# Scalable Network Path Emulation

Shilpi Agarwal Joel Sommers Paul Barford
*University of Wisconsin-Madison*
*shilpi,pb,jsommers@cs.wisc.edu*

## Abstract

*Laboratory-based experimentation is an increasingly popular method for conducting network research since it enables implementations of network systems and protocols to be evaluated. Most research conducted in lab-based environments requires the faithful reproduction of wide area network conditions. An important step toward satisfying this requirement is the creation of paths between nodes in the lab that have the same characteristics as paths between nodes in the Internet. In this paper, we describe and evaluate a new, highly scalable, software-based path emulation tool called NetPath. We describe the design and implementation of NetPath, which features fixed and probabilistic packet propagation delay emulation, probabilistic bit errors, probabilistic packet loss, packet duplication, and packet reordering capability. Through a series of controlled laboratory experiments, we demonstrate that NetPath offers over three times the loss-free throughput capacity of other popular software-based path/network emulators. We show that under moderate load NetPath's mean propagation delay emulation precision is within 1% of a hardware-based reference emulator. This result represents a significant improvement over other software-based emulators. We illustrate how, relative to our hardware-based reference, NetPath improves application traffic behavior over other software-based emulators. Finally, we demonstrate and characterize NetPath's ability to provide path emulation simultaneously on multiple physical links. This capability, which is facilitated through the use of our link configuration tool, enables laboratory system resources to be more efficiently utilized.*

## 1. Introduction

An important challenge in the networking and distributed systems research communities is the development of innovative testbed environments that enable new technologies to be evaluated thoroughly and accurately [9, 13, 15]. One of the primary purposes of testbed environments is to create a spectrum of *realistic* conditions that might be found in operational networks. Over the past five years, several important network testbed infrastructures have been developed that are scalable, flexible and offer greater realism than traditional analytic modeling or simulation tools.

The current generation of network research testbed environments can be roughly divided into two categories. The first type of testbed setting, *in situ* infrastructures, consists of managed host systems distributed across the Internet. Examples include PlanetLab [8] and the RON testbed [5], which in some ways follow in the tradition of NIMI [3] and other widely deployed measurement infrastructures. These environments provide highly realistic network conditions since hosts are deployed in the Internet. While they are ideal for addressing questions on Internet structure and behavior, the inherent lack of ability to reproduce conditions across experiments limits their utility. The second category of testbed environments include those based in laboratories [1, 7, 18, 23, 24]. A canonical example is the Emulab testbed at the University of Utah which features a large set of commodity workstations that can be flexibly configured to conduct a wide variety of experiments [24]. While these environments offer the capabilities of end-to-end node configuration and complete instrumentation and repeatability, creating conditions representative of wide area networks continues to be a significant challenge.

There are many factors that contribute to creating realistic wide area network conditions in a network research laboratory. We argue that the accurate emulation of link characteristics is one of the most important and challenging. We define *link emulation* as the task of recreating the conditions experienced by packets on a single link physically (or virtually) connecting two nodes in a network. As such, link emulation is primarily focused on the *impact* of layers 1 and 2 in the network protocol stack. In a lab environment this means emulation of two properties:

• **Propagation delay**, the speed of light delay for each bit due to the physical distance between two nodes, is modeled as a constant delay per bit.

• **Bit errors** are caused by noise and other factors along the physical path between two nodes and are modeled as a probabilistic change of state per bit.

Link emulation capability is of particular importance in experiments with topological configurations that are meant to recreate a specific wide area network. In these cases, some nodes are configured to act as routers and others as end hosts, so the lab infrastructure must be able to emulate the desired link characteristics between these nodes.

In experimental configurations that do not attempt to instantiate specific topologies, it is often desirable to emulate the network path characteristics between hosts. We define *path emulation* as the task of recreating the physical conditions experienced by packets on an end-to-end path between hosts in a network. Path emulation is a superset of link emulation and includes the following properties:

• **Packet latency** is the aggregate of signaling delay (caused by the network interface cards), the propagation delay (caused by the physical distance between two hosts) and the queuing delays (caused by routers along the path). The sum of these delays is modeled as a probabilistic delay per packet.

• **Packet loss** is the drop of a packet due to a full bottleneck queue on the path between two hosts, and is modeled as a probabilistic event.

• **Packet duplication** is modeled as a probabilistic cloning of a packet on the path between two hosts.

• **Packet reordering** shuffles packets from the order in which they arrive on a link and is modeled as a probabilistic event.

Path emulation differs from *network emulation* which attempts to "subject each packet to a delay, bandwidth, and loss characteristics according to a target topology" [23]. Path emulation has no notion of a core network topology and is therefore less ambitious in its objectives. Path emulation also does not attempt to emulate router characteristics such as statistical multiplexing, queuing and switching capabilities. Such characteristics vary widely in the Internet depending on specific hardware implementations. Path emulation targets the basic causes of packet variability on multiple-hop, wide-area connections between hosts. Finally, while mimicking link capacity is often listed as a distinct feature in network emulators, we do not consider it a distinct component of path emulation. The essence of capacity limitation is to add latency to each bit equal to the inverse of a specified rate. Thus, experiments requiring links of certain capacities can be accommodated through the propagation delay mechanism.

