Network Emulator with Virtual Host and Packet Diversion

Kunio Goto

***Abstract*—In this research, we have re-designed the network emulator, GINE (Goto's IP Network Emulator), which is a user space program on Linux. It runs on standard Linux kernel with some options enabled. It is aimed to be used for performance evaluation of network application over wide-area network, development of new network service and education.**

GINE is based on custom queues which represent telecommunication lines with delay, loss, and bandwidth. The queues are driven by a short periodic timer(up to 20 micro second). Routers and switches are either emulated by the program components, or by virtual network host and kernel bridge. They are connected one another in the program via custom frame queues. Real frames(packets) input from datalink socket or Linux netfilter NFQUEUE socket are forwarded and output to the real network.

The new version has become stable. Also it includes graphical user interface, and, therefore, it can be used without programming. The emulator can be used as end-to-end path emulation and/or a testbed network consists of 20 to 30 hosts and routers including application servers on a PC. Further, multiple instances of the emulator on a PC and separate PCs can be connected to one another to construct a combined larger emulated network.

***Index Terms*—network, emulation, virtualization**

I. Introduction

In performance evaluation of wide area network applications, it is necessary to impose various network impediments on them. While a network simulator is used to analyze the behavior of the given network model, a network emulator actually stores and forwards packets incoming from real networks, and therefore should run in real time.

There are several commercial network emulators based on expensive hardware([1], [2], [3]). Also non-expensive commercial software emulators such as [4], [5] are available, but customization is difficult. As a result, they are not very widely used in network research community.

There are several open source emulators[6]. Dummynet[7] is a simple bandwidth limit and delay emulation included in FreeBSD kernel options. NIST Net[8] is an excellent network emulation tool implemented in the Linux kernel. It is very fast by taking advantage of its loadable Linux kernel module but lacks IPv6 capability and does not support network topology description. IMUNES[9] is based on virtual IP network stacks with heavy kernel customization on FreeBSD and achieves high throughput of several hundreds Mbps.

NCTUns[10] simulator/emulator is based on its own IP divert mechanism and process/thread scheduling using its custom Linux kernel. It also includes many datalink layer emulation modules including wireless. However, as an emulator, its routing/forwarding performance does not seems to be very high (less than 100Mbps).

We have been developing a software called GINE (Goto's IP Network Emulator)[11] since 2004. It used IPv4 divert socket patch[12] and IPv6 patch developed by the authors.

In our previous research[13] in 2008, the emulator has been improved in two points: elimination of kernel customization and real router/host functionality. To achieve the first goal, packet diversion method was changed from Linux divert socket to the standard kernel feature of Netfilter NFQUEUE[14] (kernel version 2.6.14 or later). For the second objective, Network Namespace[15] is utilized. Network Namespace is a virtual kernel network stack implementation as a part of container based host virtualization. It appeared in Linux kernel 2.6.26 (July, 2008) and then became stable in 2.6.30 (July, 2009).

One of similar software network emulators using Network Namespace is the Coreemu([16], [17], [18]). While link emulations in coremu are implemented with Netem[19] in the kernel. In GINE, link emulations are implemented as a user program for flexibility. While the time resolution is limited to 1 msec in netem, it is 100 micro sec(usec) or less in GINE. Also GINE includes simple emulated router and LAN switches in addition to virtual network stack host/routers. Combining those components, the emulator becomes a more powerful network performance evaluation tool for network professional and educators. In this research, our system is re-designed for stability and graphical user interface (GUI) with archive function.

In the next section, capability of GINE is briefly explained. In section III, the architecture of the emulator is described. In section IV, the software implementation issues are discussed. In section V, the emulator is evaluated in terms of frame forwarding performance and link emulation accuracy. In sectionVI, application of the GINE libraries for new network applications and event driven simulation are introduced. In the last section, concluding remarks are given.

II. Emulator Capability

In this section, capability of GINE is briefly explained.

A. Flexible Link Emulation for IPv4/v6

The first advantage of the emulator over Nist Net is IPv6 support and filter by IP address and prefix length, protocol, TCP/UDP port, and ICMP type/code. An example is shown in Fig.1. As shown in Fig.1, our emulator is able to impose different delay, loss, and bandwidth limit to the packets from Host A to B and those from Net C to D. Also, the link

Manuscript submitted March 10, 2012.

K. Goto is with the Department of Systems Design and Engineering, Nanzan University, 28 Seirei-cho, Seto, Aichi 489-0863, Japan (e-mail: goto@nanzan-u.ac.jp)

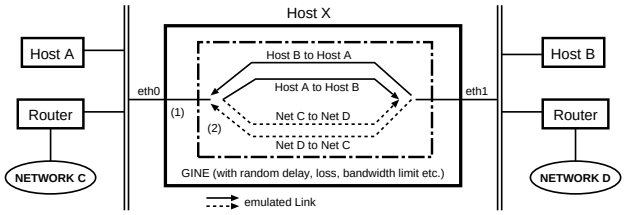

Fig. 1. Case 1: Link emulation with different parameters

parameters for the other direction can be defined differently. Note that packets are normally forwarded by the kernel routing function. Then packets should be diverted to the emulator program with **iptables** or **ip6tables** command.

