

IPV4/V6 NETWORK EMULATOR USING DIVERT SOCKET

A. Ihara, S. Murase, K. Goto *

Dept. of Information and Telecommunication Engineering, Nanzan University, 27 Seirei-cho, Seto, Aichi 489-0863, Japan, goto@it.nanzan-u.ac.jp

Keywords: network emulation; divert socket; performance evaluation; Internet.

Abstract

We designed and implemented a software network emulator, GINE, using IP divert socket mechanism. A realistic network topology which consists of many routers can be represented in a simple C++ main program with our API. Through performance test, it was found that a network topology with more than 50 routers can be emulated with very high throughput of 200 Mbps on an inexpensive PC. Also, emulated throughput, delay, and loss distribution were found to be very close to the given parameters. Although GUI and many other functionalities have not been implemented yet, our emulator is a useful tool for network professionals, education, and network application testing, since it is simple and easy to customize.

1 Introduction

In performance evaluation of wide area network applications, it is necessary to impose various network impediments on them. However, it is hard and not cost-effective to prepare a large-scale network test bed.

Simulators are often used for network testing and have many advantages over network test bed but real network applications such as a Web server and a browser cannot be used. Thus emulators have advantages over simulators when realistic tests with real application software and equipments are required. Also synthetic network traffic generated by external hosts can be easily injected to the emulated network.

There are several network emulators freely or commercially available. Commercial network emulator products such as [3, 10, 11] with dedicated hardware implementation are very fast (1 to 10Gbps) and expensive. Commercial software implementations such as [8, 14] are sold in a reasonable price but impossible for an end user to customize them since they are black boxes. As a result, they are not widely used in network research community.

There are several open source emulators. Dummynet[9] is a simple bandwidth limit and delay emulation included in FreeBSD kernel options. NIST Net[1] is an excellent network emulation tool implemented in the Linux kernel. It is very fast

with taking advantage of its loadable Linux kernel module for manipulating datalink frames, filter, and transmission scheduler, but lacks IPv6 capability and does not support network topology description.

IMUNES[13] is based on virtual IP network stacks with heavy kernel modification on FreeBSD and achieves high throughput of several hundreds Mbps. Unfortunately IPv6 has not implemented yet in it. NCTUns[12] is based on its own IP divert mechanism and process/thread scheduling with Linux kernel modification. It also includes many datalink layer emulation modules including wireless and many other functionality. However maximum routing speed seems to be quite less than 100Mbps. These emulators have graphical user interface (GUI) for network topology editing and execution control.

In this research, we design and implement a software network emulator (called GINE; Goto's IP Network Emulator) with minimum kernel modification for simplicity and portability among different versions of kernels. However, processing speed is still very important for emulating a network at least as fast as home use optical fiber links at 100Mbps. Our previous implementation[5] was based on multiple user processes which communicate with one another via Unix domain sockets. It was flexible and useful for prototyping but unfortunately very slow (10Mbps or less) compared with other approaches.

The emulator performance can be improved with multiple threads synchronized with a fine grain timer that can be frequently called without giving heavy load to the operating system. In our implementation, threads are synchronized periodically with Linux RTC timer of $\frac{1}{8192}$ sec resolution. The emulator can run either on Linux or FreeBSD, but FreeBSD version is very slow since the FreeBSD kernel used for the development does not have real RTC timer device driver but Linux RTC timer emulation. Frequent timer calls make load factor very high on FreeBSD.

2 System overview

In this section, the architecture of the GINE and the usage examples are described to show the potential purposes of the emulator.

2.1 System architecture

Fig. 1 illustrates the emulator modules (objects in C++) and packet flow. A network consists of multiple links and routers is represented as a single multi-threaded user process.

* This work was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas(2), 16016248, 2004.

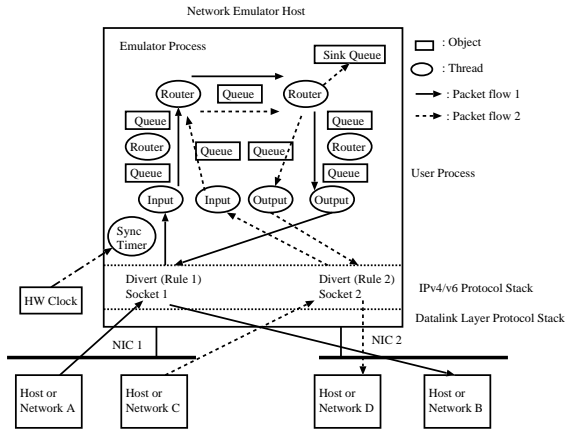


Figure 1: System overview

Fig. 1 represents one-way packet flow from external Host or Net A to B over 4 routers with cross traffic flow from external Host or Net C to D. The packets of other direction are forwarded by the OS router normally without any impediments. The ovals and rectangles in the figure denote threads and static objects, respectively. The solid arrows and dotted arrows represent main traffic flow and cross traffic flow, respectively. That is, traffic flow from A to B is interfered by the cross traffic from C to D. If any response is not required for the cross traffic application, Host D can be omitted and Sink Queue is used instead.

