clock offset

Figure 4. OTTs of the first UDP dataset.

(*Continued on next page*.)

P1: vendor, GHS
Multimedia Tools and Applications KL1280-03 April 4, 2001 19:19



(b) OTTs after removing clock offset

Figure 4. (*Continued*).

of OTTs in the long run. In this section, we discuss how to detect and remove the relative clock skew. An example is shown in figure 5(a).

As shown in figure 5(a), we can easily find the clock skew graphically, however it is not easy to detect a clock skew algorithmically because, if a delay is different from the previous one, we cannot know whether this difference was caused by a clock skew or by changes in the network traffic. To develop algorithms for detecting and removing a relative clock's skew, we assume that the skew trend is linear. To detect the clock skew, we compare the minimum OTT of first 10 percent packets of a dataset, d_1, with the minimum OTT of last 10 percent packets of the dataset, d_2. By picking the minimum values, we can reduce the noise caused by changes in the network traffic. If the difference is larger than a threshold, T, then we regard that there is a clock skew between the two clocks. The threshold is determined by the time difference, Δd, between the sent times of those two packets. If we want to detect a clock's skew larger than one percent, that means a clock is faster or slower than the other by more than one percent, then we have:

$$T = \Delta d \times 0.01 \tag{4}$$

$$| d_1 - d_2 | \geq T \tag{5}$$

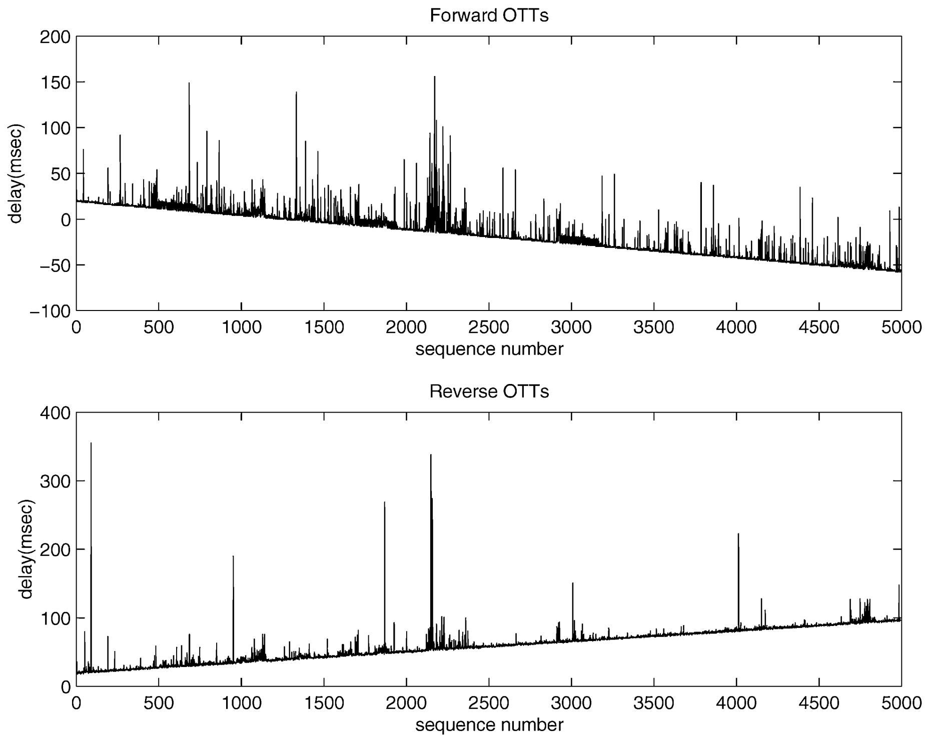
From (5), we can detect a clock's skew. If a skew, ϕ, is detected, we can calculate and remove it by (7). Let s_i and e_i denote the sent and echoed times of the ith packet, respectively. Figure 5(b) shows OTTs after removing the clock's skew.

$$\phi = \frac{d_2 - d_1}{\Delta d} \tag{6}$$

$$e_i = e_i - \phi \times (s_i - s_1) \tag{7}$$

2.2.3. Removing clock shift in ICMP echoed time. In this section, we discuss how to detect and remove a clock shift. A clock shift means that a clock is changed unexpectedly. We found several instances of this in ICMP echoed timestamps from several hosts. An example is shown in figure 6.

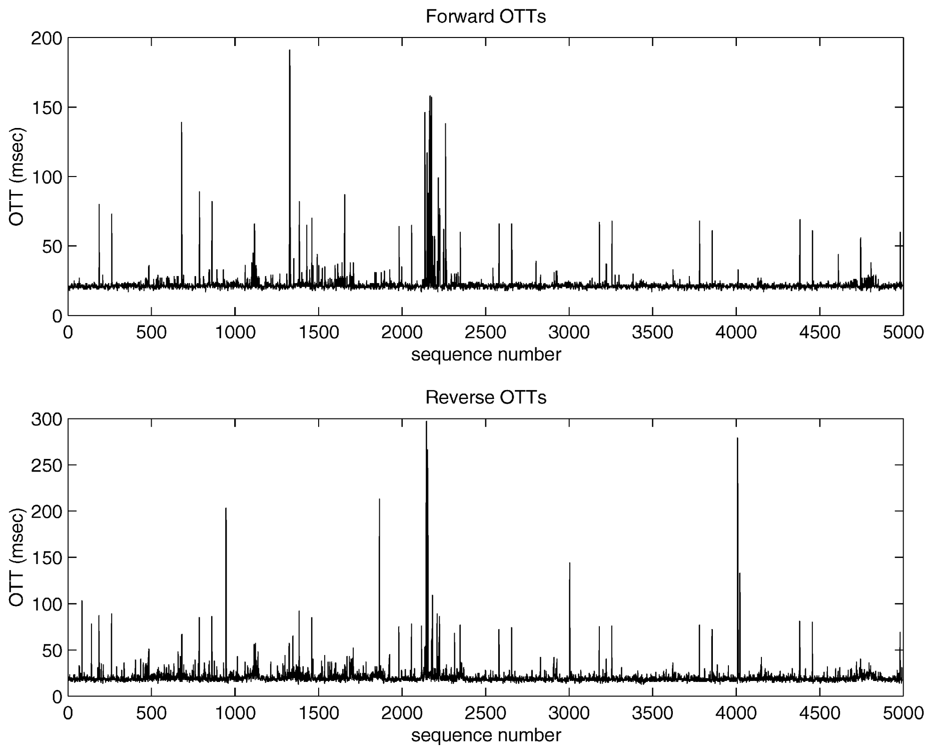
The top graph of figure 6(a) is RTTs of I_4, and the bottom graph is a part of OTTs of I_4. In the bottom graph, OTTs of 3rd, 23rd and 43rd packets are greater than others by about one second. However, the maximum RTT is less than 400 milliseconds from the top graph.



(a) OTTs with clock skew

Figure 5. OTTs of the second UDP dataset.

(*Continued on next page*.)



(b) OTTs after removing clock skew

Figure 5. (*Continued*).

Therefore, any OTT cannot be greater than others by one second, and so we can infer that the echoed times of these three packets are shifted from others. We could not figure out what causes time shifts yet, but we found time shifts in several hosts.