B. Combination of Virtual Host, Emulated Router, and Emulated Switch

The second merit is to represent a rather complex network consists of many routers and links. Fig.2 illustrates an example. A network similar to the example is suitable for performance evaluation of network application with cross traffic and static or dynamic routing practice. In Fig.2, two

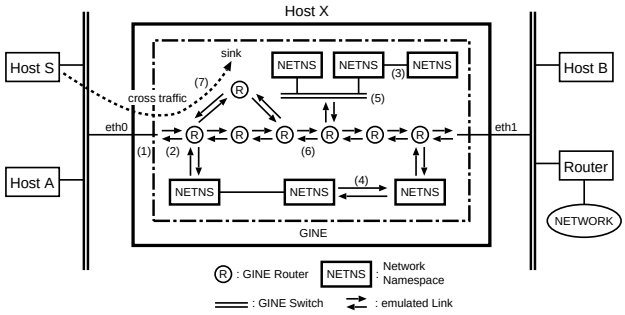

Fig. 2. Link and router emulation with cross traffic

kinds of routers are used. An '**R**' denotes a router represented by a program module. It supports ARP, ICMP ECHO for IPv4 and part of ICMPv6 to represents neighbor discovery and ECHO. However, R is not capable of dynamic routing. An '**NETNS**' denotes a virtual network space. It acts as a virtual network host or router with dynamic routing. Also emulated switch is provided as shown in the figure. Kernel bridge may be used for the switch instead. Application servers such as a Web server and routing daemon can be invoked on NETNS hosts.

In the figure, The packets from **Host A** to **Host B**, and vice versa are separately diverted to the emulator processes and go through the 4 emulated routers or NETNS.

Cross traffic can be injected at any emulated router from external hosts. In the figure, cross traffic is generated by **Host S** and injected at the first router from left and exits at the second router. Sink node simply absorbs input packets. Also GINE includes packet generator. Similar network is used for evaluating multi-path transmission scheme in [20].

C. Connection of Emulator Instances

If the processing power of a PC is not enough or topology becomes too complex for a large network emulation, then the whole network can be divided into two or more parts and each part can be run on different PCs (or on the same PC, if desirable).

Fig. 3 shows an example. In the figure, two emulator instances are shown. Each of them consists of a virtual host and a switch. The switches in different emulators are connected via UDP tunnel, and, therefore, the two emulator instances are connected each other. Note that multiple instances of the emulator can be invoked on the same CPU, if the CPU load permits.

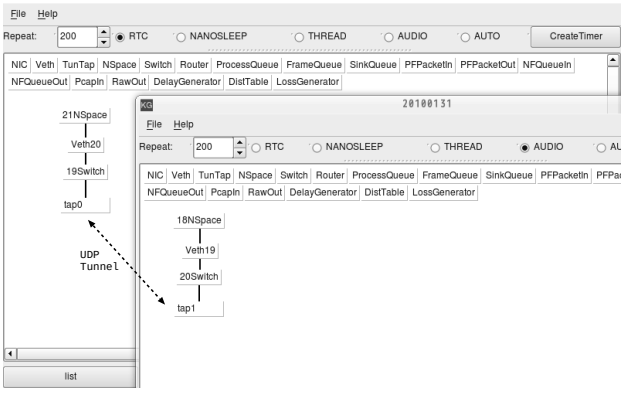

Fig. 3. Connection of two emulator instances

III. ARCHITECTURE

In this section, the GINE software architecture is described. While kernel implementation is the most efficient, user space program implementation is more flexible. Therefore, we have developed the emulator as a user space program.

A. Link Emulation with Custom Queue and Periodic Timer

A **FrameQueue** is the component for link delay, loss, and bandwidth limit emulation. Also it can be used as a generic frame/data buffer. The original version was in [11] and includes only frame data. The current version includes processing of out-of-band data for Netfilter NFQUEUE.

FrameQueue is a bidirectional linked list of Frame objects arranged in the order of scheduled departure time as shown in Fig.4. When a frame arrives at the queue, departure time is calculated by adding a constant or random delay to the arrival time. And the frame is inserted into the queue in departure time order. When the frame at the head leaves the queue, transmission time schedule of the next packet is calculated by eq.(1) according to the transmission time of the frame at the head of the queue.

$$\begin{aligned} &\text{next\_frame\_departure\_time(s)} \\ &= \max(\text{next\_frame\_scheduled\_departure\_time(s)}, \\ &\qquad \text{head\_frame\_departure\_time(s)} \\ &\qquad + \frac{\text{size\_of\_the\_head\_frame(bit)}}{\text{bandwidth(bps)}}) \end{aligned} \quad (1)$$

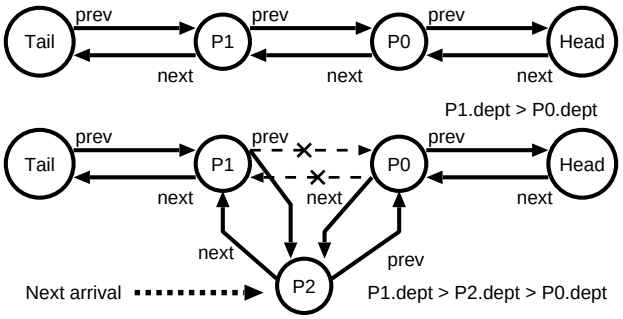

Fig. 4. Frame queue in departure time order

This emulates delay and capacity of a transmission link at the same time. Delay is generated as a pseudo random number according to a given probability distribution, such as uniform, exponential, normal, Pareto, or arbitrary distribution given in a table, with correlation between subsequent frames. Frame(packet) loss is generated similarly according to independent constant bit error rate or frame loss probability. Also frame loss generator with simple 2-state Markov chain is provided.

The custom frame queues are driven by a very short periodic timer. In other words, each frame queue is checked when the timer expires to see if there is a head frame scheduled to depart by the time. The details are described in section IV-B.