Note that cross traffic can be injected from the same physical network interfaces as long as the total traffic is less than the physical interface capacity.

Packet diversions are specified by the two command lines shown in Fig. 2.

```
# iptables -A FORWARD -s NetA -d NetB \
  -j DIVERT --div6-port 8000
# iptables -A FORWARD -s NetC -d NetD \
  -j DIVERT --div6-port 9000
```

Figure 2: Diversion set up with iptables command

The first rule instructs the kernel netfilter to divert the packets from NetA to B from FORWARD chain and output the packets to the socket buffer of a divert socket bound to port 8000. The second rule instructs the netfilter to divert the packets from Net C to D to another divert socket bound to port 9000. Note that if there is no socket bound to the ports, the packets matched the filter rules are simply discarded.

Fig. 3 shows a sample main program using GINE API. A one-way end-to-end path with shifted exponential random delay (constant 10 msec + mean 10 msec exponential) is emulated in the program. This example program was used in the delay emulation test in subsection 4.2

A user must write this kind of C++ code and compile it to

```
1 int main(int argc, char* argv[]){
2
3 // IP tables setup (omitted)
4 // Set scheduler
5 Process::setScheduler("fifo");
6 // Set up Timer
7 Conditional *sync = new Conditional();
8 RtcTimer *timer = new
RtcTimer(inittimer(),sync,1);
9
10 // Divert socket I/O
11 DivertIn* divertInFwd = new
DivertIn(4,8888,sync); // IPv4 port 8888
12 DivertOut* divertOutFwd
13 = new DivertOut(
14 divertInFwd->getSocket(),sync);
15 // use the same socket and sync
16
17 // Create line as a queue with delay,
loss, bufsize, and capacity(Mbps)
18 DelayGenerator *delayGen = new
DelayGenerator();
19 delayGen->setExponential(0.01, 0.01,
atof(argv[1]));
20 // base = 10msec, mean = 10msec,
corr= argv[1](0: independent)
21 LossGenerator *lossGen = new
LossGenerator();
22 lossGen->setRandomPacketLoss(0.0); //
No loss
23 PacketQueue* queue;
24 queue = new PacketQueue(
25 delayGen,lossGen,1000000,1000.0);
26 // connect to the queue
27 divertInFwd->setOutLine(queue);
divertOutFwd->setInLine(queue);
28
29 // Start all the threads
30 fprintf(stderr,"starting threads\n");
31 timer->start(); divertInFwd->start();
divertOutFwd->start();
32
33 timer->join(); // Wait until timer
thread stops
34 terminate(0); // clear iptables
35
36 return 0;
37 }
```

Figure 3: Sample program listing (end-to-end emulation with shifted exponential random delay)

run an emulation since GUI topology editor has not been implemented. The program consists of some routine parts and topology setup specific to an emulation model, that is;

1. iptables setup (can be done in command lines),
2. process schedule, sync. object, timer thread creation (line 5, 7, and 8),
3. divert input thread creation (line 12),
4. divert output thread creation (line 13–14),
5. delay generator object creation and setup (line 18–19),
6. loss generator object creation and setup (line 21–22),
7. packet queue object creation (line 24–25),
8. connect packet queue with divert input/output (line 27),
9. start all threads (line 31),
10. join timer thread to avoid termination of main thread (line 33).

Note that changes of network parameters or topology can be described between line 33 and 34 according to the emulation scenario. This kind of program is easy to write for network professionals with some programming experience.

2.2 Usage examples

Some usage examples are given here to demonstrate the usefulness of the proposed emulator.

Fig. 4 represents a one-way chain of n routers consists of $n + 2$ threads. This model can be used to emulate a long end-to-end path on the Internet. Also it is used for performance limit test in section 4. n might be 10 to 30 in most cases, and no more than 64 since default TTL (Time To Live) of most OS is 64 and no more initial TTL is not necessary for the Internet today.

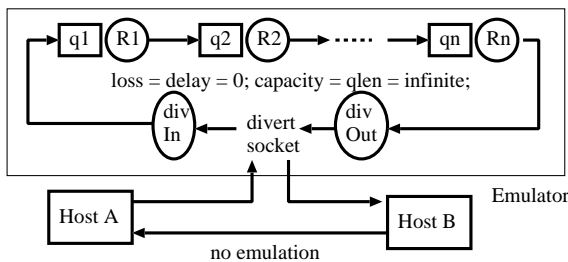


Figure 4: n Routers in series

Next, Fig. 5 is a two-way emulation of end-to-end Internet path or one datalink. Two divert sockets are used to avoid confusion among inbound and outbound traffic.