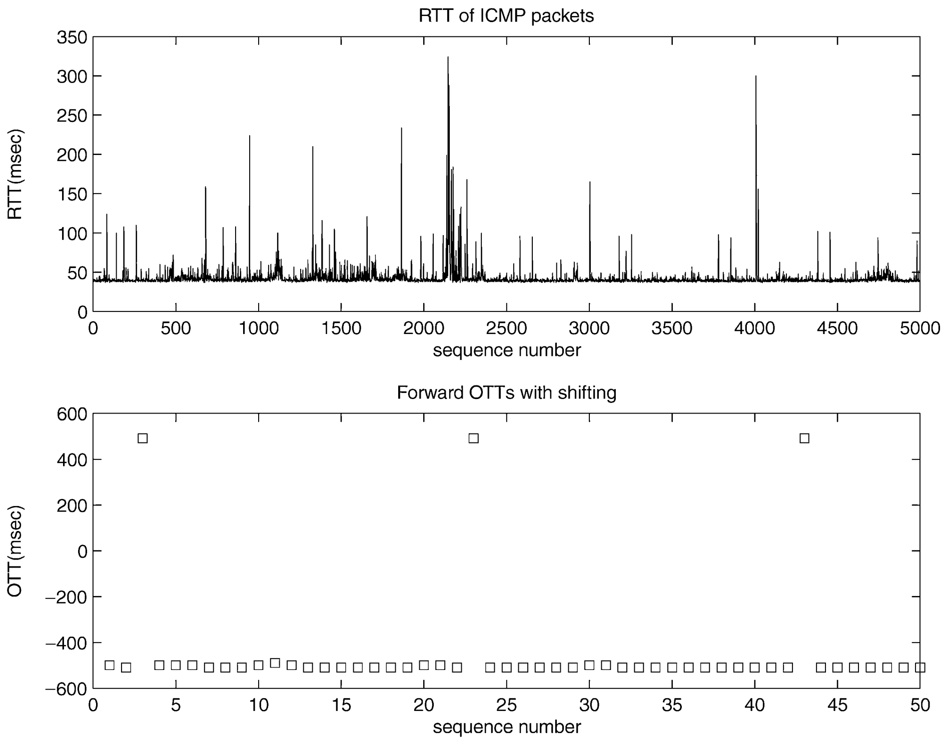
To detect time shifts, we compare the difference between the maximum and the minimum echoed times with the maximum RTT. If the difference is greater than the maximum RTT, that means there are time shifts. Then, we need to estimate the time shift, δ, and remove it. For convenience, we suppose δ is positive. First, we divide the OTTs into original and shifted groups. If the echoed time of a packet is less than the mean of the maximum and the minimum OTTs, we regard that it is not shifted. Otherwise, it is shifted. After dividing the whole dataset, we calculate the difference between the minimum echoed times of each group. If we assume that the minimum times of each group are the same, the difference is δ. By subtracting δ from the echoed time of packets in the shifted group, we can remove the clock shifts. The result is shown in figure 6(b). Even though the time shifts are removed, there are still problems to be solved. We will discuss about them in the following section. In the above algorithm we assume there are single level time shifts. If multiple level time shifts occur, they cannot be removed. Fortunately, we did not find any host where multiple level time shifts occur.

P1: vendor, GHS
Multimedia Tools and Applications KL1280-03 April 4, 2001 19:19

2.2.4. Compensation of clock rounding in ICMP echoed time. The bottom graph of figure 6(b) looks like a stair-case. It is because of the clock's skew and limited time resolution of the ICMP time stamp. We observed that several hosts on the Internet use clocks with limited time resolution of 10 milliseconds for ICMP echo time stamps. A clock's skew can be removed by the algorithms used for UDP packets. However, we need to develop a new algorithm to compensate the times that are rounded off because of the limited clock resolution.

The discarded times can be simply compensated by adding an integer, ε, that is uniformly distributed from zero to nine because the time resolution is ten milliseconds. This would not cause fatal errors in relatively large OTTs. However, it can incur fatal misestimations in small OTTs. To reduce errors in estimating OTTs, we introduce "expected line algorithm." From figure 4(b), 5(a) and (b), we can find that the lower values of OTTs look like a straight line. Using this property of OTTs, the small values of OTTs in figure 6(b) can be estimated as shown in figure 7.

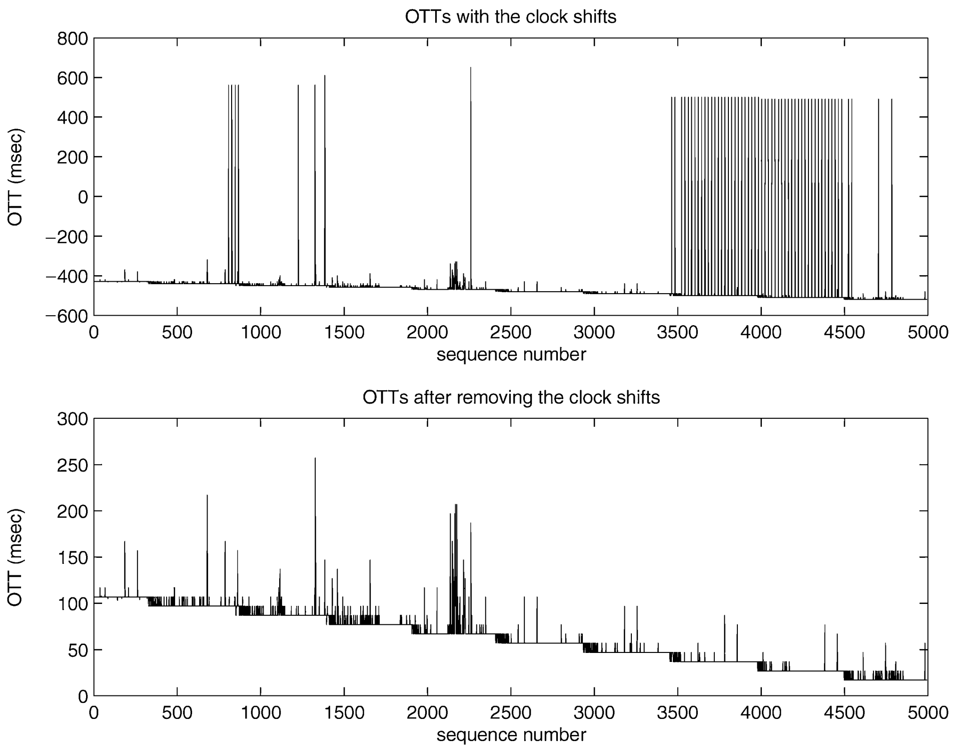
The expected line is the line that connects every point at which OTT starts to round off. To estimate the lower OTTs accurately, noise needs to be removed from OTTs. In



(a) RTTs and forward OTTs

Figure 6. RTTs and OTTs of the second ICMP dataset.

(*Continued on next page*.)



(b) Forward OTTs with and without clock shift

Figure 6. (*Continued*).

OTT measurements, the noise means an additive and positive increase that is induced by other network traffic. We use "de-noising algorithm" proposed in [15] to remove noise. The basic idea of this algorithm is that after dividing a given dataset into subsets, we can find OTTs with very little network-induced noise by looking at the smallest values in the subset. Figure 8 shows the de-noised OTTs. Each subset has ten OTTs.