### 1.1. Current Methods for Path Emulation

The basic task of path emulation can be abstracted to the ability to flexibly buffer packets transmitted between two nodes. Buffering enables delays to be assigned to individual packets. Also, while packets are buffered, they can be duplicated, reordered or dropped. Over the years, many software-based systems have been developed for commodity hardware with the general objective of providing these functions [4, 6, 10, 11, 12, 14, 20, 21, 23]. Each of these systems differs from the others either in terms of their emulation objective, performance, or in how they are meant to be deployed for use in experimental environments.

There are many possible approaches to both link and path emulation. One extreme method for implementing propagation delays is the use of large spools of cable to effect specific delays between systems. While this approach has obvious drawbacks, it provides accurate delays and is actually used in commercial labs. Another approach is to use specialized hardware developed for the purpose of delay emulation such as that offered by Spirent Communications [2]. Such systems provide precise link delays, probabilistic bit errors and probabilistic packet loss.

The most common approach to link and path emulation is to use general purpose computing hardware to run link and path emulation software. Two widely-used software-based link/path emulation tools are Dummynet [21] and NIST Net [12]. Dummynet is an in-kernel network emulation system that can be run on end hosts running other applications, or on a dedicated system interposed between two others in a lab. NIST Net is also a kernel-based emulation and can be interposed much like Dummynet. Like other network emulators, including [10, 23], NIST Net was developed with the high level design objective of being a "network-in-a-box" [12]. We consider this to be an overly ambitious objective for many reasons (*e.g.,* see [16]). While software-based emulators like Dummynet and NIST Net offer many useful features and are cost effective, they must provide a high level of accuracy and performance across a broad range of use scenarios.

### 1.2. Our Approach to Path Emulation

In this paper we describe NetPath, a new network path emulation system designed to provide performance approaching hardware-based solutions—well beyond the capabilities of other software-based systems. NetPath offers the ability to add fixed and probabilistic packet delay, probabilistic packet loss, probabilistic packet duplication and packet reordering to links in a network. NetPath's implementation is based on the Click modular router platform [17]. Click is an extensible, high performance packet processing system that runs on commodity hardware. Click offers several of the basic capabilities required for path emulation, making it a natural platform for our work. The primary challenge in NetPath's development was to create a reliable, high capacity queuing system that significantly

extends the basic packet buffering capabilities in Click. We extended Click to seamlessly and efficiently utilize both RAM and disk resources on the NetPath host system. We also briefly describe VICT (virtual interposition configuration tool), a configuration tool we developed to allow NetPath to easily be used in a variety of testbed setups.

The objectives of our evaluation of NetPath are to demonstrate its raw performance capabilities from the perspectives of precision, throughput, and scalability. We also compare these capabilities with Dummynet, NIST Net and Modelnet [23]. Our results show that our new queuing module yields high throughput rates and supports over three orders of magnitude longer emulated delays without packet loss versus the standard Click queue. We also show that under a moderate traffic load NetPath's delay emulation precision is consistently within 1% of a hardware-based reference emulator. Our comparative analysis shows that NetPath offers over three times the loss free throughput capacity and orders of magnitude better delay precision than other software-based reference emulators. We demonstrate how NetPath improves application traffic behavior over other software-based emulators, relative to our hardware-based reference. Finally, we investigate the scalability of NetPath under multiple link configurations.

### 1.3. Implications

There are several key implications of this work. First, the ability of NetPath to provide higher throughput and better precision than other software-based network emulation systems means that evaluation of delay sensitive protocols and systems can be conducted under more realistic conditions. Second, the scalability of NetPath and its ability to accommodate high-throughput multi-link configurations means that resources in research labs can be used more efficiently. Third, the automated network configuration capability offered by VICT should enable NetPath to be used widely in a variety of laboratory environments. NetPath is freely available for download and evaluation.

## 2. Architecture and Implementation

In this section, we describe the design objectives and general architecture of NetPath. We then describe implementation details of NetPath, including specific challenges faced in its development. We also briefly describe the configuration tool VICT.

### 2.1. Architecture

Network laboratory testbeds can contain thousands of individual links between network interfaces on routers, switches, and end hosts. Our project is framed in the context of balancing the desire of testbed operators to use a limited number of commodity host systems for path emulation and the need of researchers to accurately create a range of network path conditions.

The design objectives for NetPath were to create a system that could (1) offer high precision link and path emulation capability, (2) operate effectively on network links beyond 100 Mb/s, and (3) operate effectively on multiple network links at the same time. Practical considerations related to the use of NetPath in lab environments led to extending the design objectives to include the ability to easily configure VLANs and/or MPLS paths to deflect packets on specific links through a node running NetPath.

### 2.2. NetPath Implementation

Path emulation as defined above implies the need for a packet processing system as the basis for our implementation. For this functionality we chose the Click modular router[1]—an open source platform for building packet switching system that runs on commodity hardware [17]. Click offers three key features that make it a natural choice for NetPath. First, Click has high performance packet switching capabilities due to its kernel-based implementation and the ability to operate in polling (instead of interrupt) mode with respect to the network interface card (NIC). Second, Click is designed for extensibility. Click-based systems consist of primitive elements which are arranged to perform some desired function. The Click distribution provides a library of elements that can be extended for new functionality. Third, Click is designed to be easily configured through a declarative language that defines inter-element connectivity. The configuration language also supports higher level abstractions through definitions of compound elements.

