The 7th International Symposium on Operations Research and Its Applications (ISORA'08)
Lijiang, China, October 31–Novemver 3, 2008
Copyright © 2008 ORSC & APORC, pp. 351–358

Design and Implementation of a Network Emulator using Virtual Network Stack

Yusuke Sugiyama[1] Kunio Goto[1]

[1]Graduate School of Mathematical Sciences and Information Engineering
Nanzan University
27 Seirei-cho, Seto, Aichi 489-0863, Japan

Abstract

In this research, we have extended the functionality of the network emulator, we developed on Linux and FreeBSD. The previous version requires custom Linux or custom FreeBSD kernel for packet diversion. Also, it lacks of many functionalities of real router or host, such as dynamic routing, application servers and clients.

In the new version, realistic dynamic routing and host emulation have been successfully implemented with standard Linux kernel 2.6.26. The emulator has become a more powerful network performance evaluation tool although it still requires programming in C++ and is in lacks of graphical user interface.

Keywords network emulation; performance evaluation; Internet protocol

1 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 runs in real time.

We have been developing a software called GINE (Goto's IP Network Emulator)[2], which emulates a wide area network with many routers and links. For each link, delay and/or loss, according to a given probability distribution function, are easily emulated, and also bandwidth can be limited to the capacity given. However, it requires custom Linux or custom FreeBSD kernel for packet diversion and is not easy to use for non-expert users. Also, it lacks of many functionalities of real router or host, such as dynamic routing, application servers and clients.

In this research, the emulator has been extended in two points: avoiding kernel customization and adding real router/host functionality. To achieve the first goal, packet diversion scheme is modified to use Netfilter NFQUEUE[7], which is available in Linux kernel 2.6.14 or later. For the second objective, Network Namespace[5], which is a virtual kernel network stack implementation, is used. Network Namespace is available in standard Linux kernel 2.6.26 (released in July, 2008).

Using both of emulated router and real router/host (of virtual network stack), the emulator becomes a more powerful network performance evaluation tool for network professional and educators.

2 System Architecture

In this section, design goal and system architectures are described.

2.1 Design Goal

The most popular network emulator to emulate an internet link (or end-to-end path) is probably NISTNet[6]. Although it runs fast with Linux kernel implementation, it works only for IPv4 and filter does not support range of IP addresses. While kernel implementation is the most efficient way for the emulation, an implementation as a user space program is more flexible. Therefore, we have developed the emulator as a user space program. The processing speed of out emulator with recent multi-core CPU is better than expected.

Flexible Link Emulation for IPv4/v6

The first extension to the emulator is to add IPv6 link emulation and filtering with range of IP addresses, protocol, and ports. An example is shown in Fig.1.

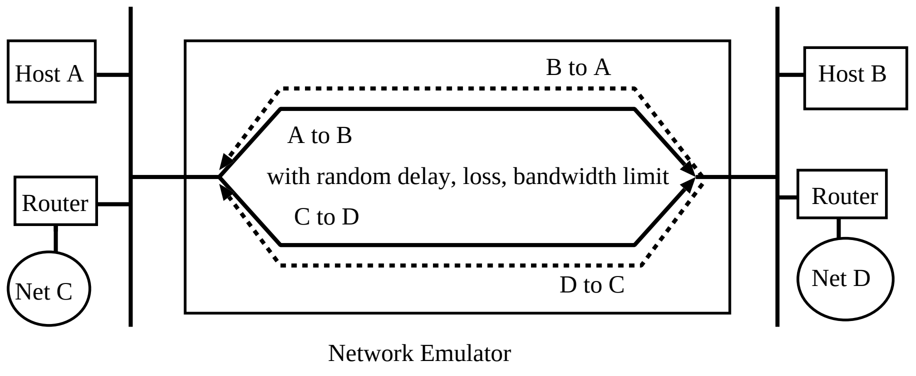

Figure 1: Usecase1: Simple link emulation

As shown in Fig.1, our emulator should 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 parameters for the other direction can be defined differently.

The first enhancement to the emulator is to add IPv6 link emulation and filtering with range of IP addresses, protocol, and ports. An example is shown in Fig.1. Note that packets are normally forwarded by the kernel routing function. Then packets should be diverted to the emulator program.

Network Topology Emulation in the Emulator

The second extension is to represent a network topology consists of routers and links. Fig.2 illustrates an example.

In Fig.2, the packets from Net C to D and D to C are separately diverted to the emulator processes and go through the four emulated routers. In addition to the point-to-point link emulation, cross traffic can be added to a specific link. For example, this emulation is useful for testing voice or video conferencing between Net C and D.