We can obtain a equation of expected line by connecting two points, A and B, in figure 8.

$$y = \frac{d-b}{c-a}x + \frac{bc-ad}{c-a} \tag{8}$$

To estimate original OTTs, we compare the observed OTTs with expected line. If an observed OTT is less than the expected line, the difference between them and ε are added to the observed OTT. Otherwise, just ε is added. Then, the clock's skew is removed using the algorithms discussed in the above section. The final results are shown in figure 9.

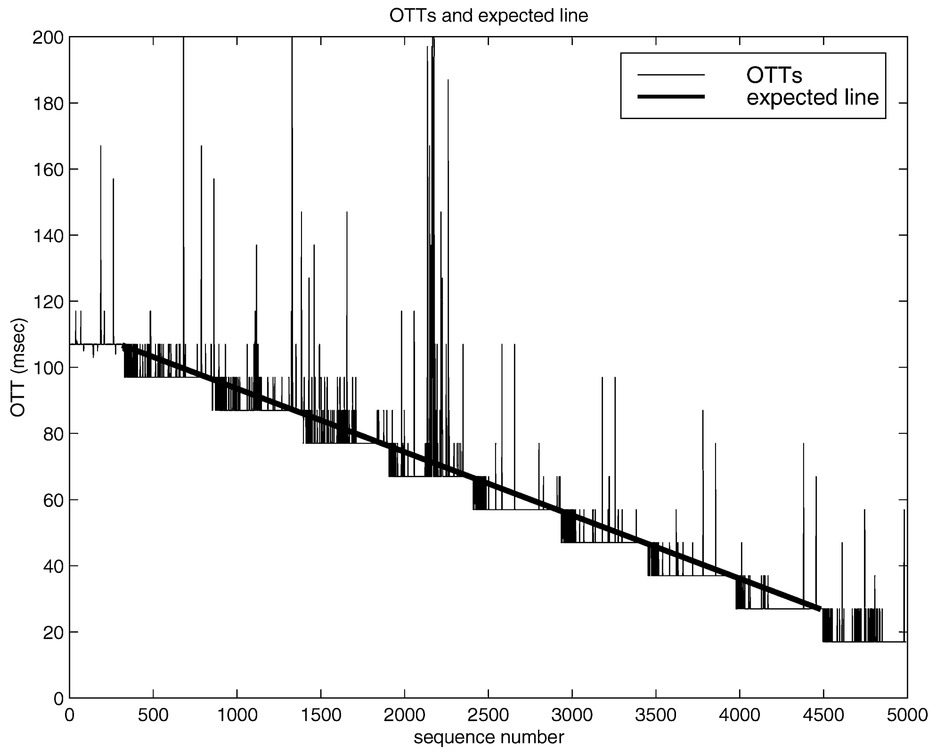


Figure 7. Forward OTTs and expected line.

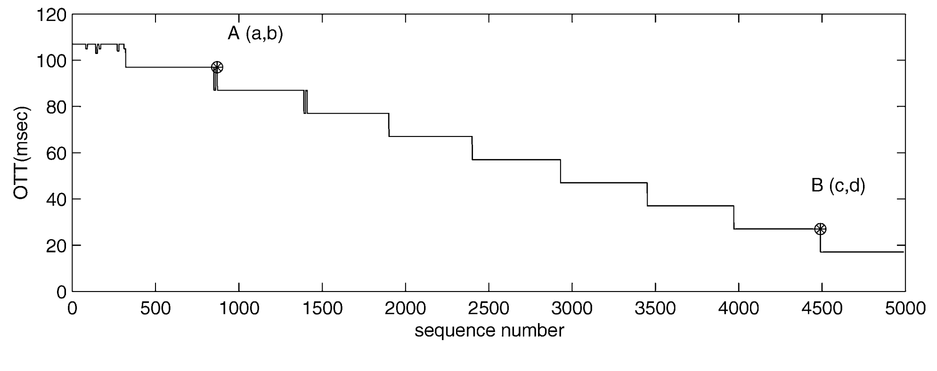


Figure 8. OTTs after removing noise.

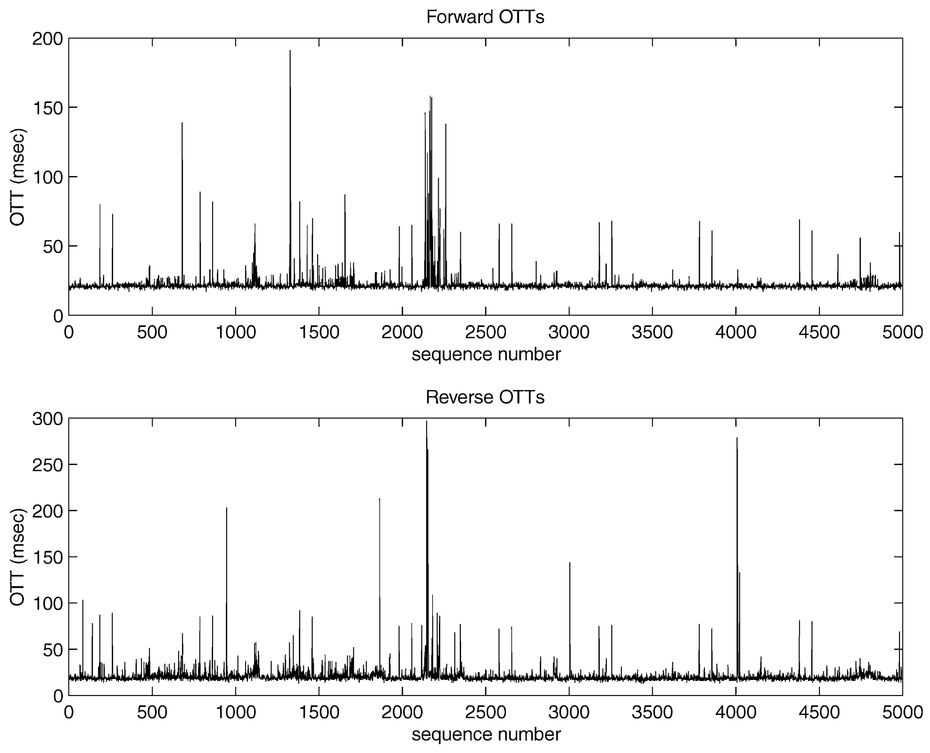


Figure 9. Forward and reverse OTTs of the second ICMP dataset.

2.3. *Experimental results using IPBAT and UPBAT*

In this section, we present the results of the four datasets used in the above section. Table 1 characterizes the results of ICMP based and UDP based measurements.

From Table 1, it is clear that the characteristics of ICMP packets are very close to that of UDP packets. The individual packet OTT of U_1 and I_1 are shown in figure 10. We can find that the forward and reverse paths are asymmetric. From the table and figures, the mean and distribution of OTTs of U_1 and I_1 agree with each other.

In this section, we have explained how to get delay characteristics on a path using UDP and ICMP packets and presented the results. The results show that the delay characteristics of UDP and ICMP packets are pretty close. From the results, it is clear that we can obtain the same characteristics of UDP packets using ICMP packets without access to the remote host.

