

# The Design and the Implementation of an Emulated WAN

Elliot Limin Yan  
Computer Science Department  
University of Southern California  
Los Angeles, CA 90089  
E-mail: [lyan@caldera.usc.edu](mailto:lyan@caldera.usc.edu)

## Abstract

*Wide-Area Networking (WAN) becomes more pertinent to a distributed system design. We need a better model to investigate the scalable distributed system design and efficient communication protocols. We propose to emulate a wide-area network on a local-area network. It can serve as the test-bed for scalable distributed applications and wide-area networking protocols. The WAN emulation is achieved by providing propagation delay, packet drops, and network bandwidth between hosts.*

## I. Introduction

Much research work in distributed systems relates to the issues of networking and scalability. Communication is usually the biggest bottleneck within a distributed system. The improving efficiency of networking protocols will inevitably increase the performance of the system. In addition, scale has becoming increasingly important, since Internet has been expanding and contains more resources than ever before. For a system to grow, it is fundamentally important for a distributed system to scale well over the Wide-Area Network (WAN), while having efficient communication across wide-area networks.

Modern distributed systems depend a great deal on the Internet. Thus we have to systematically study the interaction between the Internet and networking protocols and topologies. Because Internet has becoming increasingly commercial, it is nearly impossible to perform experiments that will disrupt normal activities of the network. Traditionally researchers have studied WAN through simulations that inevitably make assumptions about various aspects of the network. While this approach is convenient, it does not portray the Internet as accurate as we think. An alternative approach is to emulate the internet, in real time, on a local area network. We can use real network applications to generate internet traffic and configure different network topologies analogous to the ones found on the Internet. Then the emulator can be used to study the behavior of existing networking protocols on WAN; it can be used to serve as the test-bed for new transport level protocols; it can also be used to investigate scalability and measure performance of a given distributed system. We have also examined UNIX kernel's ability to handle real-time scheduling. since this project involves stretching the UNIX kernel to handle real-time events.

This paper is organized as follows. In section II, we look at the some of the related work. In section III, we investigate some interesting design decisions. In section IV, we describe the implementation of the wide-area emulator. In section V, we present the results of a number of experiments employing this emulator. In section VI, we discuss our conclusion and present some future research directions.

## **II. Related Work**

BBN offers a hardware based long-link emulator that emulates two or more unidirectional optical links. The emulator offers two optical interfaces. It can emulate link delay up to 200 ms, programmable in increments of 1 microsecond resolution. It also provides various error patterns and error rates [5].

BBN is far too expensive. The emulator itself is essentially a SUN Sparc station with special hardware and software installed. It has the complexity of a router. It is targeted for special applications involving optical-links. As it does not emulate packet drops, it cannot realistically emulate the existing Internet. Thus it is not valuable for evaluating distributed system design and network analysis. Our wide-area network emulator will focus on the faithful emulation of the Internet. It is our goal that accurate performance analysis for wide-area distributed systems and communication protocols can be done on our emulator.

## **III. Design of the WAN Emulator**

To instrument WAN emulator, we must detach our design from any logistic assumptions such as traffic flow models, network topologies, etc. about the network. We concentrate on the physical characteristics of the wide area network. Traditionally the simulation based approaches derive their result from the models and abstractions they define. Often their results are under scrutiny if their models are called in question. Therefore we make no logistic assumptions in our design.

To emulate a wide-area network on a local-area network, we must carefully characterize the differences between their behavior. One major difference is the network propagation delay. Obviously it takes longer for a packet to travel from point-to-point within a WAN than from within a LAN. Secondly, on a LAN, rarely packets are ever dropped whereas packets on a WAN are dropped in a higher percentage. In addition the Internet is composed of various network hardware with different data bandwidth which results in different transmission delay and queuing delay of data. These three differences are fundamental enough that they have to be accurately mimicked by the emulator. Other aspects of the network, such as communication error, are currently neglected. Its occurrence is rare enough to be ignored for now. Thus our emulator configures each network link according to the given propagation delay, packets drop rate, and link bandwidth.