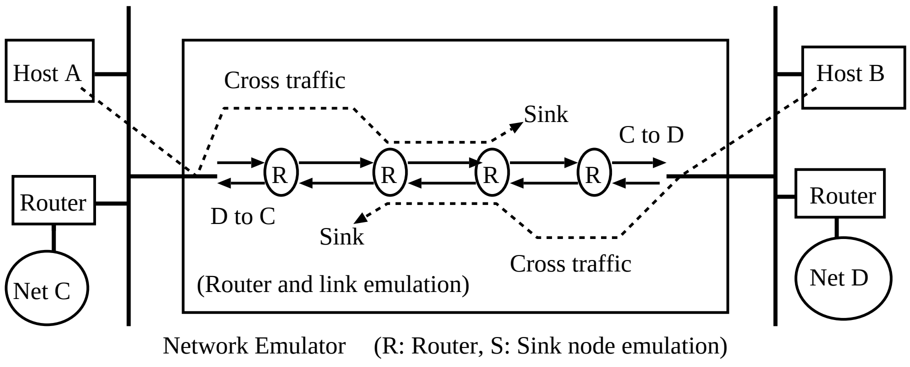

Figure 2: Usecase2: Link and router emulation with cross traffic

Cross traffic can be injected at any emulated router from external hosts as long as the total traffic is less than network interface capacity. In the figure, cross traffic is generated by Host A and injected at the second router from left and exits at the third router. Sink node simply absorbs input packets. Similarly, cross traffic from Host B can be added. Also packet generator in the emulator can be easily implemented.

In the case of TCP cross traffic, a Sink node does not work. A server or client process should talk with Host A. For more complex network topology and multicast emulation, Dynamic routing functionality is necessary for more complex network topology and multicast emulation.

Host/Router Emulation with Virtual Hosts

More realistic network model consists of routers, virtual hosts, and switch or wireless links as shown in Fig.3.

Implementing dynamic routing in the emulated router is like “re-inventing a wheel” and rather complex and time consuming task. It is recommended to implement a dynamic protocol in the emulator only for testing a new routing protocol. Also, implementing server programs for application protocols in the emulator is not an effective way.

There are some alternative to virtual host emulation as follows:

- Bind an application to virtual network interface (tuntap[9])
- Virtual host (User Mode Linux, Xen. etc.)
- Virtual network stack (Network namespace)

Virtual host or virtual network stack emulates a separate host in terms of complete network function including routing/forwarding function. Usefulness of virtual network interface is limited to launch application process bound to a different IP address, and modification to the existing network application program.

Virtual host provides full host functionality including file system and process space separation at rather high cost for memory and diskspace. Virtual network stack gives the functionality necessary just for the network emulation purpose. Fortunately, Network

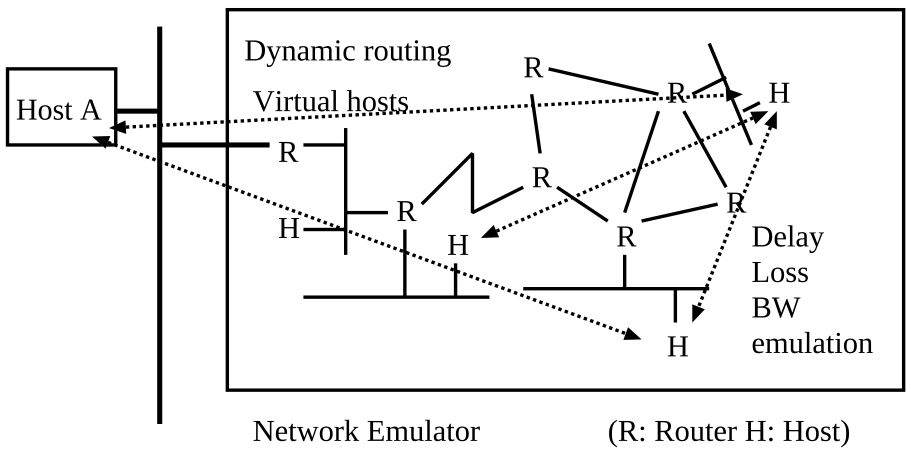

Figure 3: Usecase3: Dynamic routing, host, and link emulation

Namespace has been officially included in the recent Linux kernel (2.6.26) as a part of kernel container functionalities[5] and we apply it to our network emulator.

For example, running dynamic routing with RIP (Routing Information Protocol) is just running routed (RIP daemon) in the created virtual host with multiple network interfaces.

3 Implementation

In this section, implementation issues are briefly discussed.