3. **Implementation of ENDE**

In this section, we present a methodology we used to implement ENDE. ENDE works by intercepting packets sent to a specific port number with TCP or UDP and holding them for

P1: vendor, GHS
Multimedia Tools and Applications KL1280-03 April 4, 2001 19:19

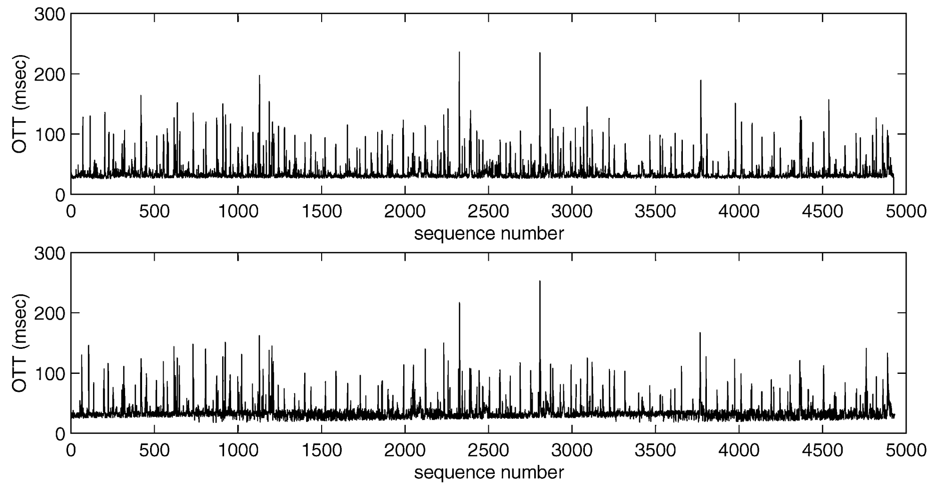
Table 1. UDP and ICMP packet behavior.

Dataset	U_1	I_1	U_2	I_2
Number of loss	76	73	6	6
Minimum RTT (msec)	54	53	37	37
Average FWD delay (msec)	35.8	35.2	22.8	22.9
STD of FWD (msec)	16.2	15.7	10.3	10.1
Average RVS delay (msec)	33.1	33.9	21.9	20.8
STD of RVS (msec)	10.6	11.4	11.8	10.9
Average RTT (msec)	69.0	69.2	44.7	43.8
STD of RTT (msec)	21.2	20.5	16.4	15.6

a certain delay which is determined by the current network state between the local and the destination hosts. It keeps sending ICMP packets, called probe packets, at regular intervals for observing the current network state. Figure 11 shows a block diagram of ENDE. In the following sections, the implementation of ENDE is explained in detail.

3.1. Communication between Kernel and ENDE

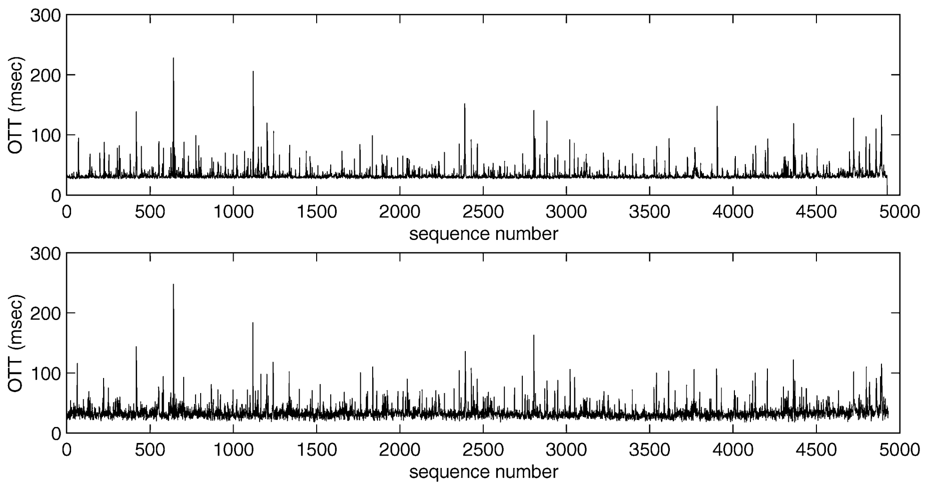
Packet interception is implemented in the kernel in order that any real-world application (e.g. FTP, Telnet) can be emulated by ENDE without modification. Current version of ENDE can manipulate TCP and UDP packets. To intercept a packet being emulated and send to ENDE, two port numbers, N_{server} and N_{ende}, are reserved for a simulated server and ENDE,



(a) Forward OTTs

Figure 10. OTT comparisons between UDP (top) and ICMP (bottom).

(*Continued on next page*.)



(b) Reverse OTTs

Figure 10. (*Continued*).

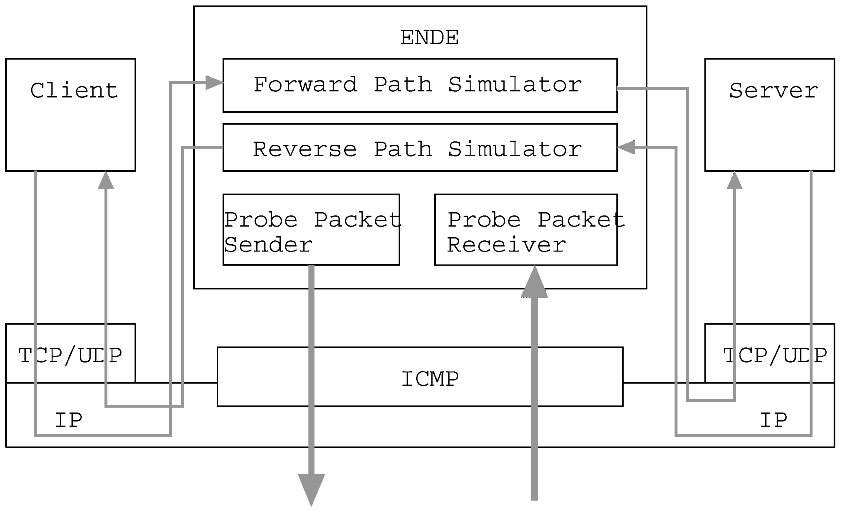


Figure 11. A block diagram of ENDE.

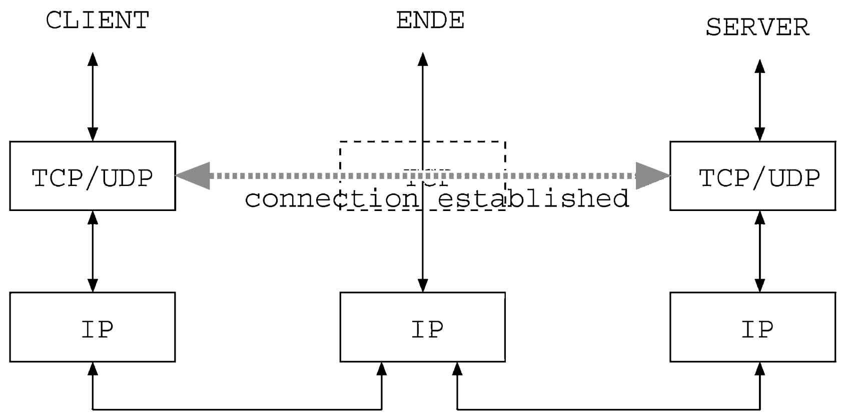


Figure 12. Connection between a client and a server using ENDE.

respectively. Then, a packet whose destination or source port number is equal to N_{server} is regarded as being simulated and is redirected to ENDE by the kernel.