Notice all I have mentioned so far are the characteristics of point-to-point wide area network behavior. It is our intention not to make further assumptions about other aspects of

network. Our goal is eventually that experiments will be carried out by setting up the desired network topology physically. By wiring hosts together with various delay, drop rate, or bandwidth applied to the network links through the use of this point-to-point emulator, we believe that this emulation scheme ought to make a local-area network look physically very close to a real wide-area network. Through the use of real-time traffic generated by distributed systems or applications such as TELNET or FTP, we believe that we can observe behavior close to what we would see on the Internet. By experimenting with different networking protocols, different network topologies, different RPC mechanisms, different distributed system design, etc. on our emulated WAN environment, we believe that our emulator can provide a fair prediction of how a system is going to behave on the Internet. With this emulated environment, we are also able to see the outcome and the performance of distributed applications in real time. We believe this approach will give a realistic emulation of the actual Internet. Entrusting this environment to offer a foundation for profiling of wide-area distributed applications, one can make alternative design decisions or fine tune the application programs before releasing them to the public.

The algorithm for this emulator is simple. Before releasing packets onto the network, we intercept them and put them on a queue. For the given network bandwidth, we can find each packet's transmission delay, which is bits-in-the-packet/bandwidth. Along with the given propagation delay of the emulated link between a pair of hosts, the actual delay, which is sum of propagation delay and transmission delay, for a packet traveling from one host to another can be determined. The packet is held on the queue for the duration of the network delay plus the queuing delay, the amount of time waiting for the preceding packets on the queue to be sent. Then the packet is flushed onto the network. This will correctly provide the delay and bandwidth aspects of WAN. Packets on the queue are dropped randomly according to the given drop rate. A detailed account of our implementation of this algorithm is discussed in the implementation section.

The major obstacle of this approach is the control of timing. In essence, the environment is a soft real-time process since each packet has fixed delay bound in order to ensure the correct and constant frequency of outgoing packets. In addition it requires a fixed upper bound as well, so that the outgoing packets do not leave earlier than they should. Much of the recent work done in real time concentrates on hard real time where the CPU is non-preemptive and all of the system events and their response are calculated a priori [1]. The static scheduling strategy is undesirable in this case since the system events are not critical events as in the case of hard real time systems, and in our environment workstations should proceed with its normal work without dedicating themselves to process real time events. Therefore we must resort to nondeterministic scheduling using a modification of the existing UNIX's system events scheduler. Our solution is that in order to have more control over kernel scheduling, we need scheduling resolution in millisecond precision. However, many variants of UNIX, depending on vendors, do not support millisecond level scheduling. So this problem needs to be addressed. Since we need to maintain the workstations under the normal working environment to generate realistic network traffics, the scheduling mechanism should not add unreasonable amount of overhead to the entire system. Thus the usability of this refined scheduler has to be verified. The details of our particular implementation will be mentioned in the implementation section as well.

## IV. Implementation

We implemented the emulator on SUN Sparc 10's with SUNOS 4.1.3, a derivant of BSD UNIX. Thus our emulator should work on most BSD-based UNIX. The emulator is tested on SUN Sparc 10's and SUN Sparc 1+'s.

The emulator project requires the modification of the SunOS kernel. A pseudo device is created so that a user level program can communicate with the kernel to change the emulation parameters (delay, bandwidth, drop, etc.). In addition, users can specify the destination, in host name or IP address, to which the WAN emulation applies. In other words one can choose to apply WAN emulation between one or more pairs of hosts with distinct propagation delay, drop rate, or bandwidth. Only packets traveling between these specified hosts are subject to emulation, while other packets are forwarded to network directly.