**2.2.1. Basic Link Emulation.** Basic link emulation consists of the following: read packets from the NIC, delay them in a queue for a specified fixed interval and then send them out on a network interface ensuring that they are headed for the proper destination. These functions require the selection of appropriate Click elements, which are illustrated in Figure 1. For good performance, the first element in our system is `PollDevice` which polls a specified NIC and reads and timestamps packets when they arrive. Deferring discussion of the `Classify` and `StrideScheduler` elements for now, the next step is to delay packets. This function is implemented through the `Queue` and `Delay` elements. `Queue`, a simple FIFO, is required since the interarrival time of packets can be

[1] We developed NetPath using the v1.3pre1 distribution of Click.

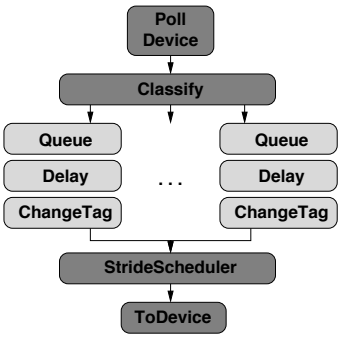



**Figure 1. Click elements used in basic and multiple link emulation. Lighter elements were modified and extended for NetPath.**

shorter than the specified delay interval. `Delay` is a single-packet buffer which, if empty, pulls packets from the head of `Queue`, computes the elapsed time since the packet's arrival, then holds it according to the desired delay setting. Our definition of link emulation includes the ability to cause random bit errors in packets. If enabled by the user, this function is included in the `Delay` element by toggling the first bit in the packet payload based on sampling a precomputed table of uniformly generated random numbers. The final step in basic link emulation is to place the packet on a network interface with the proper addressing information. The `ChangeTag` element takes a packet from `Delay`, encapsulates it with the appropriate addressing information, then sends it to the final element. `ToDevice` accepts packets from `ChangeTag` and is responsible for sending them to the NIC.

**2.2.2. Emulation of Multiple Links.** In addition to the elements related to basic link emulation, Figure 1 includes the `Classify` and `StrideScheduler` elements. The design requirements for flexible and efficient use of host systems implies the ability to assign different delays to packets based on their source and destination addresses and port information. `Classify` pulls packets from `PollDevice` and, based on addressing information, passes them to the appropriate queue. The `StrideScheduler` multiplexes packets from the different link emulation paths and sends them on to the `ToDevice` element. The Click distribution includes all of these basic elements except for `ChangeTag`, and `Delay` only required a minor modification to enable probabilistic bit errors.

The `ChangeTag` element insures that outgoing packets from the emulator are correctly forwarded. Our primary consideration in developing `ChangeTag` was for NetPath to be easily used in any one of two physical configurations: *direct interposition* between hosts (*i.e.*, through the use of two network interface cards on the NetPath host and no other hub, switch or router) or *virtual interposition* between hosts in both switched and routed environments. We developed the `ChangeTag` element to operate at the link layer of the protocol stack, and to facilitate packet routing through NetPath via Virtual Local Area Networks (VLANs), in the case of switched links, or via Multiprotocol Label Switching (MPLS), in the case of routed links. While configuration of the required VLANs and/or MPLS routes in the switches or routers can be done by hand, we developed VICT to facilitate this task.

**2.2.3. High Performance through Enhanced Buffers.** While the Click platform enabled us to bootstrap our development effort, the performance of the basic elements fell short of our design objectives. As we will see in § 3, when larger delays are specified, the basic `Queue` element quickly overflows and packets are lost. Eliminating bottlenecks (and by extension unwanted packet loss), while necessary for maintaining precise control over packets passing through NetPath, was one of the primary challenges of this project. In comparing NetPath to other emulators, we found that they are also vulnerable to unwanted loss. We use loss as one of our metrics for comparison in § 3.3.

Our initial step in addressing unwanted loss was to modify the basic `Queue` element. `Queue` is implemented as a queue of packet pointers and is restricted to a maximum of 32K packets. Our implementation consists of an array of circular queues in which each queue has a maximum size of 32K (due to kernel restrictions on memory allocation). With this implementation, users can specify the desired quantity and size of each queue up to the total amount of available RAM on the host. While this modification has an important positive impact on performance, the overall scalability of the system is still limited by available RAM.

For greater scalability we created a seamless buffering mechanism that couples `Queue` to storage allocated on disk. We created the `DiskQueue` element to allocate and manage auxiliary queues on disk. When a queue in RAM fills, `DiskQueue` enables subsequent packets to be written to an associated queue on disk. As space is recovered in the RAM queue, `DiskQueue` shifts packets from the auxiliary queue to the RAM queue. Our challenges in developing this element were in keeping track of the boundaries of packets written to disk and in efficiently interacting with the virtual memory system of the host. The tasks of writing to and reading from the auxiliary queues are managed by a single thread which uses configurable thresholds to determine the quantity of data to transfer between RAM and disk. When these tasks execute, other tasks may be preempted which, depending on the traffic conditions, can lead to packet loss. Thus, the thresholds must be selected based on the NetPath configuration, host capabilities and expected traffic load.

**2.2.4. Queue Management on Two Disks.** While the ensemble of `DiskQueue` and `Queue` enhances the scalability of NetPath, random accesses to disk can cause performance degradation. To address this problem, we enhanced `DiskQueue` to take advantage of a host system with two disk drives. Our approach is to begin writing on disk $A$ until a read request arrives. Subsequent write requests are then directed to disk $B$ until reads have exhausted all of the data that had been previously written to disk $A$. Reading then switches to disk $B$ and writing switches back to disk $A$. We call this *opportunistic disk queue management*. This simple algorithm reduces the amount of seek time for disk heads and we show in § 3.2 that it increases performance. The caveat for this algorithm is that it will not necessarily improve performance when multiple disk queues are in use.