TCP is connection-oriented protocol, and a connection must be established before communication can start. However, ENDE is designed to be used as an Internet path, and so we do not want to open connections between ENDE and the client or the server. Therefore, interception of a TCP packet needs more complicated procedures because system level packets are used for connection establishment, acknowledgments and closing connections. Theses packets are generated and sent by the TCP layer, not by the user level application. Following figure 12 shows the connection between a client and a server using ENDE.

3.2. Preprocess

Before the start of the simulation, ENDE collects data to estimate the effective bandwidth between the local and the destination hosts and to synchronize the two clocks.

To estimate the effective bandwidth, ENDE uses minimum RTTs of two different sizes of packets as described in Section 2. It sends two different sizes, 20 and 1020 bytes, of ICMP packets alternatively 3000 times for obtaining minimum RTT of each packet size. Then, the effective bandwidth can be estimated using (1). After estimating the effective bandwidth, ENDE sends 20 byte ICMP packets 3000 times at 30 millisecond time interval to estimate the relative clock offset, the clock's skew and the time shifts. The procedures of the estimation are explained in Section 2.

3.3. Probe packet sender and receiver

After the preprocessing stage, the probe packet sender keeps sending ICMP probe packets to the destination at 20 milliseconds to observe current network traffic between the local and the destination host. The probe packet receiver receives the probe packets echoed by the destination host and records the arrival times with the sequence numbers. If a packet is lost, RTT of the packet remains as zero. Then the receiver checks the echoed time stamp. If the echoed time is shifted, it is adjusted by subtracting the shifted time estimated in the preprocess stage. If there is clock skew, it can be removed by using (7). Finally, the clock offset is removed from the echoed time. Then, forward and reverse delays are estimated using the following equations.

$$\text{forward delay} = \text{the echoed time} - \text{the sent time} \tag{9}$$
$$\text{reverse delay} = \text{RTT} - \text{forward delay} \tag{10}$$

3.4. *Forward and reverse path simulators*

Forward path simulator simulates a path from the local host to the destination host, and reverse path simulator simulates a path from the destination to the local host. In the implementation of ENDE, forward and reverse path simulators are separated because the two paths are asymmetric occasionally. Each of forward and reverse path simulators consists of a sender, a receiver and a buffer.

The receiver receives TCP or UDP packets, called simulated packets from the client or the server and calculates delay by which the packets need to be deferred. Figure 13 shows the probe packets and the simulated packets.

To calculate the delay, the receiver refers to the record of the probe packet that was sent 20 seconds prior to the arrival of the simulated packet. If the probe packet was returned within 20 seconds, the forward or reverse delay of the probe packet is used as the delay of first 20 bytes of the simulated packet. Otherwise, the probe packet is regarded as being lost, and the simulated packet is also discarded. Then, the delay of the whole simulated packet can be calculated from the delay of first 20 bytes of the packet, d, and the effective bandwidth, μ, estimated in the preprocess stage using (11). P_s and P_p denote the size of

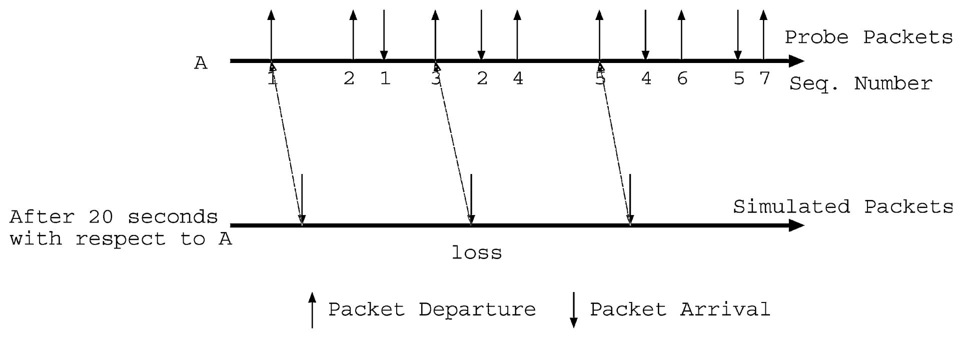


Figure 13. An example of probe packets and simulated packets.

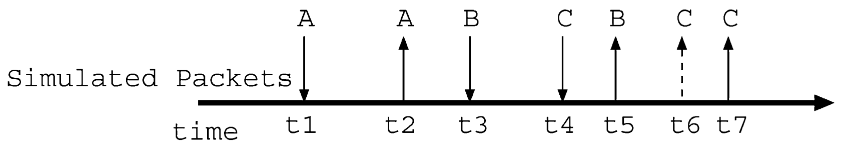


Figure 14. An example of delays of simulated packets.

simulated packet and probe packet, respectively.

$$\text{simulated packet delay} = d + \frac{P_s - P_p}{\mu} \tag{11}$$

From (11), the delay of the simulated packet can be calculated. However, there is a case in which the calculation is not correct. Suppose that the size of the simulated packet is too large to be transmitted before the next simulated packet's arrival, or the next simulated packet arrives so early that the former is not transmitted yet. In either case, the next packet must wait until the former is transmitted completely. Figure 14 shows an example.

In the figure, packet 'A' is transmitted completely before packet 'B' arrives, and so it does not affect the delay of packet 'B.' Packet 'C' arrives before packet 'B' leaves. From (11), packet 'C' is expected to leave at time 't6,' but packet 'B' is not transferred completely and remains in the queue. Thus, the actual leaving time of packet 'C' is time 't7.'

To calculate time 't7,' we define 'effective packet length.' The effective packet length of ith packet, P_{eff}^i, is iteratively defined as sum of the length of ith packet and the remaining length of $(i - 1)$th packet. If we suppose that ith packet arrives at time t_i, then we have:

$$P_{\text{eff}}^i = \begin{cases} P^i & \text{if } t_i - t_{i-1} > \frac{P_{\text{eff}}^{i-1}}{\mu} \\ P^i + P_{\text{eff}}^{i-1} - \mu(t_i - t_{i-1}) & \text{otherwise} \end{cases} \tag{12}$$

Then, the delay can be calculated by substituting P_s in (11) with P_{eff}. Finally, the receiver puts the simulated packet in the buffer with its delay. The sender checks the delay of the packets logged in the buffer with round-robin discipline, and sends the packets that are already deferred by their delay.

3.5. *ENDE in a delay-impacting mode*

All of the techniques described above use ENDE in a delay-observing mode. This is sufficient to produce accurate traces/delay information about existing traffic of the network. However, to test the impact of the protocol/application on the traffic in the network, we also have to simulate the effect of sending data over the network. In other words, ENDE needs to simulate the *delay-impacting* behavior of the protocol/application under study.