There are two modes of operation in the emulator. One is the point-to-point mode, and the other is the gateway mode. In the point-to-point mode, WAN emulation is carried out only between hosts. The gateways between them are not affected by the emulation. Under gateway mode, one can specify that any packets going to a particular gateway are affected by the emulation. For example, let us assume the destination host can be reached by going through one or more gateways. In point-to-point mode, all packets going to that destination are delayed. But none of the packets going to other destinations are delayed. (See Figure. 1) Thus if we apply 100 ms delay from host A and host C, only the packets going from host A to host C are delayed. The total propagation delay between A and C, according to Figure. 1, is 133 ms. The propagation delay between host A and host B is not changed, nor is the propagation delay between host A and the gateway. However in the gateway mode, we can specify that any packets going through a particular gateway are subject to WAN emulation, (i.e. delayed or dropped). Thus any of the hosts connected to that gateway is affected by the delay. (See Figure. 2) Therefore when we issue the command that we want to add 100 ms delay to all packets going to the gateway, the propagation delay between host A and host C becomes 133 ms, the propagation delay between host A and host B changes to 123 ms, and the propagation delay between host A and the gateway is 103 ms. This additional 100 ms delay affects all of the hosts connected to that gateway. The gateway mode is essential in constructing our large emulation environment. In addition we can specify the amount of buffers allocated for the output queue on the gateway. Thus we will be able to observe the effects buffer size has on the network. We can specify the entire network topology by defining delay, drop rate, output queue size, and bandwidth between gateways.

The emulator functions as follows. The emulator hijacks all of the outgoing packets by intercepting network interfaces. All packets from the network layer (IP, ICMP, etc.) have to be forwarded to the link layer, which are the network interfaces. All of the packets from the network layer are sent to the underlying networking hardware by calling the output routine specified by the network interfaces. By intercepting the network interfaces and modifying its output routines, we insert an additional level indirection between the network layer and the link layer. This gives us full control of over the packets outputted to the network via any of the network interfaces. Notice that the emulator intercepts all outgoing packets, not just the ones of TCP/IP, even though the packets of interest are of TCP/IP. By examining the packets passed into the output routine as parameters, we can find out its destination address, or gateway address when the emulator is

running the gateway mode. Therefore we know whether or not a packet is subject to delay or drop. If a packet is not subject to emulation, then it is immediately forwarded to the real output routine of the network interface.

Internally the emulator keeps two hash tables. One of them hashes the network interfaces. It is entirely possible that a host possesses more than one network interfaces, and we might like to intercept and apply WAN emulation on all of them. When a packet arrives at the emulator and the emulator decides the packet is not subject to emulation, it has to forward it to correct interface output routine instantly. This hash table is used to speed up the search for the correct network interface. The other hash table is for network channels. Network channels in this context is defined as the traffic between a pair of hosts. For instance, if we have multiple TELNET and FTP sessions between a pair of hosts, then all of the traffic generated by these TELNET and FTP sessions belongs to the same network channel. For each channel, we keep a separate data structure including the emulation parameters, statistics, the network interface the channel uses, and a queue for storing delayed outgoing packets. When a packet is subject to emulation, it is put onto the queue of its channel. The hash table is used to for searching which network channel a particular packet belongs to. From this design, we can create multiple wide area emulation sessions of distinct pairs of hosts with different delay, drop rate, and bandwidth while sharing the same network interface. The biggest advantage of keeping a separate queue for each channel is that the send time of each of the enqueued packets is naturally ordered. The system scheduler can deal with each of the channels separately. It is worth noting that this feature is not required to set up the proposed WAN experiments, but it is an interesting feature nevertheless.

In BSD-styled UNIX, the system is interrupted by periodic clock ticks. Most of system events which are less time critical are handled by the kernel's softclock, except extremely mission critical events which are handled by the hardclock. The hardclock must process each of its hardware clock tick, else system will slowly lose track of time. The softclock does not have that limitation. However the timing will not be as accurate. The timeout mechanism is essentially a softclock interrupt handler which certain timer events can be scheduled to happen some clock ticks from the moment a timeout is scheduled [4]. Since scheduling to send a packet in our emulator does not need to have extremely precise timing, the emulator can tolerate minimal amount of scheduling inaccuracy given by softclock and timeout mechanism. Thus timeout mechanism is used extensive in our WAN emulator. When a packet is enqueued, its delay is the sum of the propagation delay, transmission delay, and network queuing delay. Each of the packets will be timestamped with its actual send-time, which is the sum of the time the packet has arrived at the queue plus its delay. We schedule a packet to be sent after its appropriate amount delay using the timeout mechanism if no other packets are scheduled to sent within its queue. If another packet is scheduled for timeout already, then it does nothing. Once the timeout has occurred, the scheduled packet will be sent along with other packets on the queue with send-time less or equal to the current time. Then we schedule the next available packet on the queue for timeout for the difference between its send-time and current time. Usually if this difference is less than two milliseconds, as an optimization, the emulator sends this packet out to the network without going through timeout since the emulator can tolerate this amount of relaxation. This optimization also saves many timeouts during a bulk data transfer where data travels in a long continuous stream. It is important to note that we use timeouts to schedule each network channel independently. Scheduling done on one queue is logically independent from scheduling done on a different queue