**2.2.5. Other Path Emulation Features.** NetPath's configuration of standard Click elements plus our modified queuing capabilities implement the basic link emulation functions scalably and with good performance. However, further additions were required to meet our objectives for path emulation. Specifically, in addition to applying fixed constant delays to packets we wanted the ability to probabilistically delay, drop, duplicate and reorder packets. Dropping and duplication is a simple matter of making a binary decision based on sampling a precomputed table of uniformly distributed random numbers (distributions other than uniform could be trivially added to NetPath). Probabilistic delay values are assigned from a precomputed table of normally distributed random numbers (with user specified mean and standard deviation). A subtle problem in our implementation causes head-of-the-line blocking to occur in the queues for some packet arrival processes. This can skew the shape of the resulting delay distribution. As future work, we are investigating different methods for addressing this problem. Probabilistic reordering of packets also depends on the packet arrival process and requires us to create a customized combination of the `Queue` and `Delay` elements. This combined element only makes a reordering decision when multiple packets are enqueued. If a reordering decision is made (based on sampling the uniform random number table), then the second packet in the queue is serviced before the first. More complex reorderings are certainly possible but are beyond the scope of this work.

**2.2.6. VICT implementation.** NetPath can be used for path emulation on multiple links. To configure such setups in both routed and switched environments, we developed VICT. To implement NetPath's virtual interposition, we employ VLAN tagging in switched environments and MPLS tagging in routed environments. VLAN tags are 12-bit fields identifying virtual LAN membership information. Similarly, MPLS tags are 20-bit virtual circuit identifiers. VLAN and MPLS tags, along with MAC addresses, are the primary mechanism for directing packets between other lab systems and NetPath. Obviously, VICT and NetPath will not be able to be used on multiple links in environments that do not support VLANs or MPLS. However, both of these protocols are commonly available in network systems today so our tools should be widely applicable. Our VICT implementation is about 1,000 lines of Perl, and currently supports Cisco's IOS configuration language. As future work we intend to modify and expand VICT to support configuration languages of other popular network hardware vendors.

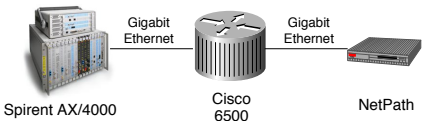
## 3. Performance Evaluation

Our objectives in evaluating NetPath were to assess its raw performance characteristics and to compare its performance with other similar tools. We verified the probabilistic bit errors, loss, duplication and reordering capabilities of NetPath for correctness but do not present results on their performance other than to say that the functions work as described in § 2. The focus of our evaluation efforts was on the throughput, scalability and precision properties of NetPath. As discussed in § 2, one of the primary limitations of path or network emulation systems is their ability to handle high traffic loads without dropping packets. Thus, instead of simply considering raw performance as a function of load, we consider *loss-free* performance as a function of load as one of our primary evaluation metrics.

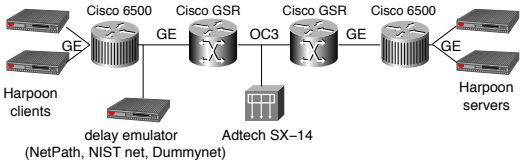
### 3.1. Experimental Testbeds

Our basic experimental testbed, shown in Figure 2(a) consisted of a commodity workstation, Cisco 6500 switch, and a Spirent AX/4000 traffic generator [2]. The workstation contained a 2 GHz Intel Pentium 4 with 1 GB main memory and a 32 bit, 33 MHz PCI Bus, two Intel/Pro 1000 Gigabit Ethernet adapters, and ran Linux (2.4.18 kernel)[2]. The host was also configured with two disk drives, both 30 GB IDE with DMA/ATA-133 interfaces. The Spirent AX/4000 is a high performance network test appliance with precise traffic stream generation and measurement capability. Our system was configured with OC-3 (149.76 Mb/s) and Gigabit Ethernet interfaces. Unless otherwise noted, the experiments below used the Gigabit Ethernet interfaces. All ports are synchronized to one clock. Thus, using the AX/4000 as both a traffic source and sink we were able to measure one way propagation delays with precision on the order of single microseconds. Unless otherwise noted, the traffic for each test was a constant rate stream of 66

[2]For the NICs, we used the Intel e1000 device driver and set `TxDescriptors` to 4096, the maximum possible value.




(a) In our basic setup, constant rate UDP streams produced by the AX/4000 were redirected to NetPath via a Cisco 6500 switch. After NetPath processing, packets returned to the AX/4000, where propagation delays and loss rates were measured.




(b) An extended setup was used for evaluating application traffic performance under different path emulation systems. We generated self-similar web-like TCP flows using the Harpoon traffic generator across a dumbbell-like topology.