Figure 15 shows how ENDE impacts the network traffic. To impact the network traffic, ENDE sends two kinds of probe packets, delay-observing probes and delay-impacting probes. Delay-observing probe is a 20 byte ICMP packet sent every 20 milliseconds to

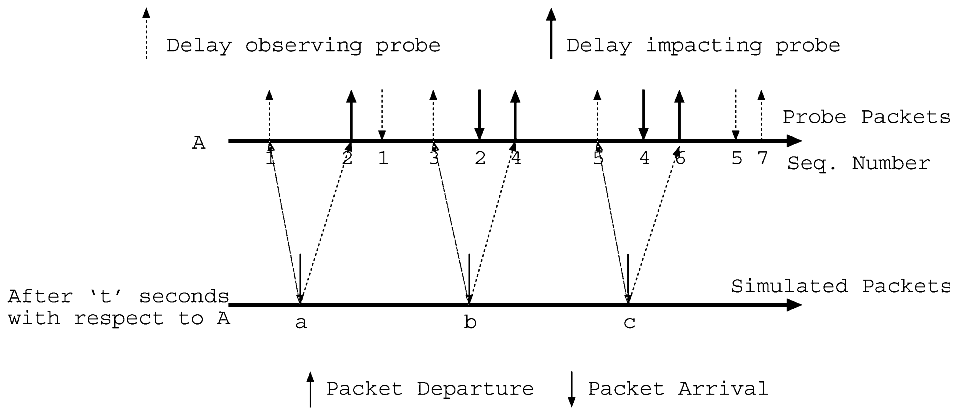


Figure 15. Delay-impacting and delay-observing probes.

collect current packet delay between source and destination. The choice of this probe will be validated by experimental data below. Delay-impacting probe is sent just after ENDE receives simulated packets larger than 20 bytes to make the same impact on the Internet traffic as the protocol/application would if it were to send data over the network.

To measure packet losses and other congestion situations, ENDE uses TCP-like timeout, t, instead of fixed timeout [20]. The timeout parameter t is given by (13)

$$t = \mathit{Avg}.\ \mathrm{RTT} + 4 \times \mathit{std}.\ \text{of RTT} \tag{13}$$

Without access to a remote machine, ENDE can inject delay-impacting probes into the network only from one of the hosts involved in the communication. Probes of appropriate size are injected into the network to model the transfer of data over the network. UDP packets of appropriate size with the TTL (time-to-live) set appropriately can be sent to model the forward path transfers accurately. For reverse path transfers, we use ICMP probes which get reflected from the remote host. Even though this models the reverse path transfers well, it imposes an extra load on the forward path that would not be present in a real experiment. Because of this reason, we expect the delay-impacting results of ENDE to be approximate. However, since ENDE relieves the requirement of access to a remote machine, experiments over a larger number of paths within the network can be conducted.

The following section shows the experimental results of ENDE in both delay-observing and delay-impacting modes. The results will show that ENDE is highly accurate in generating traces (in a delay-observing mode) and fairly accurate in a delay-impacting mode.

4. Simulation and results

In this section, we describe several sets of simulations and experiments performed to compare ENDE's packet behavior with those of the real Internet. The issues dealt in these simulations are:

- What is the proper packet size for the probe packet of ENDE?
- How does ENDE work with real-world applications?

At first, we test ENDE with various probe packet sizes using UPBAT and TPBAT (TCP Packet Behavior Analyzing Tool, TCP version of UPBAT) and compare the results with the experimental results in the real Internet. In these tests, we try to find optimum packet size for ENDE's probe packet. Then, we show that ENDE can be used to simulate general applications.

Each simulation or experiment was performed between Texas A&M University and two destination hosts, one at Syracuse University and the second one at the University of California at San Diego. To avoid data clutter, the results from University of California at San Diego are not presented here. They are available via [19]. In all simulations, ENDE was run on a Pentium 166 Mhz PC running Linux 2.0.29.

4.1. *Tests with UPBAT*

In this section, we present the results of simulations and experiments using UPBAT. To determine the probe packet size, we considered two choices, one is to use minimum ICMP packet (20 bytes) as the probe packet, and the other is to use the same size probe packet as the size of the simulated packet. To examine the two choices, we ran two sets of simulations and experiments. The first set was performed with five different sizes of simulated packets using the minimum size of probe packet. The second was performed with five different sizes of simulated packets and probe packets. We sent 3000 UDP packets at 100 milliseconds interval in each simulation or experiment. Table 2 shows the results. In the tables the left column of a packet size shows the simulation results, and the right column shows the experimental results.

The results of the experiments in Tables 2 and 3 show that the transmission delay increases as the packet size increases. As described in Section 3, we use the effective bandwidth and the effective packet size to estimate the delay of different size packets from the probe packet delay. The results of the simulations in Table 2 show that the effective bandwidth and the

Table 2. UDP results using 20 byte probe packets.

	30		100		512		1024		1450	
Packet size	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.
Number of losses	10	11	7	7	8	10	10	9	14	12
Minimum RTT (msec)	47	49	47	48	49	50	51	52	55	55
Avg FW OTT (msec)	29.8	28.2	30.1	28.4	30.8	30.0	33.1	32.8	34.6	34.0
STD of FWD (msec)	6.3	7.3	10.1	10.0	12.4	11.9	15.4	13.9	17.2	16.8
Avg RV OTT (msec)	28.1	29.1	27.9	29.2	28.4	30.0	31.6	32.9	32.9	32.6
STD of RVS (msec)	8.2	7.0	11.3	10.0	13.1	11.1	15.0	12.7	16.4	15.7
Average RTT (msec)	58.0	57.3	58.1	57.6	59.2	59.9	64.6	64.5	67.6	66.7
STD of RTT (msec)	10.4	10.3	11.3	10.0	10.1	9.0	12.0	10.8	10.2	11.0

Table 3. UDP results with probes the same size as the simulated packets.

	30		100		512		1024		1450	
Packet size	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.
Number of Losses	9	7	9	9	12	11	14	9	18	12
Minimum RTT (msec)	46	47	47	47	53	51	54	51	58	55
Avg FW OTT (msec)	29.2	28.2	30.1	28.4	30.8	30.0	34.1	32.8	35.6	34.0
STD of FWD (msec)	10.1	9.0	8.7	10.0	14.7	13.9	12.5	13.1	10.2	10.8
Avg RV OTT (msec)	28.0	30.1	28.4	29.7	29.6	29.9	33.1	32.1	35.5	34.7
STD of RVS (msec)	10.9	11.2	10.3	9.9	8.7	9.0	11.0	12.1	12.4	13.1
Average RTT (msec)	57.2	58.3	58.5	58.1	60.4	59.9	67.2	64.9	71.1	68.7
STD of RTT (msec)	10.1	10.9	10.3	11.0	10.6	11.8	10.0	9.8	14.6	15.0

effective packet size are successfully estimated and that ENDE can work with various sizes of simulated packets.

However, the results of the tests using the same size probe packet as the simulated packet, presented in Table 3, show that the differences between the delays of the simulations and the delays of the experiments increase as the packet size increases. This is due to increased queuing delays. Since ENDE sends the probe packets at a small time interval (20 milliseconds), as the probe packet size increase, the traffic of the probe packets, itself, occupies more bandwidth on the paths. Therefore, the increased probe packet size causes the increase of the queuing delay in addition to the increase of the transmission delay. Probe packets cannot be sent after ENDE receiving packets from the application. We will have to wait for a RTT before we can figure out the OTT for this packet, which is too late. The application's behavior may depend on the delay in receiving the packets at the destination. This prevents us from sending probe packets at the same rate as the application packet sending rate.

4.2. Tests with TPBAT

In this section, we present the results of simulations and experiments using TPBAT. With TCP, there are two types of packets, one is a data packet and the other is an acknowledgment. Users can control the data packet length, but the acknowledgment is always 20 bytes. For testing TCP simulation, we performed three sets of simulations and experiments. The first set was conducted using the same size probe packet as the acknowledgment, 20 bytes, the second using the same size probe packet as the simulated packet size, and the third using the probe packet which is a mean of the acknowledgment size and the simulated packet size. Since the destination host just echoes the transmitted ICMP probe, we cannot have different size ICMP packets through the forward path and the reverse path. We sent 500 TCP packets at one second interval in each simulation or experiment.