Fig. 6 is a two-way emulation of end-to-end Internet path over multiple routers with cross traffic. Since one way path parameters to be estimated are not known on the real network, it is hard

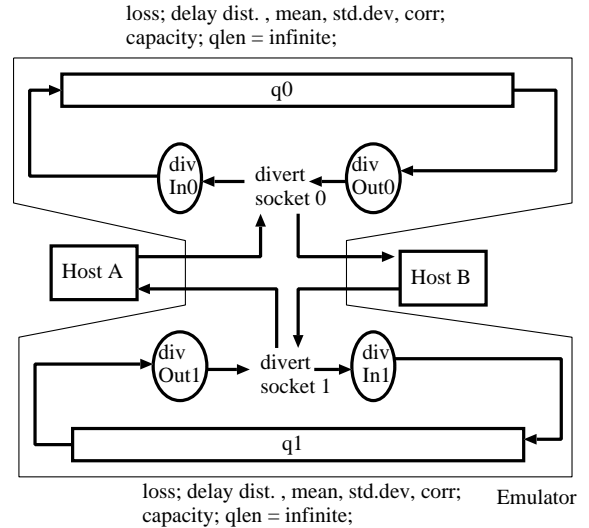


Figure 5: Line or end-to-end emulation

to validate an estimation tool for delay, bandwidth, or bottle neck capacity. This model is useful to test and validate such estimation tools with emulated network in which parameters are known. GINE has been used to validate measurement tool[4] developed by the author.

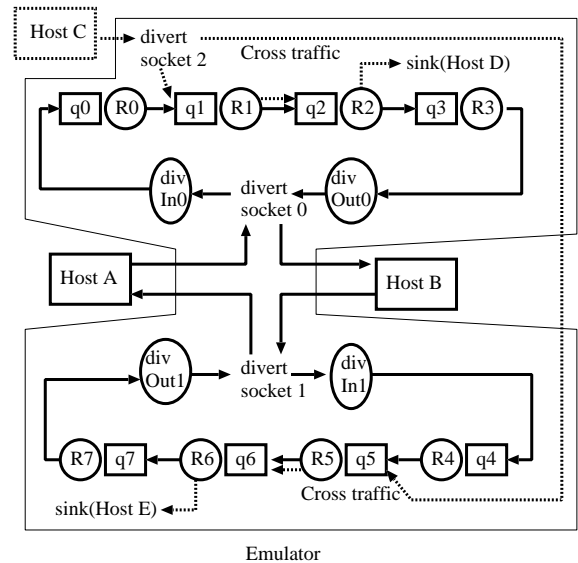


Figure 6: One-way bandwidth, delay, and bottleneck capacity estimation

Fig. 7 illustrates a multi-path IP datagram transmission over 2 Internet connection (BGP4 multi-homed but two paths cannot be used at the same time) to gain a double bandwidth in total. The dispatcher is a special router to evenly use the two transmission paths. It is not cost effective to lease Internet connections just for the experiment. Then the emulator is very useful to develop dispatch algorithms for non-homogeneous paths. Note that the dispatcher prototype program need to be written

using GINE API. The dispatcher is not included in the emulator.