**Figure 2. Experimental testbeds used to evaluate NetPath.**

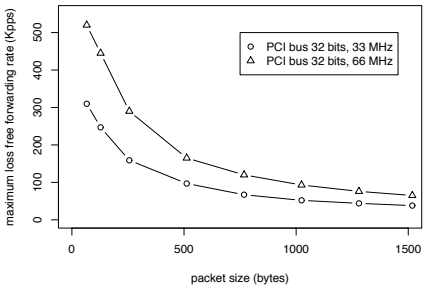



**Figure 3. NetPath's throughput as a function of packet size for two different host PCI bus configurations.**

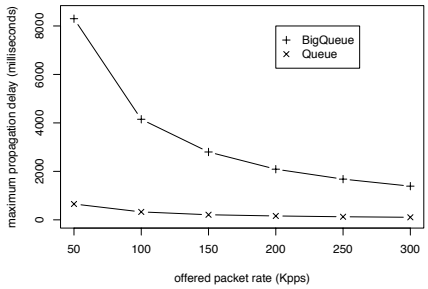



**Figure 4. Maximum propagation delays possible with the standard Click `Queue` element compared with the enhanced BigQueue element.**

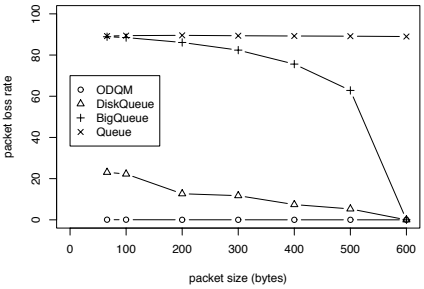



**Figure 5. Packet loss rates for four different queue implementations. Offered loads are fixed at 100 Mb/s with varying packet sizes. NetPath is configured to limit link capacity to 10 Mb/s.**

byte UDP packets. Our extended testbed, shown in Figure 2(b), was used for evaluating application traffic performance. It consisted of a dumbbell-like topology with a bottleneck OC-3 link between two Cisco GSRs, with Cisco 6500s used to aggregate end host traffic and divert packets to the path emulation host. The testing protocol for our experiments consisted of generating traffic streams and taking measurements over a period of 10 minutes.

### 3.2. NetPath Raw Performance Characteristic

We first consider the performance of NetPath by evaluating its throughput in terms of maximum loss-free packet forwarding rate (MLFFR). The MLFFR is the maximum rate at which no loss is measured in the received packet stream for the entire duration of the experiment. The focus of this experiment is on measuring the MLFFR of NetPath for packets of various lengths. As discussed in [17] throughput in Click can be limited by various hardware components such as CPU, PCI bus or the network adapter. To consider this issue, we evaluated throughput performance on two different workstations which differed only in their PCI bus configuration (both 32 bits wide, running at 33 or 66 MHz). As shown in Figure 3, NetPath can sustain a MLFFR of 310K minimum sized packets per second in the 33 MHz configuration and at least a 50% higher rate on the 66 MHz PCI system. This suggests that the PCI bus bandwidth plays an important role in limiting throughput[3].

Next, we evaluated our modifications to the `Queue` element. For these experiments, we tested four different instances of `Queue`: the base element provided in the standard Click distribution (which we refer to as `Queue`); the extension enabling all of RAM to be used (which we refer to as BigQueue); the extension enabling auxiliary queues on disk (which we refer to as DiskQueue); and the extension enabling auxiliary queues on two disks (which we refer to as opportunistic disk queue management—ODQM). We begin by showing the maximum possible propagation delay that can be realized on a link without packet loss for various input rates in Figure 4. We can see that BigQueue enables an order of magnitude longer delays than `Queue`. Since typical transcontinental propagation delays in the Internet are of the order of 10's of milliseconds, BigQueue is capable of providing this capacity on high throughput

[3]The fact that the CPU is always near 100% utilization due to polling makes it somewhat more difficult to do bottleneck analysis.

links. We will show later in § 3.3 that these propagation delays are also highly precise.

DiskQueue and ODQM were created to enhance NetPath's scalability for applications such as large propagation delay emulation and capacity emulation. We evaluated the capabilities of these systems by fixing the incoming traffic rate to 100 Mb/s, and configuring NetPath to emulate a capacity of 10 Mb/s. In this experiment, tests are run for 3 minutes—what might be considered to be an exceptionally long burst in the Internet. In Figure 5, we show the packet loss rates for packet streams with different size packets for all of the queue element implementations. We experimented with different packet sizes since Click allocates memory based on packets received—thus the effective allocation rate and the amount of space allocated goes down as packet size increases. The figure shows that the basic `Queue` is not useful for this application. When BigQueue is used, packets are lost because of overflow. DiskQueue enhances performance by about 300% and ODQM achieves 4 times better throughput than the DiskQueue. Packet loss still occurs in both DiskQueue and ODQM due to buffer overflows at the NIC. This is due to the overhead of the read/write tasks in these implementations which occasionally block the polling task. The implication of this result is that the auxiliary queues created by the `DiskQueue` element enable bursty traffic, common in the Internet [19], to be handled effectively.

As explained in § 2, our implementation of probabilistic delays follows a normal distribution based on user specified mean and standard deviation. We demonstrate NetPath's ability to create delays that follow the specified distributional characteristics in Figures 6(a) and 6(b). These figures show the cumulative distribution function of generated delays and compare them to the measured distribution. We considered two different configurations, each with mean delay of 30 milliseconds, one with standard deviation 3 milliseconds (*i.e.,* variable delays) and one with standard deviation 30 microseconds (*i.e.,* relatively stable delays). Each test was run with two different offered loads, 10 Kpps and 1 Kpps. From the figure, as variability increases, the mean of the measured delay is shifted by about 10 milliseconds for 10 Kpps and about 5 milliseconds for packet rate 1 Kpps. This effect is due to head-of-line blocking. Clearly, the distributional difference between the specified delay values and measured delay values depend on the packet inter-arrival times and the variability in delay. This result indicates that the current functionality for probabilistic delays is adequate for applications in which typical arrival times are longer than typical delay times.