B. Communication between Virtual Host/Router

Network Namespace is a part of Linux kernel virtual host function. We use only network virtualization and do not use file system nor process virtualization since it is rather inconvenient for network emulation purpose. Network Namespace creates a different network stack as shown in Fig.5. To use the network namespace, recompilation of generic kernel

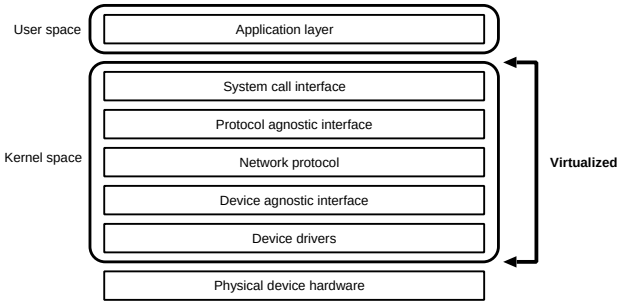

Fig. 5. Network Namespace

(recommended 2.6.32 or later) with 'CONFIG_NET_NS=y', is necessary if it is not included.

Then, the system call;

```
syscall(__NR_unshare,CLONE_NEWNET|CLONE_NEWNS);
```

binds the calling process and its child processes with the newly created network namespace.

Since network interfaces including loopback interface(lo) and Unix domain sockets are not shared among virtual network stacks and the host OS's original network stack, a special virtual network device, called **Virtual Ethernet Pair(veth device)**, must be used. When a frame is received by one side of a veth, it is forwarded to the other side. Therefore, communication between independent network stacks becomes possible by attaching each side of the veth device to two different virtual network stacks (or leaving one in host OS). Fig.6 shows the way of communication between them in the proposed network emulator. Communication between virtual

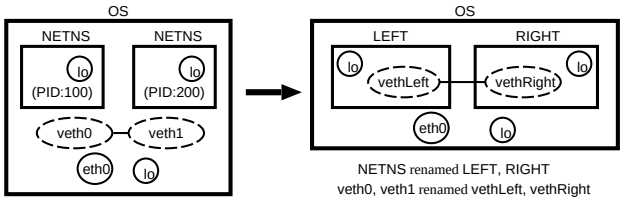

Fig. 6. Communication between virtual hosts

host and emulated router is a little troublesome, but possible with emulated switch, datalink socket, and frame queues. Fig.7 illustrates an example. Note that IP address is not assigned to

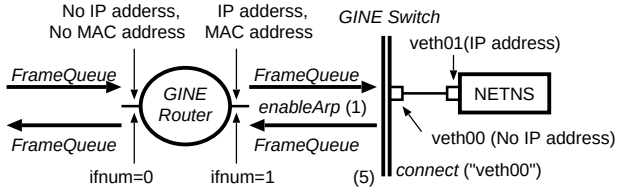

Fig. 7. Communication between virtual host and emulated router

the switch side of a virtual Ethernet pair.

IP addresses can be assigned to both side if necessary, but it is not desirable in the case. If IP address is assigned to the switch side of the virtual Ethernet pair, it becomes visible from the host OS. Then the packet is forwarded by the kernel, i.e. bypasses the emulator.

C. Graphical User Interface

GINE was originally designed for students familiar with IP networking and object oriented programming. However, it is a rather difficult task even for them to write a main program, which represents connection of network components, especially without drawing a network figure. Also, there will be greater merit to make the emulator used without programming. Therefore, graphical user interface (GUI) is designed as an add-on for GINE. Fig.8 illustrates the interface between GINE and its GUI. As shown in the figure, the interface is concentrated around the class **GineBaseObject** to make the relationship between the core library classes and GUI classes simple. Object archiver is also included in GineBaseObject.

IV. Implementation

In this section, software implementation issues are discussed. GINE is written in C++ with GNU CommonCpp[21] library classes (6000 LOC; Line of Codes). GUI is also written in C++ with Qt4[22] library classes (800 LOC).

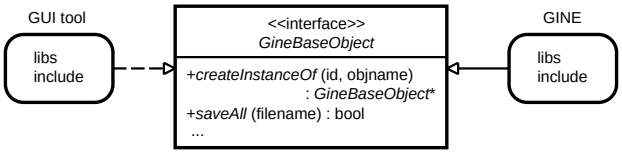

Fig. 8. Interface between GINE and GUI

A. Essential Library Classes

Fig.9 shows the GINE essential library classes. Dotted

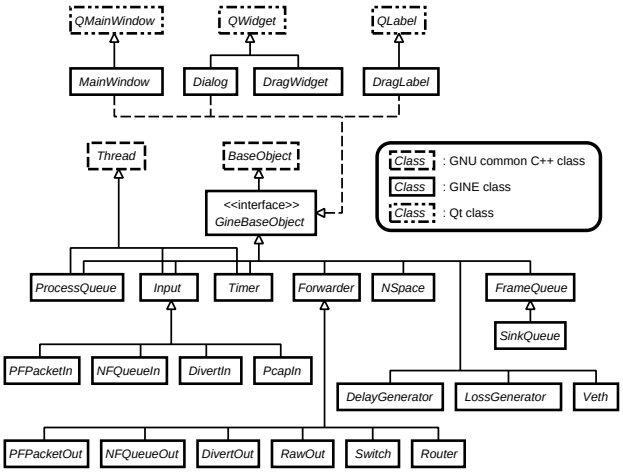

Fig. 9. Essential library classes

rectangles are GNU CommonCpp or Qt classes, and Solid rectangles are GINE classes. Upper part includes GINE GUI classes and lower part includes GINE core classes.

