Experimental Evaluation Linux Routing Performance

CS898 Project Final Report

Chaoxing Lin
Dec 20, 2001

Content

Introduction

Having an open source Linux router allows us to introduce different kinds of impairments, such as packet dropping, packet delay, queue reordering. In this project, we will set up Linux (Kernel 2.4.10) network environment, develop some tools, introduce packet dropper in router, test the performance and compare the experimental result to simulation result.

Chapter 1 talks about the NS. Chapter 2 introduces the Linux system setting and network environment. Chapter 3 talks about the tools used in this project. In chapter 4 we measure the performance of the environment before packet dropper is applied. Chapter 5 applies packet dropper module and measure the performance. Chapter 6 compares the result from regular dropping, random dropping, and the result from NS. Chapter 7 will open to discussion on why these 3 results are far apart from one another. Chapter 8 is the conclusion. Chapter 9 lists the references.

Acknowledgements

I wrote this document as a wrap-up of my Master's project for the Computer Science Department of the University of New Hampshire. I would like to take this opportunity to show my appreciation to Dr. Radim Bartos my advisor. He gave me many resources, a lot of help and encouragement. I also appreciate JieZou's work on NS simulation. Special thanks go to Glenn Herrin, thanks for his document "Linux IP Networking". I learned a lot from Glen's document. I also would like to thank Professor Bob Russell. Thanks for his "CS 820 OS Programming" course. I learned a great deal on Linux system and network programming.

Chapter 1 Linux Network Infrastructure

This chapter gives the computer system information, network configuration.

1.1 Computer System Information

Table 1.1 shows the CPU, memory, OS information of the computers that are used in the project. This information is from `/proc/cpuinfo`, `/proc/meminfo`, `uname -a`

Machine	CPU Information	Memory Info	OS Info
Dublin.cs.unh.edu	Pentium III (728.455MHz)	Mem:256MB Swap:1GB	Red Hat Linux 7.0 kernel2.4.10
Madrid.cs.unh.edu	Pentium III (728.448MHz)	Mem:512MB Swap: 1GB	Red Hat Linux 7.0 kernel2.2.16
Prague.cs.unh.edu	Pentium III (728.458MHz)	Mem:256MB Swap:1GB	Red Hat Linux 7.0 kernel2.2.16

Table 1.1 Computers used in this project.

1.2 Network Configuration Information

Table 1.2 shows the Network interfaces setting in the project. This information is got from the result of command "`ifconfig`" and "`/sbin/lsmod`". The basic diagram is shown at Figure 1.1.

Machine	Interface	Chip	IP address	Mac Address
Dublin.cs.unh.edu	*100Mb/s* Eth0	3Com 3c90x	132.177.8.28/25	00:B0:D0:FE:D8:09
	100Mb/s Eth1	Tulip	192.168.2.1/24	00:C0:F0:6A:56:51
	100Mb/s Eth2	Tulip	192.168.1.1/24	00:C0:F0:6A:6D:0C
Madrid.cs.unh.edu	*100Mb/s* Eth0	3Com 3c90x	132.177.8.27/25	00:B0:D0:D8:FD:EA
	100Mb/s Eth1	Tulip	192.168.1.2/24	00:C0:F0:6A:74:F1
	100Mb/s Eth2	Tulip	192.168.3.1/24	00:C0:F0:6A:74:ED
Prague.cs.unh.edu	*100Mb/s* Eth0	3Com 3c90x	132.177.8.29/25	00:B0:D0:D8:FE:91
	100Mb/s Eth1	Tulip	192.168.2.2/24	00:C0:F0:6A:6D:4E
	100Mb/s Eth2	Tulip	192.168.3.2/24	00:C0:F0:6A:75:10

Table 1.2 Network interfaces settings.

1.2.1 Basic Diagram

```
                      -----------------
                      |     switch     |
                      |                |
                      |  10    9    11  |
                      -----------------
                          |    |      |
                          |    |      |
                          |    |      |  132.177.8.0/25
      -------------------      |      -------------------
      |                        |                        |
      |                        |                        |
      |                        |                        |
      |                        |                        |
-----------------    ------------------    ---------------------
|     eth0      |    |      eth0      |    |       eth0        |
|               |    |                |    |                   |
|    Madrid     |    |      Dublin    |    |      Prague       |
|               |    |                |    |                   |
| eth2   eth1   |    | eth2     eth1  |    | eth1     eth2     |
-----------------    ------------------    ---------------------
3.1|     |1.2        1.1|       |2.1        2.2 |       | 3.2
   |     ----------------       ----------------        |
   |         192.168.1.0/24          192.168.2.0/24    |
   |                                                   |
   -----------------------------------------------------
                 192.168.3.0/24
```

Figure 1.1 Infrastructure diagram.

Experiments in this chapter refer to this route:

Madrid:eth1 ==> Dublin:eth2 ==> Dublin:eth1 ==> Prague:eth1

Dublin is used as router.

1.2.2 Static Route Setting:

Now we show the route setting in these machines. We get this information by command "route".