One of the design goals for NetPath was that it support emulation of multiple point-to-point links at the same time. NetPath's design enables multiple `Queue-DelayChangeTag` sequences to be operational on a single host.

**Table 1. Comparison of delay precision of NetPath versus the Adtech SX-14 hardware-based emulator. Values indicate mean delay (standard deviation) delivered by each system, in millisec.**

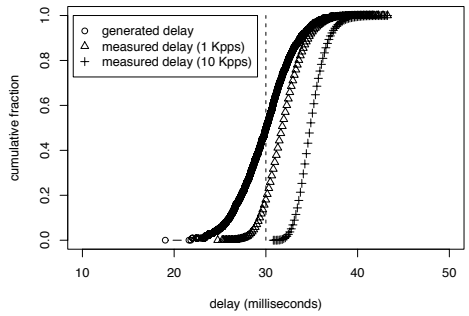
| **offered load (Kpps)** | NetPath $\mu$ ($\sigma$) | Adtech SX-14 $\mu$ ($\sigma$) |
|---|---|---|
| 50 | 10.03 (0.01) | 9.99 (0.00) |
| 100 | 10.03 (0.01) | 9.99 (0.00) |
| 150 | 10.04 (0.01) | 9.99 (0.00) |
| 200 | 10.05 (0.01) | 9.99 (0.00) |
| 250 | 10.06 (0.01) | 9.99 (0.00) |
| 300 | 10.09 (0.03) | 9.99 (0.00) |

Packets exiting each of these element sequences are subject to the `StrideScheduler` element prior to arrival at the `ToDevice` element. Thus, increasing the number of links not only increases classifying overhead but also the scheduling overhead which can affect both the cumulative forwarding rate and the precision of the system.

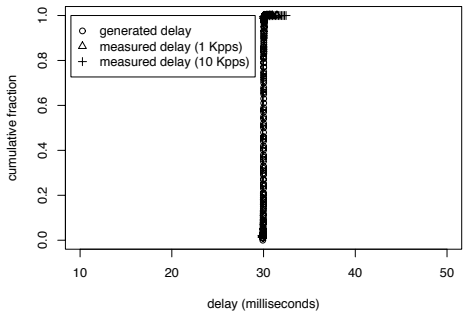
Figure 7(a) demonstrates these effects. In this experiment, we increased the offered packet rate evenly across all links until packet loss occurred on any link. Increasing the number of links from one to ten results in about a 7% reduction in the overall forwarding rate. This result indicates that NetPath can be effectively used for simultaneous path emulation on multiple links as long as the overall input rate for all paths is less than about 90% of the maximum throughput rate for one link. An important implication of a highly scalable system is in reduction of the number of hosts required for path emulation in a laboratory testbed. Figure 7(b) illustrates the number of systems required if one per link are used (as is the case in some network research labs today) compared with assigning multiple links per NetPath host for given offered loads. Clearly, substantial savings are possible.

### 3.3. Comparative Performance

Our first comparative performance test of NetPath considers its capability of generating high precision delays compared with a hardware-based emulator, an Adtech SX-14 [2]. Our SX-14 was configured with two OC-3 interfaces enabling it to be physically interposed between two nodes. We used OC-3 ports on AX-4000 for traffic generation and measurement of the SX-14. We configured the system for a fixed delay of 10 milliseconds, and, like our other experiments, measured the delays of packets passing through the system using different offered packet rates over a period of 10 minutes. We ran the same series of tests using NetPath (with the configuration shown in Figure 2(a)). Table 1 shows that NetPath's mean delay is within 1% of the SX-14 mean delay. The table also shows that NetPath's mean delay is relatively higher than the SX-14 due

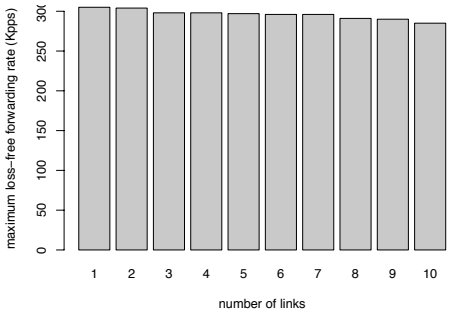



(a) Generated delay of mean 30 millisec. and standard deviation 3 millisec.

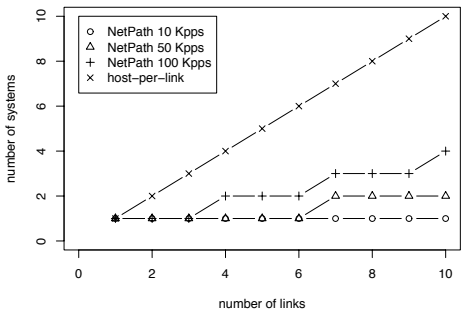



(b) Generated delay of mean 30 millisec. and standard deviation 30 microsec.

**Figure 6. Probabilistic delays produced by NetPath under two different offered loads for generated delays of mean 30 millisec. and two standard deviations of 3 millisec. (left) and 30 microsec. (right).**




(a) Maximum loss-free forwarding rate of one NetPath host with multiple links. Offered load is spread evenly across all links.




(b) Number of host systems required for NetPath under different maximum link loads. A one-host-per-link reference line is also shown as this is the approach used in some network testbeds today.

**Figure 7. Scalability of NetPath when multiple links are configured.**

to the inherent overhead of software-based systems. We also see from the table that there is very low variability in delays generated by NetPath. An implication of these results is that as input rates approach the maximum loss-free forwarding rate, NetPath provides precise delays—approaching those of dedicated systems like the SX-14.