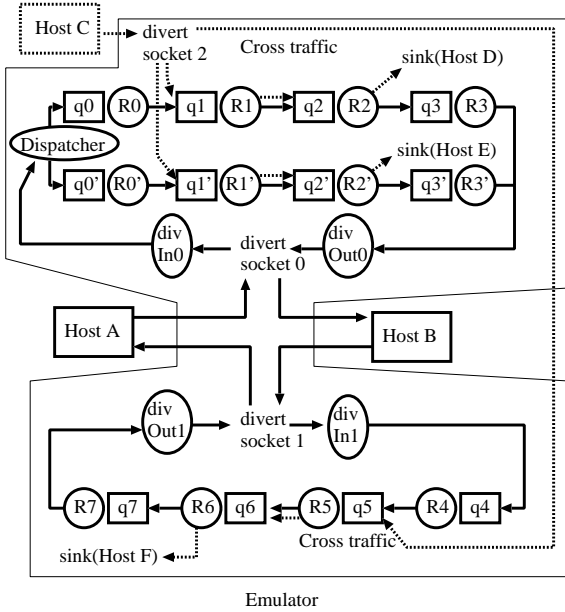


Figure 7: IP multi-path transmission

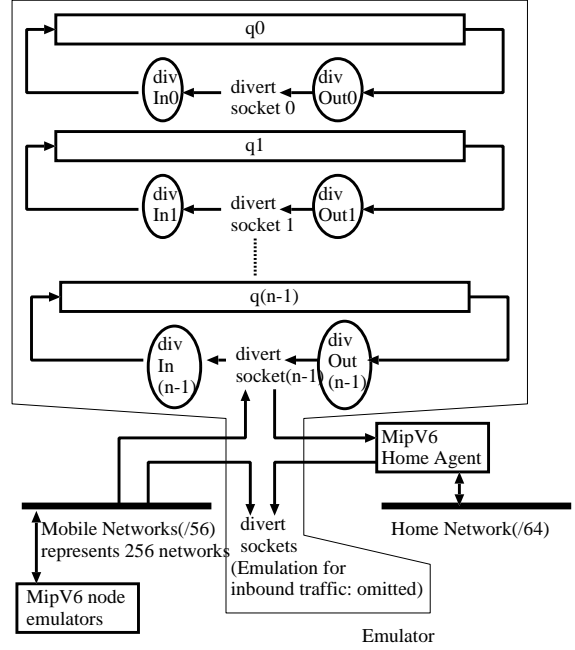


Figure 8: Mobile IPv6 network emulation

Fig. 8 shows Mobile IP network emulation. GINE can be used to represent impediments for many subnetworks where mobile nodes travel. By using different prefix length (subnet mask in case of IPv4), many subnetworks can be emulated with a single emulator host. Note that mobile node emulators are not part of GINE.

3 Implementation

GINE is a small software which is less than 1400 C++ lines of code (LOC). Kernel modification to support IPv6 divert socket and major implementation issues are discussed in this section.

3.1 Kernel modification

There are several ways to receive raw IP packets from the kernel. One way is an OS specific datalink layer diversion. It is suitable for bridge type network emulator. If datalink layer diversion is used for IP emulation, it is necessary to include fast IP filter function in the emulator since datalink layer does not have IP and transport layer filters.

IPv4 Divert socket was originally developed for FreeBSD kernel and then it was ported to Linux[6]. It diverts packets which match a firewall rule (iptables or ipfw rule) introduced in Fig. 2 to a special RAW socket, socket(PF_INET, SOCK_RAW, IPPROTO_DIVERT), or socket(PF_INET6, SOCK_RAW, IPPROTO_DIVERT).

The divert socket is bound to a port with bind() system call. Port for divert socket is similar to TCP/UDP port but does not conflict with the same number of TCP or UDP port. Therefore,

the number of possible emulation points is as many as $2^{16} = 65536$. It brings flexibility to the emulator.

If GINE will be used only for IPv4, no kernel modification is required for FreeBSD but adding configuration options. In case of Linux, one should get and apply patch to the kernel source from [6], and configure the patched kernel. There will be no trouble if the kernel version is exactly the same as the target version of the patch. Recently patch kit including iptables for kernel 2.6.x has been released. Note that Linux IPv4 divert socket code for kernel 2.4.x does not work well with Symmetric Multi Processor (SMP) kernel. SMP aware code is provided in the 2.6.x divert code. Therefore, socket buffer hash operation lock/unlock lines have been added to the IPv4 divert socket code in 2.4 kernel and its IPv6 port.

We have ported divert socket to IPv6 by adding about 1000 lines of C program code either on FreeBSD and Linux kernel for this research. Since current implementation of GINE only runs fast on Linux, the details are described only for Linux kernel modification. Table 1 summarizes kernel and utility command modifications.

Current IPv6 divert socket runs on Linux kernel 2.4.22 and 2.4.31. iptables and ip6tables commands extended to target DIVERT is based on iptables-1.2.7a. Also note that the emulator can be used between external networks/hosts and the localhost, if iptables/ip6tables are applied to INPUT or OUTPUT chain.

3.2 Delay, line capacity, and loss emulation

Fig. 9 denotes PacketQueue class, the core of the network impediment implementation. Queue may represent an in-

Table 1: Linux kernel modification for IPv6 Divert

file	LOC	note
net/ipv6/		
Config.in	5	config menu
Makefile	1	compile new files
ip6_divert.c	880	new file
addrconf.c	20	added
af_inet6.c	5	addition
netfilter/ip6_DIVERT.c	162	new file
netfilter/Makefile	1	compile new file
ip6tables		
extensions/libip6_DIVERT.c	130	new file

put/output buffer, or a point-to-point link.