An **Input** class object receives packets from a socket and stores them into a **FrameQueue** object. It runs autonomously as a thread with read wait at a socket. **NFQueueIn** and **PFPacketIn** are the classes for input from NFQUEUE and datalink layer socket, respectively. **PcapIn** (libpcap) and **DivertIn** (works only with custom kernel) are alternatives.

A **Forwarder** class object is not a thread. Instead, a **ProcessQueue** thread object periodically checks for all the registered FrameQueue objects and calls the packet forwarding method in the Forwarder class. Then the method moves the frame at the head of the FrameQueue to a network interface or next FrameQueue object. Multiple FrameQueue objects may be registered in a ProcessQueue object to avoid creating too many threads in the emulator.

Fig.10 shows object dependency. Frame represents data which includes frame header and payload and used by all objects.

B. Periodic Timer Implementation

Three types of periodic timers using RTC, AUDIO, and NANOSLEEP are implemented, for the case that more than one periodic timers are required on a CPU.

With older kernels, resolution of Linux Real Time Clock(RTC) was limited to 1/8192 sec. With the recent kernel, our experiments showed the resolution seems to be up to

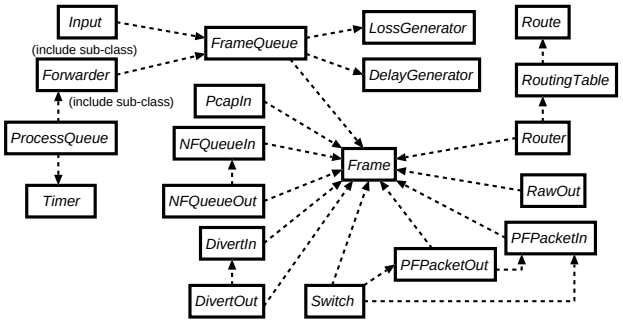

Fig. 10. Object relations

1/32768 sec (30 usec) in RTC emulation mode. Although RTC is stable, it is used only in a single process because it is based on the clock hardware.

Similarly, AUDIO timer is based on the read time for an audio sample from the audio device and used by a single process. Theoretically, its resolution is the sampling rate of the audio device, typically, 44.1kHz or 48kHz(20 usec).

nanosleep() with Linux High Resolution Timer provides short sleep about 100 micro sec (depends on CPU). The merit of nanosleep() based periodic timer is that nanosleep() can be used in multiple processes at the same time.

C. Delay Distribution

Exponential random numbers are converted from uniform random numbers generated by re-entrant version of drand48() library function. Lookup table (size 65536) is initialized according to the arbitrary probability distribution given. Table lookup is preferred since lookup is faster than executing complex mathematical function, typically, inverse of the probability distribution function. Also measured distribution can be used to initialize the lookup table.

In addition to independent delay, linear correlation between delays for successive packets is implemented as eq.(2) [8].

$$\text{delay} = c \cdot \text{delay}_{\text{previous}} + (1 - c) \cdot \text{delay}_{\text{random}} \qquad (2)$$

where $-1.0 \leq c \leq 1.0$ (usually $c \geq 0$).

For example, $c = 0.8$ significantly reduces variance and slightly changes mean from those of the original distribution.

D. NETNS Command Execution and Terminal Control

While command execution in a created network namespace (NETNS) is not easy without a terminal, opening terminals for all NETNS is not desirable for many NETNS (child) process executions. It consumes a lot of memory and makes confusion.

Then opening/closing selected terminals is a better approach. Note that terminal invoked in the child process cannot use main window(Xserver) because TCP, UDP, and Unix domain communication are independent among different network namespaces. Fig.11 illustrates the communication between the emulator parent process and its child NETNS process using pseudo tty and xterm connecting specified pty.

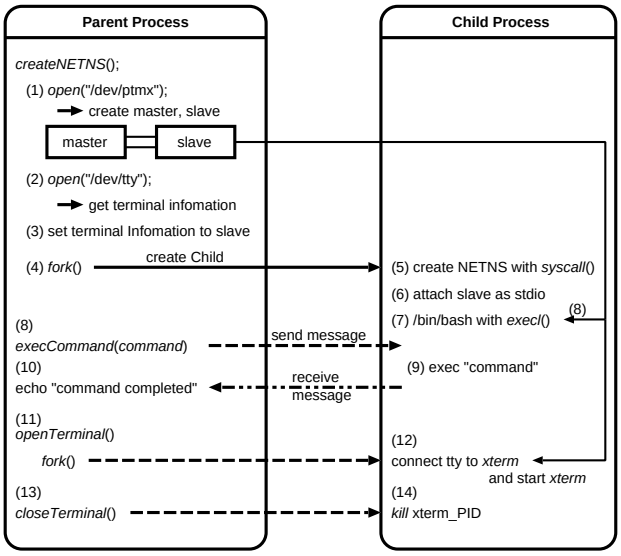

Fig. 11. NETNS process and terminal, command execution

E. Object Archiver

Object archiver is mandatory for GUI. A user writes a new network configuration, then wants to saves and edits it as in a word processor. Fig.12 shows the scheme. Fortunately,

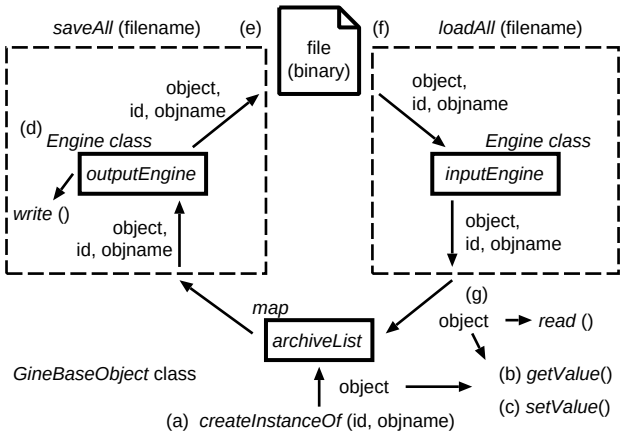

Fig. 12. Object archiver

CommonCpp includes **BaseObject** and **Engine** class which consistently provide persistent object archiver to the child class of BaseObject. Actual read/write methods should be implemented in each child class. By adding GineBaseObject (BaseObject subclass), object archiver is easily used from GUI or C++ main program.