One of the problem with BSD timeout scheduler is that the softclock runs 100 ticks per second, which is 10 ms scheduling resolution. We would like to have finer resolution of timeouts so that we can schedule packet sends in milliseconds precision. This also reduces the amount of softclock deviation in each tick. So kernel clock routines are modified. Since softclock and timeout scheduling is used extensively within the kernel, ten times more precision will incur ten times more context switches. We designed a program quicksorting 400,000 integers as the performance test to caliber the system performance. The performance overhead is about 3% with the new timer resolution. The reason for such a low overhead is that SUN's internal clock runs in microsecond resolution and most of timeout events can be processed in very small intervals. The only events scheduling at the regular basis are the timeout events for forwarding delayed packets. However they are usually finished in very short time and can relinquish the rest of the processing power to the user level processes. Thus this new scheduler incurs minimal overhead.

## V. Performance and Testing

We performed a number of tests to verify correctness of the emulator.

The first test is the ping test, which each ping session outputs ICMP message to the target hosts, and round-trip time of each packet is measured. Ten sessions of pings runs concurrently on a host, each with different delay, drop rate, bandwidth, and ICMP packet size. All of their round trip time are, as expected, within 1 ms deviation from the precise value for each session

The second test is the ftp test. A file of two megabytes is ftped from one site to another within local area network with the following parametric specification:

| 1-way<br>delay(ms) | drop<br>rate | bandwidth (Mbs) | time (s) | throughput (K/s) |
|--------------------|--------------|-----------------|----------|------------------|
| 0                  | 0            | 10              | 15       | 140              |
| 100                | 0            | 10              | 19       | 110              |
| 100                | 0            | 1               | 21       | 100              |
| 0                  | 0            | 0.5             | 33       | 65               |
| 100                | 0            | 0.5             | 33       | 65               |
| 0                  | 0            | 0.25            | 68       | 31               |
| 100                | 0            | 0.25            | 68       | 31               |
| 300                | 0            | 0.25            | 68       | 31               |
| 100                | 2            | 10              | 70       | 30               |
| 100                | 2            | 1               | 88       | 24               |
| 100                | 2            | 0.5             | 1e+2     | 22               |
| 100                | 2            | 0.25            | 1.1e+2   | 19               |

Observe the above data. In the case of FTP or bulk data transfer, propagation delay contributes almost nothing to overall time, since under an ideal environment, data travels almost like as one long continuous packet. Longer transmission delay and queuing delay resulted from

low bandwidth is the major factor for difference in time and throughput. This behavior agrees the ideal behavior of bulk transfer. Notice tests with 2% of packets drop, there is also 50% difference in time and throughput with their counterparts without 2% packet drop. This agrees completely with the traditional design of TCP which employs slow start algorithm [4]. The slow start algorithm is very sensitive to extremely small packet drops and much of the recent research aims at redesigning TCP for WAN [7].

A number of tests were performed to analyze the correctness of hitbox. With the command: "hitbox ip\_address", we can query the current status of hitbox. Amongst the statistics we see the number of packets dropped, the amount of buffers used. We can clearly see that with FTP traffic when output buffer is set smaller, the amount of packets dropped increases. With these statistics, we can inspect the correctness of output buffering.

The source code can be FTPed from [catarina.usc.edu/pub/lyan/delayemulator.tar.gz](http://catarina.usc.edu/pub/lyan/delayemulator.tar.gz).