3.1 Packet(frame) Diversion and Input/Output

Packet diversion to the user space network emulation is essential for the proposed network emulator. The following schemes have been tested;

- Packet diversion with filter and I/O
  - IPv4/v6 Divert Socket in the Linux custom kernel (with netfilter)
  - IPv4/v6 NFQUEUE in standard Linux kernel
- Frame I/O with input filter
  - Packet capture library (libpcap for tcpdump)
  - Datalink Layer socket API and IP layer filter (netfilter)

As mentioned above, while divert socket is not a standard Linux kernel feature, NFQUEUE is included in the Linux kernel 2.6.14. Filter for packet diversion is defined with "iptables" command except for the case of packet capture library. Packet capture library can be used for older kernels but is rather slow.

A diverted packet should be simply write back to the device used for diversion, if the packet is re-injected to the point where it was diverted. Otherwise, the diverted packets are sent with a RAW IP socket or datalink layer socket.

3.2 Software Components

The emulator program is written in C++ with Thread and a several classes in GNU common C++ library. The essential object classes are illustrated in Fig.4.

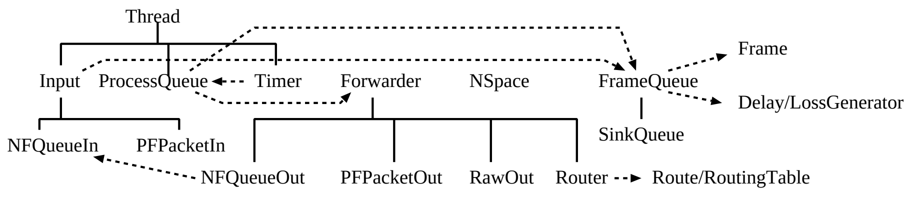

Figure 4: Software Components: Essential Classes

An Input class object receives diverted packets and put them into a FrameQueue object. It runs autonomously as a thread with wait until a packet arrives. NFQueueIn and PFPacketIn are the classes for input from NFQUEUE and datalink layer socket.

A Forwarder class object is not a thread. Instead a ProcessQueue object periodically checks for all the registered FrameQueue objects and calls the packet forwarding method in the Forwarder class to forward the packet in the head of the FrameQueue object to a network interface or anther FrameQueue object. ProcessQueue object waits for a wakeup call from a Timer thread after having checked the registered FrameQueue objects to avoid using too much CPU time. A Timer object is a periodical timer in a very short period of 200 micro sec or less. Current implementation uses either Linux high resolution timer, Linux Real Time Clock(RTC), audio input at 48kHz.

3.3 Link Emulation with Custom Queue

A FrameQueue is the component for link delay, loss, and bandwidth limit emulation, while it can be used as a generic datalink frame buffer. For details, see [2], since there is no modification for FrameQueue class.

FrameQueue is a bidirectional linked list of Frame objects arranged in the order of scheduled departure time. 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 packet at the head leaves the queue, transmission time schedule of the next packet is obtained as max (departure time of the next packet, transmit time of the head packet + 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.

3.4 Communication between Emulator and Virtual Host/Router

Network interfaces including loopback are not shared among virtual network stacks and the host's network stack. A special virtual network device, called Virtual Ethernet

Pair, is provided for communication between them. When a frame is received by one side of a pair, it is forwarded to the other side. Therefore, communication between independent network stacks becomes possible by moving one side to a virtual network stack. Fig.5 shows the way of communication between them in the proposed network emulator.

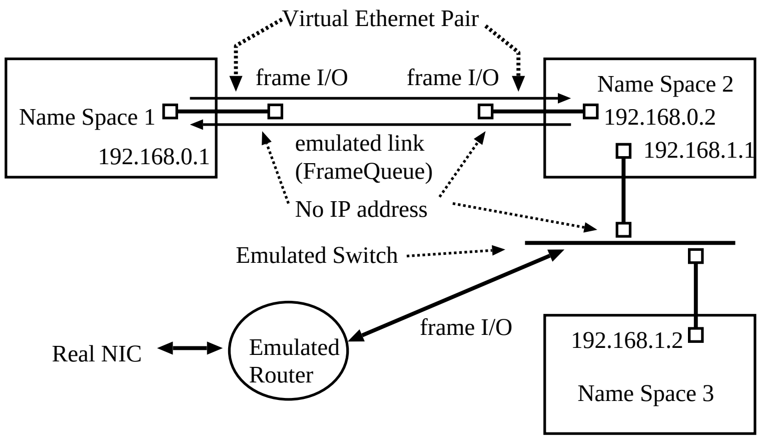

Figure 5: Communication between Emulator and Virtual Host/Router

Note that IP address is not assigned to the emulator host side of a virtual Ethernet pair although IP addresses can be assigned to both side. If IP address is assigned to the emulator host side of the virtual Ethernet pair, a packet either from outside of the emulator host or the host itself will be directly forwarded to the interface and, therefore, the packet bypasses the emulator.