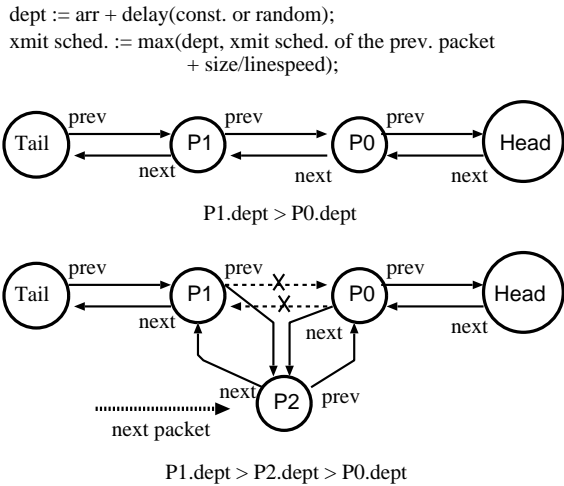


Figure 9: Packet queue in departure time order

Queue is a bidirectional linked list of Packet objects arranged in the order of scheduled departure time, since packet reordering may happen. Packet objects include raw IP packet, arrival time, departure time, and other information necessary or added for efficient routing and statistics. Note that packet reordering or duplication function has not been implemented in GINE yet, since we have not found appropriate mathematical models or measured statistics for those types of impediment. Currently, only natural packet reordering caused by random delay is supported in consecutive transmission of packets with very short interval.

When a packet arrives at the queue (enqueue), departure time is calculated by adding a constant or random delay to the arrival time. And the packet is inserted into the queue in departure time order. When the packet at the head leaves the queue (dequeue), transmission time schedule of the next packet is obtained as $\max(\text{departure time of the next packet, transmit time of the head packet} + \text{size of the head packet (in bits)/line speed (in bps)})$. This emulates delay and capacity of a transmission link or end-to-end path at the same time.

Packet loss is generated from a uniform random number upon arrival with a probability approximately proportional to the size of a packet (independent constant bit error rate) or with an independent constant packet loss probability. Also length of a queue is finite, then bulk packet loss caused by buffer overflow is emulated. Buffer overflow happens often if the queue length limit is small and traffic is high relative to the service rate (line capacity). Other non independent packet loss models such as replay of measured samples or on/off Markov chain can be easily added to the current implementation. The difficult part is to choose loss models based on theory or measured statistics.

3.3 Packet forwarding with threads

Fig. 10 illustrates thread synchronization and packet input/output time line. Threads are implemented with Linux POSIX thread and used via C++ library commoncpp2-1.3.22[2].

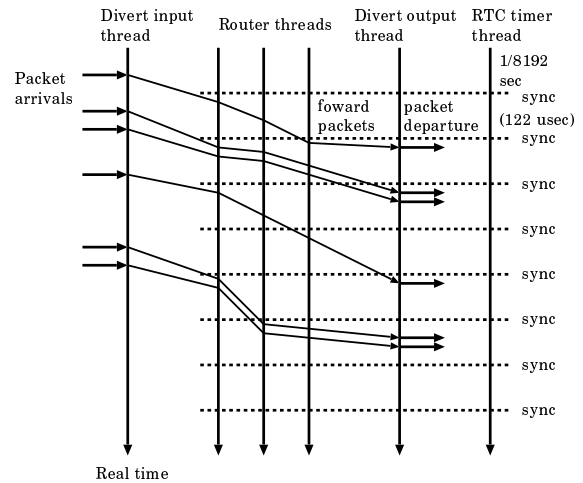


Figure 10: Thread synchronization and packet input/output time

Real Time Clock is a hardware clock on a PC and used only at system boot up. Linux RTC device driver enable a programmer to use it up to the resolution of $\frac{1}{8192}$ sec with root privilege. While frequent use of other timer imposes heavy load to the operating system, RTC timer is safe but there is a restriction that only one process can use the device.

In Fig. 10, arrived packets go through a Divert input, three Routers, and a Divert output. All the threads except Divert input are synchronized with the RTC timer thread every 122 micro sec (usec). Divert input thread continuously waits for an incoming packet and stores it in the output queue immediately. Routers are connected via queues in Fig. 9. If there are one or more packets behind the transmission schedule in the queue, all of them are forwarded by the Router or Divert output thread. That is, the minimum latency for a packet to traverse the emulator is 61 usec times. In reality, a packet may not be forwarded all the way in the same time slot. Sometimes it is processed in

the next slot. Therefore, the expectation of the time required for a packet from input to output is approximately $61 \times (\text{the number of routers} + 1)$ usec if we assume the probability that a packet is forwarded in the same time slot is 0.5. This latency is the major limiting factor of the proposed emulator's performance. However, 61 usec per router is not too bad. It takes 120 usec to transmit 1500 octet on 100Mbps line, and transmission time in the emulator is negligible since packet forwarding operation is just a pointer substitution.