## **VI. Conclusions and Future Directions**

In this paper, we have presented design and implementation an emulated WAN environment. From our preliminary results, the emulator performs as expected. More tests are needed to analyze the emulator in detail. This emulator will serve as the basis for establishing a emulated WAN environment for doing wide area performance analysis of scalable distributed system or network protocol analysis accurately. Much more work needs to be done in perfecting such an environment. As a by-product of this project, UNIX kernel's softclock mechanism has been successfully modified to do some lightweight real-time scheduling.

Currently there are a number of projects which are using the WAN emulator for their performance analysis. Amongst them is the Prospero Resource Manager, a scalable resource manager which can allocate resource and processing power from large networks [6]. A performance analysis of C-Slip is in progress. C-Slip claims to be efficient over phone lines. But with digital phones lines, small data packets are collected in bulk and send over phone together. Is C-Slip still efficient with under greater network delay? The question is yet to be answered. A study on RSVP, a real-time wide area multicasting protocol is planned [2]. Through simulation, RSVP is supposed to provide reliable and fair multimedia communication. This fact is yet to be verified. A study on TCP Vegas, a TCP optimized for wide-area networking, is to take place soon [7]. TCP Vegas claims to have good throughput over wide-area network. However if the entire Internet implements TCP Vegas, does TCP Vegas still perform better in throughput? Lastly a study of phase effects in packet-switched gateway is also planned [3]. Floyd and Jacobson claim phase effects occur on the Internet from their simulation and theoretical derivation. Nobody has ever been able observe it on the Internet due to its random nature. Thus we plan to create an environment pure enough to verify or disprove this claim. We hope that in the future all of the above questions will be answered. As our emulator matures, we would expect more applications to take advantage of our emulated WAN environment.

### **References:**

- [1] A.K. Agrawala, D. Mosse, and O. Gudmundsson. The MARUTI System and its Implementation. TCOS Newsletter, 5(3)-18, 1991.
- [2] D.Clark, S. Shenker, L. Zhang. Supporting Real-Time Applications in an Integrated Service Packet Network: Architecture and Mechanism. Proc. ACM SIGCOMM'92. August, 1992.
- [3] S. Floyd and V. Jacobson. Traffic Effects in Packet-Switched Gateways. ACM SIGCOMM Computer Communication Review. 1991.
- [4] S. Leffler, M. K. McKusick, M. Karels, and J. S. Quarterman. The Design and implementation of the 4.3 BSD UNIX Operating System. Addison-Wesley Publishing Company, Inc. October 1990
- [5] W. Milliken. Product Overview on BBN Long-Link Emulator. 1993.
- [6] B.C. Neuman and S. Rao. Resource Management for Distributed Parallel Systems. Proc. of the 2nd International Symposium on High Performance Distributed Computing, Spokane. July, 1993.
- [7] L.L. Peterson. Private communication on TCP Vegas, 1993.