Next, we compare the performance of NetPath with several widely-used software-based network emulation tools including NIST net, Dummynet, and Modelnet[4]. Our assessment of performance considers the throughput and precision of each tool through a series of four identical experiments. Each experiment measures the software-based emulator's performance under a variety of input rates using 66 byte UDP packets. We configured each tool to provide a modest 10 millisecond delay on a single link in each experiment.

[4]On the same host system as NetPath, we configured NIST net (verson 2.0.12), and on a separate partition installed FreeBSD version 4.8 for Dummynet and Modelnet (version 0.96). On the FreeBSD system, we set the system clock rate to 1 kHz and set NMBCLUSTER to 65K, according to Dummynet and Modelnet documentation.

Figure 8 shows the mean and standard deviation of measured delays for NetPath, NIST net, Dummynet, and Modelnet respectively. Table 2 contains selected values from these experiments. The results indicate that NIST net's delay precision is fairly accurate up to 80 Kpps but beyond this the deviation from the target delay begins to increase. Degradation at this threshold can be attributed to the interrupt driven implementation of NIST net. Similar to NIST net, Dummynet shows good performance up to 80 Kpps but beyond this, performance degrades dramatically. Modelnet's performance is the best of the software-based emulators up to about 140 Kpps, but beyond 150 Kpps, we were unable to get any response. A further point of comparison of delay precision is the variability of delays. Table 2 shows that standard software-based emulators will not provide precise delays on links under heavy load, and that NetPath is stable over the range of these conditions.

Delay distribution is a useful metric for comparing software-based emulators but is incomplete since there is no notion of loss-free forwarding. Figure 9 shows the packet loss characteristics of each tool as a function of of-

**Table 2. Delay precision for emulation systems over a range of offered loads (incoming packets per second). Values indicate mean delay (standard deviation) delivered by each system, in millisec.**

| offered load (Kpps) | NetPath $\mu$ ($\sigma$) | NIST net $\mu$ ($\sigma$) | Dummynet $\mu$ ($\sigma$) | Modelnet $\mu$ ($\sigma$) |
|---|---|---|---|---|
| 40 | 10.04 (0.01) | 10.14 (0.03) | 10.00 (0.35) | 10.06 (0.13) |
| 80 | 10.04 (0.01) | 10.52 (0.40) | 10.23 (0.46) | 10.10 (0.20) |
| 120 | 10.04 (0.01) | 12.75 (0.84) | 29.08 (1.68) | 10.56 (0.45) |
| 160 | 10.04 (0.01) | 16.47 (1.96) | 399.00 (3.50) | -failure- |

fered load. NetPath does not drop packets over the entire range. Dummynet and Modelnet begin to drop after 40 Kpps and NIST net after 80 Kpps[5]. Implications of these results are significant since inadvertent packet loss can have a great negative impact on many types of experiments.

As a final point of comparison, we looked at effects on application traffic by NetPath, NIST net, and Dummynet using the testbed shown in Figure 2(b). We configured each path emulation system to introduce an 80 millisecond round-trip time propagation delay and used the Harpoon traffic generator [22] to create self-similar web-like TCP traffic over a range of bit and packet rates, from approximately 60 Mb/s (20 Kpps) through 150 Mb/s (45 Kpps). At Harpoon clients, we measured response times (time between the initial SYN to the first data response packet from the server). We also captured all packet headers on the bottleneck link using an Endace DAG 3.5 card and measured packet drops at the bottleneck queue. In addition to the three software emulators, we ran experiments using the Adtech SX-14 to obtain a hardware-based reference point. All experiments were run for 10 minutes.

Figure 10 shows the distribution of response times measured by the Harpoon clients for the lowest offered load (60 Mb/s, 20 Kpps). Response times are shown relative to the SX-14 reference. We see that response times through NIST net are, on average, slower than the SX-14, and are quite variable. NIST net drops packets during this experiment, causing retransmissions and an overall shift in response times. For Dummynet, while the mean response times are closer to the reference point than NIST net, there is very high variance indicated by the tails in the distribution. Dummynet drops fewer packets than NIST net for this experiment, but with much more variable delays. NetPath delivers response times that are quite close to the hardware reference point, with relatively low variability. For higher data rates, NetPath delivers performance similar to that shown in Figure 10, while the other two systems degrade further.

It appears that the more realistic packet arrival process presented by Harpoon elicits somewhat different performance from each system than with constant-bit rate streams. While NetPath exhibits behavior nearly identical to the SX-14 under CBR traffic, it performs somewhat differently under this setup. Still, NetPath's performance is far superior to both NIST net and Dummynet. As future work, we intend to examine NetPath's behavior more closely to further improve performance.

[5]The bump in the NIST net curve is caused by interrupt coalescence.

## 4. Conclusions

Experiments conducted in laboratory-based network research testbed often require the capability to recreate characteristics of the wide area Internet paths accurately. In this paper, we described the design and implementation of NetPath, a new tool that provides scalable, precise path emulation capabilities. NetPath is based on the Click modular router platform and was developed by significantly extending the capabilities of the basic elements offered in the Click software. In particular, we enable all of main memory to be used by NetPath, and provide auxiliary queuing capability on disk. Our changes greatly improve the scalability of the system.

Our evaluation of the performance of NetPath considers throughput, ability to generate precise delays, and scalability. We show with that NetPath, precise path emulation for high throughput links is now possible using a software-based system. NetPath provides throughput and precision that are well beyond the capabilities of other software-based network emulators and approach those of an expensive hardware-based system. We also show that NetPath can be effectively multiplexed to provide path emulation across several links simultaneously.