? **Madrid.cs.unh.edu**

```
Destination Gateway Genmask Flags Metric Ref Use Iface
132.177.8.0    *    255.255.255.128  U      0     0   0   eth0
192.168.3.0    *    255.255.255.0    U      0     0   0   eth2
192.168.2.0 192.168.1.1  255.255.255.0  UG     0     0   0   eth1
192.168.1.0    *    255.255.255.0    U      0     0   0   eth1
127.0.0.0      *    255.0.0.0        U      0     0   0   lo
default    phub0.cs.unh.edu  0.0.0.0     UG     0     0   0   eth0
```

? **Dublin.cs.unh.edu**

```
Destination Gateway Genmask Flags Metric Ref Use Iface
132.177.8.0  *     255.255.255.128  U     0     0   0   eth0
192.168.3.0 192.168.2.2 255.255.255.0 UG    0     0   0   eth1
192.168.2.0  *       255.255.255.0  U     0     0   0   eth1
192.168.1.0  *     255.255.255.0    U     0     0   0   eth2
127.0.0.0    *       255.0.0.0      U     0     0   0    lo
default  phub0.cs.unh.edu 0.0.0.0  UG    0      0   0   eth0
```

? **Prague.cs.unh.edu**

```
Destination Gateway Genmask Flags Metric Ref Use Iface
132.177.8.0   *   255.255.255.128   U     0     0   0   eth0
192.168.3.0   *   255.255.255.0     U     0     0   0   eth2
192.168.2.0   *   255.255.255.0     U     0     0   0   eth1
192.168.1.0 192.168.2.1 255.255.255.0 UG    0     0   0   eth1
127.0.0.0     *   255.0.0.0         U     0     0   0     lo
default   phub0.cs.unh.edu 0.0.0.0   UG    0     0   0   eth0
```

1.2.3 TCP parameters on the machines we are using

Table 1.3 shows the TCP information of Dublin.cs.unh.edu (kernel 2.4.10). Table 1.4 shows the TCP information of Madrid and Prague (kernel 2.2.16). We get this table from proc file system. `/proc/sys/net/ipv4/tcp*`

tcp_max_tw_buckets	180000	tcp_ecn	1
tcp_mem	48128 48640 49152	tcp_syn_retries	5
tcp_orphan_retries	0	tcp_fack	1
tcp_reordering	3	tcp_synack_retries	5
tcp_retrans_collapse	1	tcp_fin_timeout	60
tcp_retries1	3	tcp_syncookies	0
tcp_retries2	15	tcp_keepalive_intvl	75
tcp_abort_on_overflow	0	tcp_timestamps	1
tcp_rfc1337	0	tcp_keepalive_probes	9
tcp_adv_win_scale	2	tcp_tw_recycle	0
tcp_rmem	4096 87380 174760	tcp_keepalive_time	7200
tcp_app_win	31	tcp_window_scaling	1
tcp_sack	1	tcp_max_orphans	8192
tcp_dsack	1	tcp_wmem	4096 16384 131072
tcp_stdurg	0	tcp_max_syn_backlog	1024

Table 1.3 Dublin TCP version information.

tcp_max_tw_buckets	N/A	tcp_ecn	N/A
tcp_mem	N/A	tcp_syn_retries	10
tcp_orphan_retries	N/A	tcp_fack	N/A
tcp_reordering	N/A	tcp_synack_retries	5
tcp_retrans_collapse	1	tcp_fin_timeout	180
tcp_retries1	7	tcp_syncookies	0
tcp_retries2	15	tcp_keepalive_intvl	N/A
tcp_abort_on_overflow	N/A	tcp_timestamps	1
tcp_rfc1337	0	tcp_keepalive_probes	9
tcp_adv_win_scale	N/A	tcp_tw_recycle	N/A
tcp_rmem	N/A	tcp_keepalive_time	7200
tcp_app_win	N/A	tcp_window_scaling	1
tcp_sack	1	tcp_max_orphans	N/A
tcp_dsack	N/A	tcp_wmem	N/A
		tcp_max_ka_probes	*5*
tcp_stdurg	0	tcp_max_syn_backlog	128

Table 1.4 TCP version information on Madrid and Prague.

Chapter 2 Tools used in the project

This chapter introduces the tools that will be used in this project and their source codes.

2.1 Blast Test Code (By Prof. Bob Russell, UDP version is by Chaoxing Lin)

Most of the performance experiments will use blast test tools. Here is how it works. Client side sends a number (*iteration times, specified from command line*) of packets of given *request size* (*specified from command line*) on end and close the connection. Server side keeps receiving packets and swallows it. Then when client closes connection it sends 1 byte back. For the complete blast test code, please refer to http://www.cs.unh.edu/cnrg/lin/linuxProject/phase3/nettest

2.2 Packet Dropper Code (By Glenn Herrin, Modified by Chaoxing Lin)

The first impairment we introduce to the router kernel is packet dropper module. This is a kernel module. It randomly drops packet with a specified destination address at a given rate. Code are inserted and executed on virtual DEVICE layer.