4 Performance

The developed emulator, GINE, was tested on a cheap (less than 350 EUR) high performance server PC (Intel Pentium D 2.7GHz, 256MB, 1000Base-T NICs). NICs are Realtek RTL8169 or Broadcom BCM5751. Linux kernel for the emulator is 2.4.31 customized for IPv4/v6 divert socket with SMP (Symmetric Multi Processor) option enabled. Note that PCs connected to the both end of the emulator NICs are not as fast as the emulator host. Since CPU must be also fast to utilize full rate of 1000Base-T, the fact might affect the performance limit results in the followings.

Some basic loss models have been tested with GINE, but the results are not shown since loss emulation is the easiest part and it passed all the tests.

4.1 Routing performance limit

First of all, throughput limit and latency of the emulators are of interest. Figs. 11 and 12 show throughput and latency performance limit for one-way emulation over a chain of n routers with no impediment, respectively. That is, delay and loss are set to zero, queue length limit is very large (10^6 octets), and line capacity is 1000.0 Mbps. Note that 0 in the horizontal axis denotes a line or end-to-end path emulation without a router. The maximum n is 60 since default TTL (Time To Live) of most OS is 64, and the emulated router subtracts TTL by one as a real router does.

UDP throughputs were measured with Iperf[7] with default payload length of 1470, i.e. IP packet length is 1498 octets. Note that IP throughput is slightly higher than UDP throughput. For example, 95.7 Mbps in UDP throughput is equivalent to 97.5 Mbps in IP throughput. RTTs were measured with ping with default payload length of 64 octets.

Also, experiments for 1 CPU was conducted with another set of PCs, the emulator host has 1000Base-T NICs but the other 2 PCs have 100Base-TX NICs. As observed in Fig. 11, throughput higher than 100 Mbps might not be available for a several emulated routers even they have faster NICs.

Three kind of process/thread scheduling policies were tried. Linux kernel supports OTHER (regular, non-realtime), RR (realtime, Round Robin), and FIFO (realtime, first-in first-out). See 'man sched_setscheduler' for details. Apparently, FIFO gave the best results in expense of loosing execution time for other processes such as user terminal shell. Results for RR are

not included in the figures since it did not show significant improvement compared with OTHER.

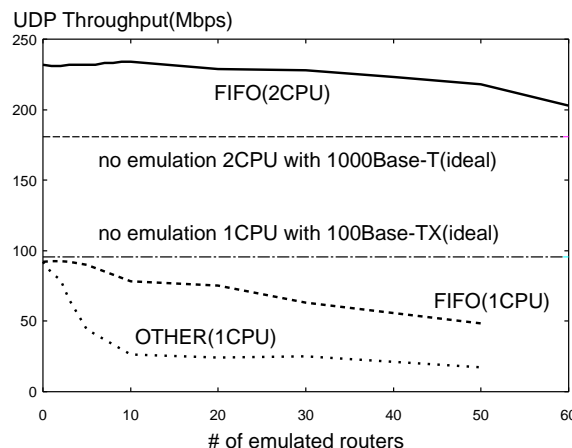


Figure 11: Throughput of one-way emulation with chain of n routers

Throughput for one CPU with OTHER was rather disappointing. One CPU with FIFO showed much better results. However, the control terminal almost froze for 4 routers with heavy traffic, and control terminal I/O was completely lost for 10 routers without any traffic. Therefore, one CPU implementation was not useful at all.

For two CPUs (dual core) case, the throughput was surprisingly high. It is higher than the native Linux router (no emulation). It seems to be because of large socket receive buffer space and emulated line buffers (queues).

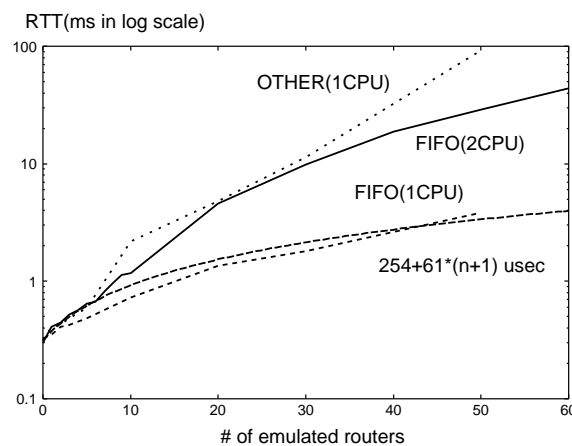


Figure 12: RTT of one-way emulation with chain of n routers

Theoretical expectation is also shown in Fig. 12 as a dotted curve as mentioned in the subsection 3.2. Note that the additional latency caused by the emulator is $(RTT - 0.254)$ ms, since RTT for native Linux router was 254 usec. Strangely, 1 CPU FIFO showed better results than 2 CPU FIFO for latency performance. This behavior implicates that FIFO scheduling does not work correctly as expected. While 20 ms latency is

usual within 20 hops on the real network, this behavior is somewhat disappointing and the proposed emulator requires further improvement.