## 5. Acknowledgements

The authors thank Vinod Yegneswaran, Remzi Arpaci-Dusseau and Muthian Sivathanu for helpful discussions. The authors also thank Spirent Communications and Cisco Systems for their support. This work was supported by grants from Intel Research, and NSF grants ANI-0335234 and ANI-0117810.

## References

[1] The DETER testbed. http://www.isi.edu/deter/, 2005.

[2] Spirent Communications. http://www.spirentcom.com, 2005.

[3] A. Adams, J. Mahdavi, M. Mathis, and V. Paxson. Creating a scalable architecture for Internet measurement. *IEEE Network*, 1998.

[4] M. Allman, A. Caldwell, and S. Ostermann. ONE: The Ohio Network Emulator. Technical Report TR-19972, Ohio University, 1997.

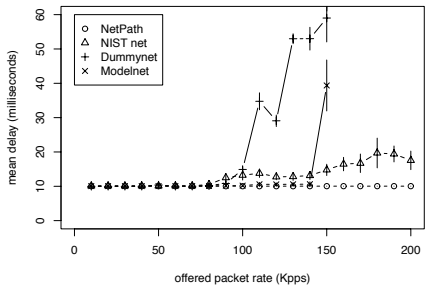



**Figure 8. Delay precision of NetPath, NIST net, Dummynet and Modelnet respectively for a configured delay of 10 millisec. Lines above and below each curve represent one standard deviation away from mean.**

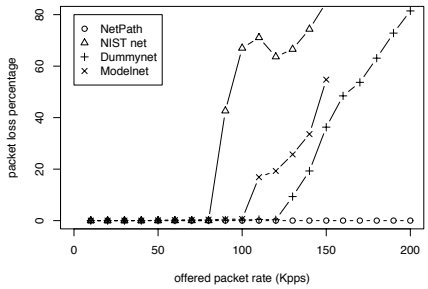



**Figure 9. Comparison of packet loss characteristics of NetPath versus other popular software-based emulators under different offered packet rates.**

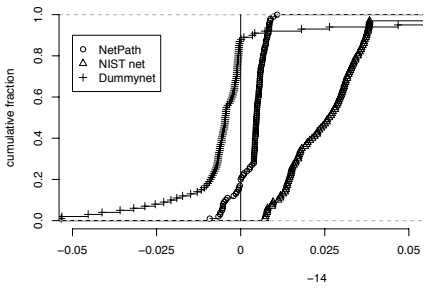



**Figure 10. Distribution of response times (time between initial SYN and first data response packet) for web-like TCP traffic using NetPath, NIST net, and Dummynet. Response times are all relative to the Adtech SX-14.**


[5] D. Anderson, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of the 18th ACM SOSP*, Banff, Canada, October 2001.

[6] G. Banga, J. Mogul, and P. Druschel. A Scalable and Explicit Event Delivery Mechanism for UNIX. In *Proceedings of USENIX Annula Technical Conference*, Monterey, CA, June 1999.

[7] P. Barford and L. Landweber. Bench–style Network Research in an Internet Instance Laboratory. *Computer Communications Review*, 30(5), October 2003.

[8] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating Systems Support for Planetary-Scale Network Services,. In *First Symposium on Network Systems Design and Implementation (NSDI'04)*, San Francisco, CA, March 2004.

[9] B. Braden, M. Gerla, J. Kurose, J. Lepreau, R. Rao, and J. Turner. Report of NSF Workshop on Network Research Testbeds. http://www-net.cs.umass.edu/testbed-workshop/, October 2002.

[10] R. Bradford, R. Simmonds, and B. Unger. A parallel discrete event IP network emulator. In *Proceedings of IEEE MASCOTS*, 2000.

[11] L. Brakmo and L. Peterson. Experiences with network simulation. In *Proceedings of ACM SIGMETRICS '96*, Philadelphia, PA, June 1996.

[12] M. Carson and D. Santay. NIST Net—A Linux-based Network Emulation Tool. *ACM Computer Communications Review*, 33(3), July 2003.

[13] Committee on Research Horizons in Networking. *Looking Over the Fence at Networks: A Neighbor's View of Networking Research*. National Academy Press, Washington, D.C., 2001.

[14] K. Fall. Network Emulation in the Vint/NS Simulator. In *Proceedings of the Fourth IEEE Symposium on Computers and Communication*, Red Sea, Egypt, July 1999.

[15] S. Floyd and E. Kohler. Internet research needs better models. In *Proceedings of HotnetsI*, Princeton, NJ, October 2002.

[16] S. Floyd and V. Paxson. Difficulties in Simulating the Internet. *ACM Transactions on Networking*, 9(4), August 2001.

[17] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3), August 2000.

[18] L. Le, J. Aikat, K. Jeffey, and F. Smith. The effects of active queue management on web performance. In *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, August 2003.

[19] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, pages 2:1–15, 1994.

[20] E. Nahum, M. Rosu, S. Seshan, and J. Almeida. The Effects of Wide-Area Conditions on WWW Server Performance. In *Proceedings of ACM SIGMETRICS '01*, Cambridge, MA, June 2001.

[21] L. Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. *ACM Computer Communications Review*, 27, January 1997.

[22] J. Sommers and P. Barford. Self-configuring network traffic generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference '04*, 2004.

[23] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.

[24] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, December 2002.