F. Graphical User Interface

Fig.13 shows an example of GUI window. GINE object templates are represented by upper rectangle buttons with class name. A template is dragged and corresponding GINE object with automatic numbering is created when dropped. If the created object button is double-clicked, property window will appear and parameters can be modified. Lines between connected objects are automatically drawn as in Fig. 17. If

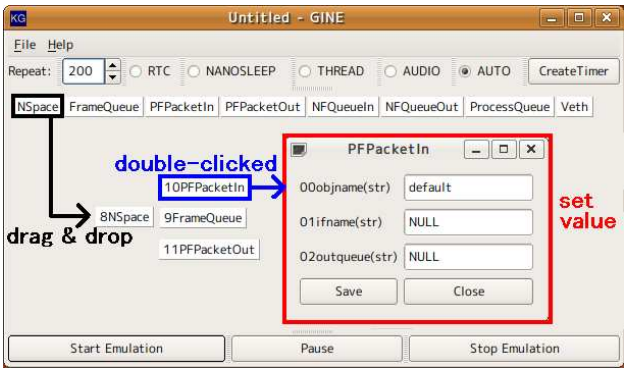

Fig. 13. Graphical User Interface

NSpace button is right-clicked, a terminal window (**xterm**) of the NETNS will open. Choice of timer, control of execution, file menus are also available.

V. Evaluation

In this section, performance of the emulator measured in the experiments is described. Table I shows the PCs used for the experiments. **PC1** and **PC2** have 4 64bit CPU cores and 2 64bit cores, respectively.

TABLE I
PC SPEC

PC1	DELL PowerEdge 840
CPU	Intel Xeon X3220 2.40GHz x86 Core™ 2 Quad (core 4)
Memory	2GB DDR2 PC2-5300E 667MHz ECC, Swap 4GB
OS	Ubuntu 9.04 (Jaunty Jackalope) 64bit OS
Kernel	Linux kernel 2.6.31 (NETNS enabled)
PC2	Panasonic Let's note CF-W7
CPU	Intel Core™ 2 Duo U7600 1.20GHz
Memory	2GB DDR2 PC2-5300 667MHz, Swap 4GB
OS	Ubuntu 9.04 (Jaunty Jackalope) 64bit OS
Kernel	Linux kernel 2.6.31 (NETNS enabled)

A. Number of NETNS Instances

Table II shows how many network stacks can be invoked on a PC.

TABLE II
MAXIMUM NUMBER OF NETWORK NAMESPACES(NETNS)

	PC1		PC2	
open files limit	32bit OS	64bit OS	32bit OS	64bit OS
1024 (default)	508	508	508	508
1048576 (max)	508	2858	508	4726

As Table II shows, the first limiting factor is the maximum number of open files (default 1024) limit the number of network stacks. The limit with default max open files is about 512 because 2 pseudo terminal devices are used per NETNS. Several hundreds might be a reasonable number of network stacks since memory swapping occurs at 1200 and also PCs slow down with many number of processes. The

second theoretical limiting factor is the maximum number of processes (pid_max = 32768) on 32bit OS, while it is 4 million on 64bit OS, and the actual number of threads(processes) never reaches the limit.

B. Forwarding Performance

Frame(packet) forwarding performance is measured through the experiments. Frame(Packet) forwarding performance was measured for the network shown in Fig.14 with PC1 in Table I. Throughputs were measured with **iperf**[23], and delay and loss were measured with ping command.

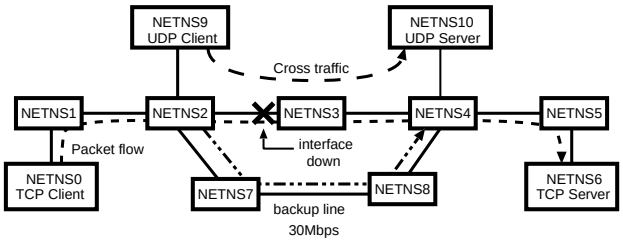

Fig. 14. Small network with dynamic routing(RIP)

1) Network Namespace Routers: The network in Fig.14 consists of 7 IPv4 routers and 4 IPv4 hosts. All hosts and routers are represented by network namespaces. The purpose of the example is to measure throughput and to demonstrate slow route change of RIP. Therefore, RIPv2 in Quagga[24] is invoked on each router. Link capacities of all links are unlimited except for the link between **NETNS7** and **NETNS8**, which is of 30 Mbps with the FrameQueue.