4.2 Delay emulation test

Table 2 shows RTT for one-way constant delay line emulation measured with ping by 8 octets payload (IP length 36).

Table 2: One-way constant delay emulation (ping -i 0.1 -c 200 -s 8 target (IPlen 36))

Set	ping RTT(ms)				
	(ms)	min	ave	max	mdev
direct		0.187	0.204	0.266	0.011
0.000		0.207	0.278	0.355	0.042
0.005		0.258	0.329	0.425	0.039
0.1		0.307	0.378	0.462	0.043
0.5		0.712	0.778	0.854	0.041
1		1.204	1.276	1.362	0.039
5		5.221	5.283	5.385	0.054
10		10.203	10.280	10.352	0.093
50		50.198	50.274	50.343	0.154
100		100.179	100.250	100.386	0.313
500		500.125	500.206	501.866	0.283
1000		1000.050	1000.130	1000.209	1.096

As Table 2 shows constant delays were correctly emulated. Note that the effect of RTC clock resolution vanishes as the parameter set becomes large.

Fig. 13 illustrates the result of shifted exponential random delay emulation. Exponential random numbers are converted from uniform random numbers generated by standard random() library function. Currently, conversion for exponential random numbers is straightforward with mathematical inverse of the distribution function. Another method with distribution table lookup is implemented. Sometimes distribution table lookup is preferred since calculation is faster than executing complex mathematical function and arbitrary probability distribution or measured distribution can be easily represented.

In addition to independent delay, linear correlation between delays for successive packets is implemented as eq. 4.1 [1].

$$\text{delay} = c \cdot \text{delay}_{\text{previous}} + (1 - c) \cdot \text{delay}_{\text{random}} \quad (4.1)$$

where $-1.0 \leq c \leq 1.0$ (usually $c \geq 0$). Note that using $c = 0.8$ significantly reduces variance and slightly changes mean from those of the original distribution.

The dotted bold line in Fig. 13 denotes theoretical density function (constant 10ms + exponential mean 10ms). Emulated exponential distribution measured with ping in 10ms interval did not show good match with the theoretical line. Then experiment with 100ms interval ping was conducted. In the case of 100ms interval, it showed much better match. Exponential density is heavy tailed and not very realistic. The effect of correlation factor is clearly shown.

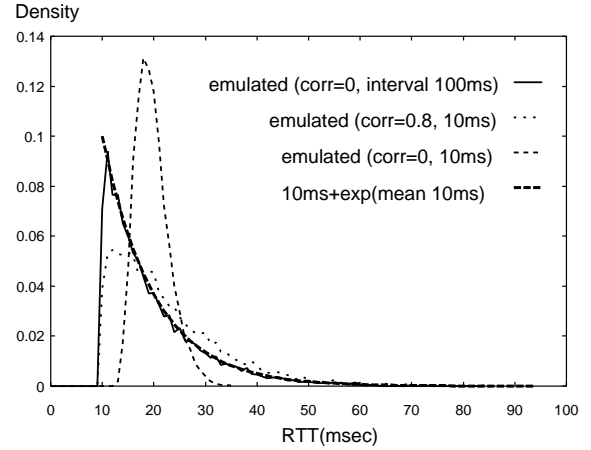


Figure 13: Density of emulated exponential delay

Successive packet transmission with random delay in short interval may cause reordering. In the samples used in Fig. 13, the numbers of reordering are 0, 1, and 1242 in 10000 packets for exponential with 100ms interval, correlated with 10ms interval, and exponential with 10ms, respectively.

4.3 Line capacity emulation test

Table 3 summarizes the results of line capacity emulation. Emulated capacity were measured as throughput without any cross traffic with Iperf again. 10 samples for each capacity were taken and min and max of them are listed in the table. Mean and standard deviation are omitted since the range between min and max is very narrow.

Table 3: Line capacity emulation (iperf -u -c target -b nM (10 times) or iperf -c target, 1470 byte + 28 = 1498)

Set	Observed			
	UDP 10 times		TCP 10 times	
(Mbps)	min	max	min	max
direct	154	157	176	181
0.1	98.1k	98.1k	96.4	96.5k
1	981k	981k	965k	965k
10	9.81	9.82	9.65	9.66
100	98.9	99.0	95.2	97.4
150	148	148	147	147
190	N/A	N/A	179	181

The results show that line capacities were successfully emulated in very wide range. Again given parameters are in IP transmission capacity and, for comparison, UDP throughput should be multiplied by $\frac{1498}{1470}$. For example, UDP 148 Mbps is equivalent to IP 150.8 Mbps. In the case of TCP, correction factor is unknown since packet(segment) size is dynamically controlled by the TCP itself. At least overhead of IP

header(20+IPOption) + TCP header(20+TCPOption) should be taken into consideration. Therefore, 181 Mbps in TCP worths at least $181 \times \frac{1500}{1460} = 186$ Mbps.