4 Example and Performance

In this section, a short program example is introduced and some performance evaluation results are presented.

4.1 Program Example

Fig.6 is a very simple test program. Some lines are omitted for briefness. The readers may understand the the program writing style with this short example as the following steps;

1. Define objects in the order of dependency,
2. Connect object according to the packet(frame) flow (in both direction if apply),
3. Start threads,
4. Modify object and parameters according to the scheduled time with sleep function (This part is not included in the example).

In the example program, only IPv4 ICMP packets from 192.168.0.249 are diverted and received at NFQueue number 1 (NFQueue number is 0 to 65535). Then received frames(packets) are put into FrameQueue, q0, which emulates link delay of 100msec.

```
// Create and register queues
 FrameQueue   q0(0.1,0.0,100000,1000.0,"in"); // 100msec delay
 FrameQueue   q1(0.0,0.0,100000,1000.0,"out");
 ProcessQueue pq(sync); pq.add(q0); pq.add(q1);
// divert icmp from 192.168.0.249 to NFQueue #1
 system("/sbin/iptables -A FORWARD -p icmp
    -s 192.168.0.249 -j NFQUEUE -queue-num 1");
// Receive from NFQueue #1 to q0
 NFQueueIn in("In", q0, 4, 1);
// Router setup
 Router r0("r0",2); q0.setOutput(r0, 0); r0.setOutLine(1, q1);
 r0.addRoute(AF_INET, "0.0.0.0/0",1);
// Virtual host setup
 NSpace ns0("ns0");
 ns0.createIfPair("veth00", "veth01");
 system("/sbin/ifconfig veth00 192.168.1.254");
 ns0.execCommand("/sbin/ifconfig veth01 192.168.1.1");
 ns0.execCommand("/sbin/route add default gw 192.168.1.254");
// Input from q1 to virtual host
 RawOut rawout0("rawout0","veth00",4); q1.setOutput(rawout0,0);
// start threads
 in.start(); timer.start(); pq.start();
```

Figure 6: Example Program

Packets in q0 are received by interface 0 of Router, r0, then forwarded to FrameQueue, q1. Finally packets in q1 is transmitted by RawOut, rawout0, to host side of the virtual Ethernet pair, veth00,and finally get to the virtual host, ns0, via veth01. In this case, an IP address is given to veth00, and the packets of the other direction are directly back to 192.168.0.249.

4.2 Packet Forwarding Performance

Packet forwarding performance were measured with "iperf"[3] in TCP and UDP mode. The emulation program was executed on Intel dual core CPU with PCI Express 1000Base-T NICs. The results are shown in Table 1.

Table 1: Packet Forwarding Performance with Chain of *n* 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

As shown in the figure, the throughput was surprisingly high. It is higher than the native Linux router (no emulation). It might be a benefit from large socket receive buffer space and large buffers in the FrameQueue objects.

5 Conclusion

In this research, we have extended the functionality of the network emulator, which we developed on Linux and FreeBSD.

In the new version, realistic dynamic routing and host emulation have been successfully implemented only with standard Linux kernel 2.6.26. Packet forwarding performance is as good as 750Mbps with fast dual core CPU and fast NIC. The emulator has become a more powerful network performance evaluation tool.

It still requires programming skill in C++ and is in lacks of graphical user interface. The program is open source and will be available at the author's Web site (`http://kmgoto.jp/`).

References

[1] F. S. Foundation. Gnu common C++. `http://www.gnu.org/software/commoncpp/`, (accessed 2008).

[2] A. Ihara and K. Goto. IPv4/v6 network emulator using divert socket. *Proc. of 18th International Conference on Systems Engineering(ICSE2006, Coventry, UK)*, pages 159–166, September 2006.

[3] NLANR. Iperf: The tcp/udp bandwidth measurement tool. `http://dast.nlanr.net/projects/Iperf/`, (accessed 2008).

[4] Tcpdump/libpcap `http://www.tcpdump.org/` (accessed June, 2008).

[5] Linux Containers – Network Namespace, `http://lxc.sourceforge.net/network.php` (accessed June, 2008).

[6] NIST Net `http://snad.ncsl.nist.gov/itg/nistnet/` (accessed June, 2008).

[7] W. Harald. libnetfilter_queue project, `http://www.netfilter.org/projects/libnetfilter_queue/index.html` (accessed June, 2008).

[8] OpenVZ, `http://wiki.openvz.org/` (accessed June, 2008).

[9] M. Krasnyansky and M. Yevmenkin. Universal driver tun tap. `http://vtun.sourceforge.net/tun/`, (accessed 2008).