Table III shows throughput between **NETNS0** and **NETNS6** when network namespaces are directly connected with Virtual Ethernet Pairs (veth). Therefore, link capacities are unlimited. For pure IPv6 network, only NETNS0 to NETNS6 are emulated with static routing for throughput comparison. The results

TABLE III
THROUGHPUT (NO FRAMEQUEUE)

from NETNS0	TCPv4/v6(Mbps)	UDPv4(Mbps)
to NETNS1 (hop 1)	1542/1460	1070
to NETNS2 (hop 2)	1013/949	1052
to NETNS3 (hop 3)	1010/722	970
to NETNS4 (hop 4)	925/631	889
to NETNS5 (hop 5)	861/555	818
to NETNS6 (hop 6)	800/500	766

in Table III are the upper limit of the performance, since only kernel function of network namespaces and virtual Ethernet pairs are used. Note that IPv6 TCP throughput is lower than IPv4. The reason might be TCP max segment is smaller in IPv6 and IPv6 packet forwarding performance in the kernel is lower than IPv4.

2) Network Namespace Routers and FrameQueues: Then Table IV shows the same model, but in addition to network namespaces and virtual Ethernet pairs, the network consists of FrameQueues, a timer, and datalink socket I/O.

TABLE IV
THROUGHPUT (W/ FRAMEQUEUE, NO LIMIT)

from NETNS0	TCP tput(Mbps)	UDP tput(Mbps)
to NETNS1 (hop 1)	828	683
to NETNS2 (hop 2)	$448 \times 2 = 896$	431
to NETNS3 (hop 3)	$302 \times 3 = 906$	240
to NETNS4 (hop 4)	$234 \times 4 = 936$	180
to NETNS5 (hop 5)	$194 \times 5 = 970$	162
to NETNS6 (hop 6)	$176 \times 6 = 1056$	104

The Maximum total throughput in the emulator, 1056 Mbps is in hop 6 case (176 Mbps on 6 links), but the end-to-end throughputs are much lower than the results in Table III because of 15 more threads and socket I/O.

One way to achieve higher throughput in this configuration is using GINE emulated routers instead of network namespace routers. Dynamic routing is, however, impossible with emulated routers.

3) External Hosts and Emulated Routers: Table V shows throughputs measured with iperf (one way traffic) between two external real hosts connected via n number of emulated routers in series. Throughputs are higher than in the previous example

TABLE V
THROUGHPUT (CHAIN OF n EMULATED ROUTERS)

n	TCP(Mbps)	UDP(Mbps)
1	730	770
10	550	580
20	300	280
30	210	220
40	190	180
50	135	150
60	110	120

of Network Namespace routers except for $n = 1$ case. In this experiment, 730 Mbps might be the maximum throughput of the Linux kernel with the 1000BASE-T NIC used.

C. Link Emulation Accuracy

To evaluate delay emulation accuracy, different constant delays are imposed on each link but only one way (from left to right) in the first example. The left to right link delays between NETNS1 and NETNS5 are set as 50, 100, 200, 300 msec, respectively. And the measured total delay was 651.3 msec (650 msec theoretical).

Fig. 15 illustrates the result of shifted exponential random delay emulation. The dotted bold line in Fig. 15 denotes theoretical density function (constant 10 msec + exponential mean 10 msec). Emulated exponential distribution measured with ping in 10 msec interval did not show good match with the theoretical line.

Then experiment with 100 msec interval ping was conducted. In the case of 100 msec interval, it showed much better match. Exponential density is heavy tailed and not very realistic. The effect of correlation factor is clearly shown.

Successive packet transmission with random delay in short interval may cause reordering. In the samples used in Fig. 15, the numbers of reordering are 0, 1, and 1242 in 10000

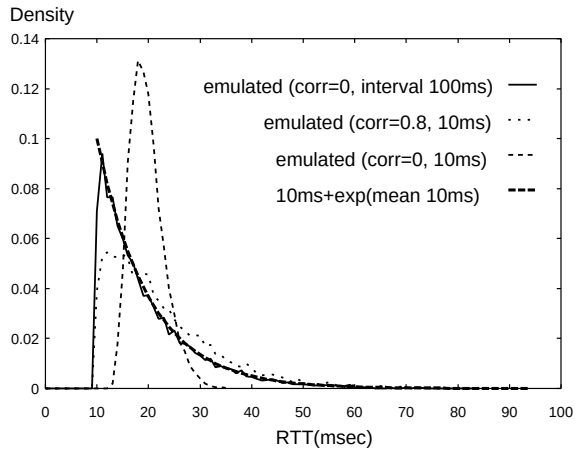

Fig. 15. Density of emulated exponential delay

packets for exponential with 100 msec interval, correlated with 10 msec interval, and exponential with 10ms, respectively.

Similarly to evaluate packet loss emulation accuracy, same packet loss probabilities of 0.1 are imposed on the left to right links between NETNS1 and NETNS5. The measured end-to-end packet loss was 0.327 and close to the theoretical value of $1 - 0.9^4 = 0.3439$.

Table VI shows the results for the effect of bandwidth limit. Note the bandwidth set by the emulator are in Ethernet frame (without counting preamble and FCS), and, therefore, TCP throughput is about 96% of the frame throughput. As

TABLE VI
THROUGHPUT (BANDWIDTH LIMIT)

Limit point	set bw(Mbps)	TCP bw(bps)
NETNS0 - NETNS1	1000	149.85M
NETNS0 - NETNS1	100	98.4M
NETNS0 - NETNS1	20	19.23M
NETNS0 - NETNS1	10	9.63M
NETNS0 - NETNS1	1	991.4k
NETNS0 - NETNS1	0.1	98.7k

the table shows, bandwidth limit up to 100 Mbps seems to work correctly. Unfortunately, 150 Mbps throughput for 1000 Mbps bandwidth is lower than expected. Improvement of the program is necessary.