Tables 4–6 show the results of TCP tests with different probe sizes. In the tables, the left column of a packet size shows the simulation results, and the right column shows the experimental results. From Table 4, we can see that ENDE can work with TCP applications

Table 4. TCP results using 20 byte probe packets.

	30		100		512		1024		1450	
Packet size	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.
Minimum RTT (msec)	52	51	51	51	55	55	58	57	62	61
Avg FW OTT (msec)	32.0	31.2	30.5	30.8	33.3	32.3	35.6	34.8	37.2	38.0
STD of FWD (msec)	12.4	10.3	18.8	16.8	20.5	21.5	23.1	33.1	25.2	22.8
Avg RV OTT (msec)	30.4	29.9	35.2	35.5	30.6	30.9	35.1	34.6	36.2	35.7
STD of RVS (msec)	34.3	36.6	24.7	39.3	38.8	40.1	32.5	40.1	35.4	33.1
Average RTT (msec)	62.4	61.1	65.7	66.3	63.9	63.2	70.6	69.4	73.3	73.7
STD of RTT (msec)	33.5	37.8	27.4	32.6	20.2	20.3	14.9	20.8	17.6	14.0

Table 5. TCP results with probes the same size as the simulated packets.

	30		100		512		1024		1450	
Packet size	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.
Minimum RTT (msec)	51	51	51	52	55	55	59	57	64	61
Avg FW OTT (msec)	31.4	31.6	32.0	31.8	32.8	33.5	35.3	34.5	38.5	37.7
STD of FWD (msec)	21.6	17.3	12.6	13.8	15.7	26.7	43.1	40.7	35.6	26.1
Avg RV OTT (msec)	31.7	30.9	33.1	31.5	33.1	31.0	35.2	33.6	36.8	35.2
STD of RVS (msec)	24.6	27.6	34.8	29.7	34.7	32.8	12.6	32.7	65.6	53.7
Average RTT (msec)	63.1	62.5	65.1	63.3	65.9	64.5	70.4	68.1	75.3	72.9
STD of RTT (msec)	23.2	27.5	17.6	12.4	25.3	25.7	26.5	30.1	47.7	34.1

Table 6. TCP results with probes half the size of the simulated packets.

	30		100		512		1024		1450	
Packet size	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.	emul.	exp.
Minimum RTT (msec)	52	51	51	52	56	55	60	58	65	62
Avg FW OTT (msec)	30.3	31.9	32.3	31.8	31.7	31.2	35.3	33.8	38.4	37.7
STD of FWD (msec)	10.5	14.6	19.4	17.1	16.2	14.5	26.2	24.8	25.2	22.8
Avg RV OTT (msec)	31.4	30.4	31.5	31.9	32.7	33.6	34.5	34.4	37.5	36.5
STD of RVS (msec)	14.4	14.8	24.7	28.5	58.2	50.8	76.2	70.3	42.8	45.2
Average RTT (msec)	61.7	62.5	63.8	63.7	64.4	64.8	69.7	68.2	75.9	74.2
STD of RTT (msec)	14.7	17.4	12.4	11.0	33.6	30.6	63.6	50.7	27.1	30.0

using 20 byte probe packets as well as with UDP based applications. The queuing delays increase with increased packet sizes as in UDP tests.

From UDP and TCP tests described here and the previous section, we can observe the following:

- ENDE can be used to emulate a TCP or UDP application.
- The optimum probe packet length is the minimum length.

Following sections present test results using real-world applications. In all the following tests, ENDE used 20 byte probe packets.

4.3. Tests with FTP

In this section, we present experiments with FTP (File Transfer Protocol). FTP is the Internet standard for file transfer and best-know application using TCP. These experiments show than ENDE can make the same impact on the Internet traffic, since FTP is sensitive to the available network bandwidth.

In the tests, we transfered a 15 MB binary file using FTP. To compare the impact of ENDE and real FTP on the Internet traffic, we conducted two sets of experiments. In the first set, we ran ENDE and a real FTP at the same time. In the second set, we ran two real FTPs at the same time. We also tested ENDE by running it separately and comparing its results against a single real FTP session. We repeated all the experiments a number of times. To check both forward and reverse path, we sent and received the file back. Table 7 shows the results.

First, it is observed that single sessions of FTP and ENDE (run separately) agree with each other very well. It is also shown that the impact of ENDE and a real FTP session is the same on a second FTP session from the concurrent experiments. This shows that ENDE is fairly accurate in the delay-impacting mode. To evaluate the performance of ENDE at various levels of Internet traffic, we ran ENDE over various times of the day. The results are presented in Table 8. In the results, we can observe that ENDE works effectively at various times of the day (at different levels of load in the network).

Table 7. Test result comparisons with ENDE and real FTP.

Avg. transfer rate (KB/s)	on Forward path	on Reverse path
Run ENDE and a real FTP concurrently		
Emulation	76.3	92.3
Experiment	75.9	89.7
Run two real FTPs concurrently		
Experiment	78.3	90.7
Experiment	75.1	91.3
Run separately		
Emulation	123.3	143.3
Experiment	125.1	136.7

Table 8. Test results with FTP at various times.

Time	8 AM	10 AM	12 PM	2 PM	4 PM	6 PM	8 PM
Transfer rate (KB/s) on forward path							
Emulation	130	71	40	47	21	42	78
Experiment	140	60	45	43	19	42	83
Transfer rate (KB/s) on reverse path							
Emulation	80	60	25	32	20	32	77
Experiment	88	61	24	33	24	32	77

4.4. *Tests with a UDP application*

In this section, we describe tests with an application based on UDP communications and present the results. The application used in these tests is an application level protocol for adaptive video delivery [17]. The protocol was designed for delivery of real-time multimedia data. The protocol varies the sending rate based on the measured available bandwidth in the network. The implementation of these protocols is based on UDP and consists of a client and a server. Table 9 shows the results.

It is observed that the results with ENDE and real experiments agree fairly well. The table also lists some application level performance metrics “stall count” and “No. of low quality rounds”. There were no significant differences between the simulation and the actual experiments.

4.5. *Other features*

The current implementation of ENDE allows real-time traffic data to be collected as a trace. It also allows this trace data to be used as a traffic source for simulations. This allows the user to collect traffic data whenever desired and to use that data several times for repeatable experiments. This allows a consistent evaluation of different parameters of a protocol under the same conditions. If we use the actual Internet, we cannot perform consistent, repeatable experiments since the traffic is ever changing. Since ENDE makes it easy to obtain the traffic traces, the developer can collect new traces when needed.

Table 9. Test results with a UDP application.

Tests	Loss percentage	Stall count	Number of low quality rounds
Run concurrently			
Emulation	5.3282	8.3	23.3
Experiment	5.9837	7.7	23.7
Run separately			
Emulation	1.3829	10.7	11
Experiment	1.5416	12	12

We can observe delay, loss, duplication and reordering of a packets transmitted over the Internet. We have shown that a packet through ENDE experiences same amount of delay and loss as through the Internet. Since ENDE simplifies the network to a single queuing model, reordering can not happen in ENDE. Duplication is one of the pathologies in the current Internet, and ENDE ignores duplicated packets.