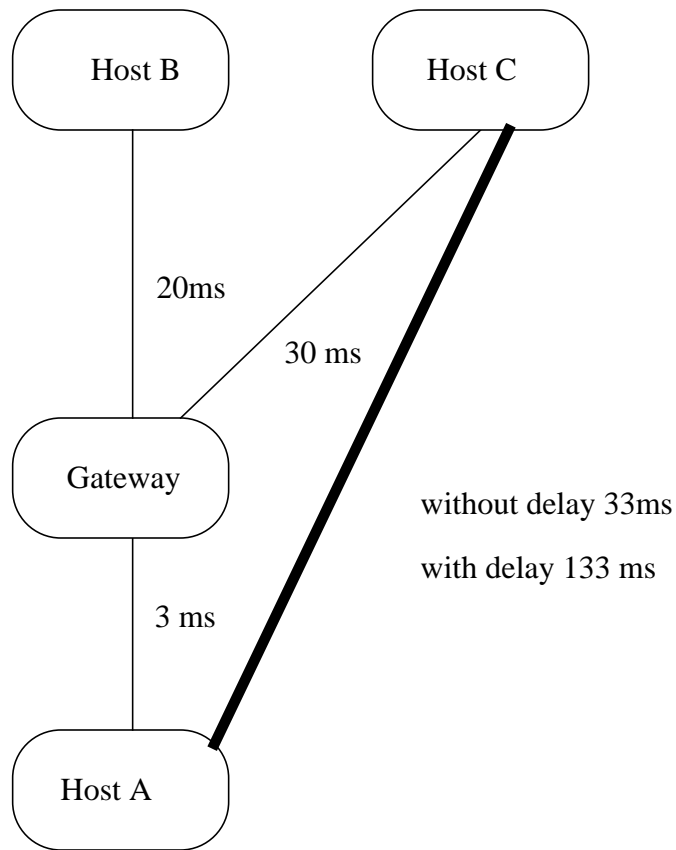


Figure 1. Point-to-Point Mode with 100 ms delay from host A to host C. The numbers associated with the links are their actual propagation delay. The thin lines represent physical links. The bold line indicates the connection between hosts.

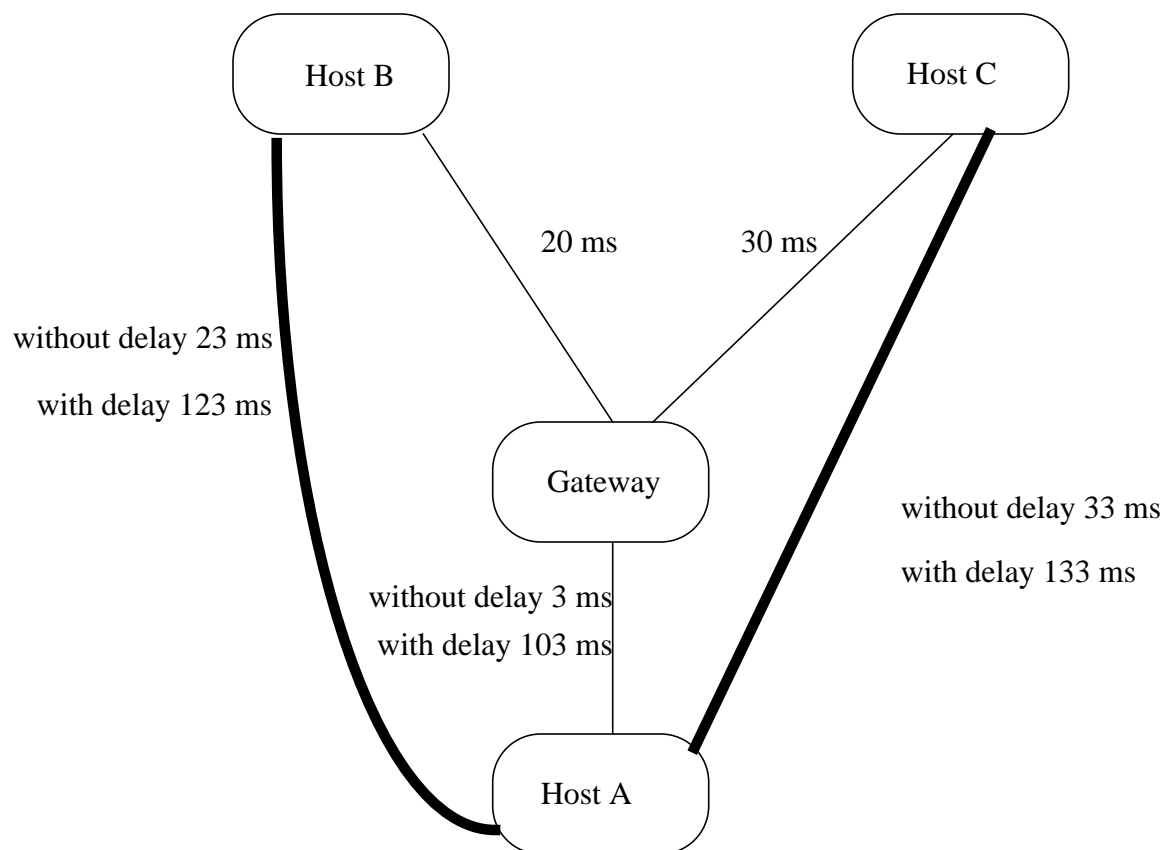


Figure. 2 Gateway Mode with 100 ms delay from host A to the gateway. The numbers associated with the links are their actual propagation delay. The thin lines represent physical links. The bold lines indicate the connections between hosts.