5 Conclusion

In this research, we designed and implemented a software network emulator, GINE, using IP divert socket mechanism. A realistic network topology consists of many routers can be represented in a simple C++ main program with our API. Through performance test, it was found that a network topology with a chain of more than 50 routers can be emulated at very high throughput of 200 Mbps on an inexpensive PC with a Pentium D dual core CPU running at 2.7GHz. Also, emulated throughput, delay, and/or loss distribution were found to be very close to the input parameters. Although GUI and many other functionalities have not been implemented yet, our emulator is a useful tool for network professionals, education, and network application testing, since it is simple and easy to understand for extension.

While the proposed emulator is already practical for basic network research or education, there are many issues to be solved. More correct and effective thread synchronization on SMP kernel should be developed to reduce latency. It can be solved either with a latest kernel which correctly support FIFO scheduling or developing an own event driven thread synchronization mechanism in the emulator. To speed up the execution, fast random number conversion, fast object creation/deletion are also important.

Also IPv6 specific functions, QoS aware queueing, congestion avoidance have not been implemented and they should be included in the near future. Currently only point-to-point link emulation is possible. Then various datalink network device, such as 100Base-TX, IEEE802.11 wireless LAN are to be included. For complex network topology, dynamic IPv4/v6 routing and multicasting should be supported.

For user functions, the emulator missing GUI, statistics report, animation playback, etc.

GINE is based on open source software and will be publicly available in the future. Currently, it is not ready for public release because there are some unfinished works. Interested readers may freely contact Dr. Goto for requesting the prototype.

References

- [1] Carson, M. and Santay, D., 2003. NIST Net – A Linux-based Network Emulation Tool, *ACM SIGCOMM Computer Communication Review*, **33**, No. 3, 111–126. (<http://www.itl.nist.gov/div892/itg/carson/nistnet/>).
- [2] Common C++ project, (accessed Jun. 2006). Common C++ Libraries. (<http://sourceforge.net/projects/cplusplus/>).
- [3] Emprix Inc., (accessed Jun. 2006). Hammer NetEm. (<http://www.emprix.com/>).
- [4] Goto, K., 2006. One-way Delay, Available Bandwidth, and Bottleneck Link Capacity Estimation on Asymmetric Internet Path. (in submission to IPSJ Journal in Japanese).
- [5] Ihara, A. and Murase, S., 2004. Design and Implementation of a Network Emulator, Graduates's thesis, Dept. of Information and Telecomm. Engineering, Nanzan University.
- [6] IPdivert project, (accessed Jun. 2006). Divert Sockets for Linux. (<http://sourceforge.net/projects/ipdivert/>).
- [7] NLANR Distributed Applications Support Team, (accessed Jun. 2006). Iperf The TCP/UDP Bandwidth Measurement Tool. (<http://dast.nlanr.net/Projects/Iperf/>).
- [8] Packetstorm Communications, Inc., (accessed Jun. 2006). Home Page(Product Guide). (<http://www.packetstorm.com/>).
- [9] Rizzo, L., (accessed Jun. 2006). IP_DummyNet. (http://info.i.et.unipi.it/~luigi/ip_dummyNet/).
- [10] Shunra Software Ltd., (accessed Jun. 2006). Virtual Enterprise Product Family. (<http://www.shunra.com/> a.k.a. Storm and Cloud).
- [11] Simena: Simena NE, (accessed Jun. 2006). (<http://www.simena.net/NetworkEmulator.htm>).
- [12] Wang, S. and Chou, C., 2006. Innovative Network Emulations Using The NCTUns Tool, *Computer Networking and Networks* (Shannon, S., ed.), Nova Science Publishers, chapter 7, 159–189. (<http://nsl10.csie.nctu.edu.tw/>).
- [13] Zec, M. and Mikuc, M., 2004. Operating System Support for Integrated Network Emulation in IMUNES, *Proc. of the 1st Workshop on Operating Systems and Architectural Support for the on demand IT InfraStructure / ASPLOS-XI*, Boston, ACM. (<http://www.tel.fer.hr/imunes/>).
- [14] ZTI Computing & Telecom, (accessed Jun. 2006). NetDisturb. (<http://www.zti-telecom.com/pages/main-netdisturb.htm>).