VI. APPLICATIONS OF THE GINE AND ITS LIBRARIES

Educational examples and some examples of new applications with the GINE library classes are introduced in this section.

A. Routing experiments and Cross Traffic Injection

Small networks up to 20 nodes are relatively easily configured with the emulator by C++ programing or by using the GUI. To appeal the usefulness of the emulator and as educational example, consider the network of Fig.14.

The network interface of **NETNS3** was brought down during the emulation run to confirm route change. Routes were re-calculated in about 150 to 180 seconds after the link down.

Also an experiment with cross traffic injection were conducted. Fig.16 shows the change of TCP throughput from NETNS0 to NETNS6. Throughputs in this figure are calculated from **tcpdump** log. A TCP stream has been transmitted for 100 seconds from NETNS0 to NETNS6 (bandwidth 30 Mbps). During the TCP transmission, UDP cross traffic from NS9 to NS10 was injected from time 30 to 40 (second) at 10Mbps, and at 20Mbps from time 60 to 70 (second). As

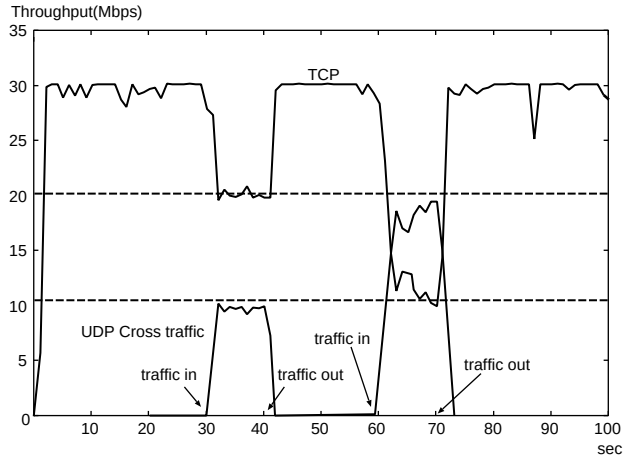

Fig. 16. Effect of UDP cross traffic to TCP traffic

Fig.16 shows TCP throughput went down by 10 Mbps from time 30 to 40 and by 20 Mbps from time 60 to 70.

B. Testing New Network Application within a PC

It is useful to develop a new network application in the emulated network, since real application runs on a virtual host and a testbed network is prepared in a PC. Not only Linux but other virtual OS can be connected via virtual network interface or packet diversion.

1) Traffic Limiter Bridge: We have been developing a network traffic limiter (called GateKeeper) as a Linux user space PC bridge software for limiting traffic against network attacks such as denial of services[25]. It is a custom bridge with bandwidth limit, delay, or packet loss filter. The application uses PFPacketIn/Out class for Ethernet frame input/output. Essential program components for this application are packet filters. The other components are easily written with Forwarder, FrameQueue, ProcessQueue, and Timer classes.

2) IPv6/v4 Translator: Fig.17 shows an example for IPv6/v4 translator and its testbed network. The translator uses PFPacketIn/Out class for Ethernet frame input/output. All the translation procedures are implemented in the user space application. IP and TCP/UDP/ICMP headers in a frame from a NIC(Network Interface Card) are translated with checksum recalculation according to the translation rules and a new frame is assembled and delivered to the other NIC. Long IPv4 packet is fragmented. Note that the translator runs on a real host as well as on a virtual host (Network Namespace). The network configuration was edited with the graphical user interface in this case.

26NSpace around the center denotes the translator, and left and right rectangles represent IPv4 and IPv6 network,

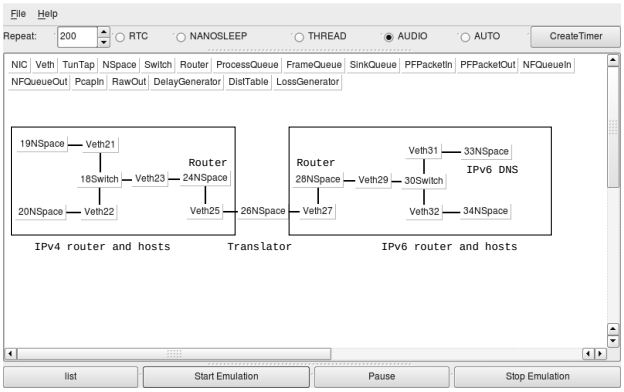

Fig. 17. IPv6/v4 translator and testbed network

respectively. **Translator** is a separate special bridge program using GINE library classes. A DNS server program is invoked on **33NameSpace** to represent IPv6 DNS server. Being able to emulate seven hosts on a single PC and to invoke arbitrary program on virtual hosts are major merits of the emulator.

C. Event Driven Simulation

Event driven simulation or custom node emulation can be easily programmed with GINE library classes. For example, a simulation program for M/M/1 queueing system is written as follows. 1) Define a Server class as a subclass of Forwarder and connect a FrameQueue to the Server as the event input queue. 2) Describe the state transition and generate events (next arrival, next departure) and their scheduled time with random number generator (DelayGenerator class), then store the events in the event queue. 3) In the main program, create Server, register the FrameQueue in a ProcessQueue, start Timer, generate the first arrival. By adjusting timescale, the program will run in real-time, fast, or slow.

VII. Conclusion

In this research, we have re-designed the network emulator, GINE (Goto's IP Network Emulator).

In the new version, realistic dynamic routing and host emulation have been successfully implemented on standard Linux kernel. Also, graphical user interface is added, and, therefore, the emulator becomes useful to those who do not have programming experience. The source codes and manuals will be available soon at **http://h303c0.sd.nanzan-u.ac.jp/GINE/**.