Accuracy of ENDE simulations is highly dependent on clock resolution. Since all events in ENDE occur with the local system timer, the accuracy of ENDE is limited by the resolution of the local system timer. We observed a few pathologies in the ICMP echoed time stamp: clock shifts and limited clock resolution. ENDE is designed to handle these cases. However, some sites echo negative ICMP time stamps. Hence, the operation of ENDE could be limited with selective destination hosts.

Because ENDE allows a user to simulate the protocols without access to remote machines, ENDE allows a more exhaustive testing/evaluation of the protocol than would be possible otherwise. We are currently exploring techniques to improve the accuracy of ENDE further in its delay-impacting mode.

Our future work will focus on extending ENDE for multicast applications. The multicast tree will be broken up into a number of unicast paths such that the union of these paths will result in the multicast tree. The probes will then be alternately sent over these different paths from the sender so as to measure the delays along different paths at different times. For example, if the multicast tree is broken up into paths A, B, C, and D, then probes will be sent over A, B, C, D and then A, . . . and so on. This will result in sampling delays over the paths at different times. Accuracy of the these delays will depend on the number of paths and it may be necessary to send probes closer than 20 ms to obtain accurate traces of delay over the multicast tree. We are currently investigating techniques for modeling the delay-impacting mode of ENDE in a multicast application. We are also considering IGMP and related multicast based protocols for extending ENDE into multicast applications.

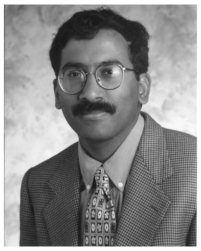
5. Conclusions

We introduced a new tool, called ENDE, for emulating end-to-end delay behavior in the Internet. Using ENDE, we can test network protocols or applications on a single system connected to the Internet without requiring login access on a remote host. Our approach gives the following advantages of simulation: simplicity of use, no need of complex hardware settings, no need of modification of real-world applications and real-time traffic generation. We validated ENDE's accuracy by using delay measurement tools and real-world applications. We showed that ENDE can be used to accurately estimate one-way transit times between two hosts. ENDE was also shown to be fairly accurate in modeling the delay-impacting behavior of the application/protocol under test. Abnormal ICMP packet behavior (time shifts, negative timestamps etc.) was observed and ENDE was shown to handle these cases effectively. We are currently extending ENDE to model multicast delivery.

References

1. S. Keshav, "REAL: A network simulator," Technical Report 88/472, Department of Computer Science, University of California at Berkeley, CA, 1988.

2. L.S. Brakmo and L.L. Peterson, "Experiences with network simulator," in Proc. of SIGMETRICS, Philadelphia, PA, 1996. Available via http://www.cs.arizona.edu/xkernel/www/people/brakmo.html/sigmetric96.ps.
3. Network simulator (Ns), University of California at Berkeley, CA, 1997. Available via http://www-nrg.ee.lbl.gov/ns/.
4. N.C. Hutchinson and L.L. Peterson, "The x-kernel: An architecture for implementing network protocols," IEEE Transactions on Software Engineering, pp. 64–76, January 1991.
5. S. Floyd, "Simulator tests," Technical Report, Lawrence Berkeley Laboratory, Berkeley, CA, 1996. Available via ftp://ftp.ee.lbl.gov/papers/simtests.ps.Z.
6. V. Paxson and S. Floyd, "Why We Don't Know How to Simulate the Internet," in Proc. of the 1997 Winter Simulation Conference, Atlanta, GA, 1997. Available via ftp://ftp.ee.lbl.gov/papers/wsc97.ps.
7. L.L. Peterson and B.S. Davie, Computer Networks, Morgan Kaufmann Publishers: San Francisco, CA, 1996.
8. D. Sanghi, A.K. Agrawala, and B. Jain, "Experimental assessment of end-to-end behavior on Internet," in Proc. of IEEE Infocom' 93, San Francisco, CA, pp. 867–874. March 1993.
9. J-C. Bolot, "End-to-end packet delay and loss behavior in the Internet," in Proc. of SIGCOMM' 93, San Francisco, CA, pp. 289–298, September 1993.
10. S. Keshav, "Packet-pair flow control," Technical Report, AT&T Bell Laboratories, Murray Hill, NJ, 1994. Available via ftp://ftp.research.att.com/dist/qos/pp.ps.z.
11. M. Mathis and J. Mahdavi, "Diagnosing internet congestion with a transport layer Performance Tool," in Proc. of INET'96, Montreal, Canada, June 1996.
12. R.L. Carter and M.E. Crovella, "Dynamic server selection using bandwidth probing in wide-area networks," Technical Report BU-CS-96-007, Computer Science Department, Boston University, March 1996.
13. pathchar—A tool to infer characteristics of Internet paths, University of California at Berkeley, CA, 1997. Available via ftp://ee.lbl.gov/pathchar.
14. V. Paxson, "End-to-end Internet packet dynamics," in Proc. of SIGCOMM 1997, 1997. Available via ftp://ftp.ee.lbl.gov/papers/vp-pkt-dyn-sigcomm97.ps.Z.
15. V. Paxson, "On calibrating measurements of packet transit times," in Proc. of SIGMETRICS, 1998. Available via ftp://ftp.ee.lbl.gov/papers/vp-clocks-sigmetrics98.ps.gz.
16. S. Keshav, "A control-theoretic approach to flow control," in Proc. of SIGCOMM'91, pp. 3–15, Sept. 1991.
17. A.R. Madhwaraj and A.L.N. Reddy, "Adaptive best-effort protocol for video delivery," Technical Report, Texas A&M University, College Station, TX, 1998.
18. L. Rizzo, "Dummynet: A simple approach to the evaluation of network protocol," ACM Communication Review, Vol. 27, pp. 31–41, January 1997.
19. I. Yeom, "ENDE: An end-to-end network delay emulator," Master's thesis, Texas A&M University, College Station, TX, 1998. Available via http://dropzone.tamu.edu/∼ikjun/thesis.ps.
20. V. Jacobson and M.J. Karels, "Congestion avoidance and control," Computer Communication Review, Vol. 18, No. 4, pp. 314–329, August 1988.



A.L. Narasimha Reddy received his B.Tech. degree in Electronics and Electrical Engineering from the Indian Institute of Technology, Kharagpur, India, in August 1985, and the M.S. and Ph.D. degrees in Electrical Engineering from the University of Illinois of Urbana-Champaign in May 1987 and August 1990 respectively.

During 1990–1995, he was a Research Staff Member at IBM Almaden Research Center in San Jose, where he worked on projects related to disk arrays, multiprocessor communication, hierarchical storage systems and video

P1: vendor, GHS
Multimedia Tools and Applications KL1280-03 April 4, 2001 19:19

servers. He has been an associate professor in the department of Electrical and engineering since fall 1995. His research interests are in Multimedia Systems, Storage systems, Network QOS and Computer Architecture. He has received an NSF Career Award in 1996.

Dr. Reddy is a senior member of IEEE Computer Society and is a member of ACM.

Ikjun Yeom received the B.S. degree in Electronic Engineering from Yonsei University, Seoul, South Korea. He worked at DACOM co. located in Seoul between 1995 and 1996. He completed his M.S. degree in Computer Engineering at Texas A&M University at August 1998. He is now a Ph.D candidate in Computer Engineering at Texas A&M University.