Future works includes improvement of the graphical user interface and frame forwarding speed, also, porting to other operating system, writing help message and manuals.

References

[1] Empirix Inc. Empirix hammer test solution. (accessed Mar. 2012). [Online]. Available: http://www.empirix.com/

[2] Shunra Software Ltd. Virtual enterprise product family. (accessed Mar. 2012). [Online]. Available: http://www.shunra.com/

[3] Simena. Simena ne. (accessed Mar. 2012). [Online]. Available: http://www.simena.net/NetworkEmulator.htm

[4] Packetstorm Communications, Inc. Home page(product guide). (accessed Mar. 2012). [Online]. Available: http://www.packetstorm.com/

[5] ZTI Computing & Telecom. Netdisturb. (accessed Mar. 2012). [Online]. Available: http://www.zti-telecom.com/pages/main-netdisturb.htm

[6] L. Nussbaum and O. Richard, "A comparative study of network link emulators," in *Proc. of Communications and Networking Simulation Symposium 2009 (CNS)*, 2009.

[7] L. Rizzo. Ip_dummynet. (accessed Mar. 2012). [Online]. Available: http://info.iet.unipi.it/~/ip_dummynet/

[8] M. Carson and D. Santay, "Nist net – a linux-based network emulation tool," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 111–126, 2003, (`http://www.itl.nist.gov/div892/itg/carson/nistnet/`).

[9] M. Zec, "Operating system support for integrated network emulation in imunes," in *Operating System and Architectural Support for the on demand IT InfraStructure / ASPLOS-XI*, Boston, America, 2004.

[10] S. Wang and C. Chou, "Innovative network emulations using the nctuns tool," in *Computer Networking and Networks*, S. Shannon, Ed. Nova Science Publishers, 2006, ch. 7, pp. 159–189, (`http://nsl10.csie.nctu.edu.tw/`).

[11] A. Ihara, S. Murase, and K. Goto, "Ipv4/v6 network emulator using divert socket," in *Proc. of 18th International Conference on Systems Engineering(ICSE2006)*, Coventry, UK, 2006, pp. 159–166.

[12] IPdivert project. Divert sockets for linux. (accessed Mar. 2012). [Online]. Available: http://sourceforge.net/projects/ipdivert/

[13] Y. Sugiyama and K. Goto, "Design and implementation of a network emulator using virtual network stack," in *Proc. of the Seventh International Symposium on Operations Research and Its Applications (ISORA2008), also in Lecture Notes in Operations Research*, vol. 8, 2008, pp. pp.351–358.

[14] Harald, W. libnetfilter_queue project. (accessed Mar. 2012). [Online]. Available: http://www.netfilter.org/projects/

[15] lxc Linux Containers. lxc Linux Containers. (accessed Mar. 2012). [Online]. Available: http://lxc.sourceforge.net/

[16] J. Ahrenholz, C. Danilov, T. Henderson, and J. Kim, "Core: A real-time network emulator," in *Proc. of IEEE MILCOM Military Communications Conference 2008(MILCOM)*, 2008, pp. 221–227.

[17] J. Ahrenholz, "Comparison of core network emulation platforms," in *Proc. of IEEE Military Communications Conference Conference 2010 (MILCOM)*, 2010, pp. 864–869.

[18] J. Ahrenholz, T. Goff, and B. Adamson, "Integration of the core and emane network emulators," in *Proc. of IEEE MILCOM Military Communications Conference 2012 (MILCOM)*, 2011, pp. 1870–1875.

[19] A. Jurgelionis, J.-P. Laulajainen, M. Hirvonen, and A. Wang, "An empirical study of netem network emulation functionalities," in *Proc. of 20th International Conference on Computer Communicatins and Networks (ICCCN)*, 2011, pp. 1–6.

[20] T. Kawamoto and K. Goto, "Design and evaluation of ip multipath transmission with feedback," in *Systems Science, Vol.35/No.2*, Nov. 2009, pp. 73–79.

[21] Free Software Foundation. Gnu common c++. (accessed Mar. 2012). [Online]. Available: http://www.gnu.org/software/commoncpp/

[22] Nokia. Qt - a cross-platform application and ui framework. (accessed Mar. 2012). [Online]. Available: http://qt.nokia.com/

[23] NLANR. Iperf. The tcp/udp bandwidth measurement tool. (accessed Mar. 2012). [Online]. Available: http://dast.nlanr.net/projects/Iperf/

[24] Ishiguro Kunihiro, et al. Quagga software routing suite. (accessed Mar. 2012). [Online]. Available: http://www.quagga.net/

[25] M. Aoyama, M. Kojima, and K. Goto, "Design and implementation of a traffic limiter for network security," in *Proc. of 16th International Conference on Systems Science*, vol. II, 2007, pp. 213–220.

Kunio Goto was born in Shiga, Japan, in 1957. He received the B.E. degree from Kyoto University, in 1980, and the M.E. and Doctor of Engineering in applied mathematics and physics from Kyoto University, Kyoto, Japan in 1982 and 1987, respectively.

In 1985, he joined the Department of Business Administration, Nanzan University, Nagoya, Japan as an assistant professor, and in 1998 became a Professor. Since 2000, he has been with the Department of Telecommunication Engineering, and since 2009, Department of Systems Design and Engineering, Nanzan University, Seto, Japan. His current research interests include performance evaluation of computer network and computer network security. He has been a member of IEEE, ACM, IPSJ, IEICE, and ORSJ.