On receiving each packet, the Dropper checks the destination IP address. If the destination is the target IP, we get a 16-bit random number, if the random number is less than the dropping threshold value which is ***rate****65535, we drop this packet, otherwise, process it normally.

```
/***********************packet_dropper****************
This is what dev_queue_xmit will call while this module is installed
****/

int packet_dropper(struct sk_buff *skb)
{
  unsigned short t;
  if (skb->nh.iph->daddr == target) {

  /* the following code is modified by lin: begin (*/
        t = getUnsignedShortRandom();
        if (t < cutoff)
        {
        number_of_packet_dropped++;
        return 1;    /* drop this packet */
        }
  /* modified by lin : end ) */

  }
  return 0;            /* continue with normal routine */
}  /* packet_dropper */
```

The random number is generated in this way. This way is only good for Intel processor because the assembly instruction “rdtsc” is only in Intel processor.

```
inline unsigned short getUnsignedShortRandom()
{
        unsigned l, h;
        unsigned short low;
        unsigned char * lp;
        unsigned char * hp;
        unsigned char ldata;

    /* get a CPU cycle. Only good for Intel processor */
         __asm__ volatile("rdtsc": "=a" (l), "=d" (h));

         /* get the lower 16 bits */
         low = (unsigned short) l & 0xFFFF;

         /****Swap lower byte with the higher byte *******/
         hp=(unsigned char*)(&low);/*point to high byte */
         lp=hp+1;                   /* point to low byte */
         /* swap the higher byte with the lower byte */
         ldata = *lp;
         *lp = *hp;
         *hp = ldata;
         return low;
}
```

The packet dropper module also exports symbols that will be used in proc file system:

```
unsigned short cutoff;           /* drop threshold value */
float rate;                    /* drop percentage */
unsigned number_of_packet_dropped;/* #of packet dropped */
_u32 target = 0x0202A8C0;      /* address 192.168.2.2 */
```

Kernel source changed:

/usr/src/linux/net/core/dev.c

at line: 942 Add: *int (*xmit_test_function)(struct sk_buff *) = 0;*
See the context:

```
.....
    919 #ifdef CONFIG_HIGHMEM
    920 /* Actually, we should eliminate this check as soon as we know,
that:
    921  * 1. IOMMU is present and allows to map all the memory.
    922  * 2. No high memory really exists on this machine.
    923  */
    924
    925 static inline int
    926 illegal_highdma(struct net_device*dev,struct sk_buff *skb)
    927 {
    928     int i;
    929
    930     if (dev->features&NETIF_F_HIGHDMA)
    931         return 0;
    932
```

```
    for (i=0; i<skb_shinfo(skb)->nr_frags; i++)
         if (skb_shinfo(skb)->frags[i].page >=highmem_start_page)
            return 1;

    return 0;
}
#else
#define illegal_highdma(dev, skb)   (0)
#endif

/* ADDED BY LIN to test packet-dropper:begin (   */

int (*xmit_test_function)( struct sk_buff * ) = 0;
/* ADDED BY LIN to test packet-dropper:end   )  */

  /**
  *  dev_queue_xmit - transmit a buffer
  *  @skb: buffer to transmit
  *
  * Queue a buffer for transmission to a network device. The caller
 * must have set the device and priority and built the buffer
 *before calling this  function. The function can be called from an
 *interrupt  A negative errno code is returned on a failure. A
 *success does not
  * guarantee the frame will be transmitted as it may be dropped due
  *  to congestion or traffic shaping.
  */
........
```

In function: int dev_queue_xmit(struct sk_buff *skb)
At the very beginning, add:

```
if ( xmit_test_function && ( *xmit_test_function )(skb) )
{
    kfree_skb( skb );
    return 0;
}
```

See the context:

```
...........
int dev_queue_xmit(struct sk_buff *skb)
{
    struct net_device *dev = skb->dev;
    struct Qdisc  *q;

/* ADDED BY LIN TO TEST PACKET-DROPPER :  BEGIN (     */
if ( xmit_test_function && ( *xmit_test_function )(skb) )
{
    kfree_skb( skb );
    return 0;
}
/* ADDED BY LIN TO TEST PACKET-DROPPER :  END   )     */

    if (skb_shinfo(skb)->frag_list &&
```

```
        !(dev->features&NETIF_F_FRAGLIST) &&
        skb_linearize(skb, GFP_ATOMIC) != 0) {
        kfree_skb(skb);
        return -ENOMEM;
    }
...........
```

/usr/src/linux/net/netsyms.c
at line: 570 Add:

*extern int (*xmit_test_function) (struct sk_buff *);*
EXPORT_SYMBOL_NOVERS(xmit_test_function);

After changing kernel source code, don't forget to re-compile the kernel.

For the complete packet_dropper code, please refer to
http://www.cs.unh.edu/cnrg/lin/linuxProject/phase3/random_dropper/

2.3 Proc File System Operation Code

In order to check the exact number of packets dropped by the packet dropper and to easily change dropping rate and destination IP address, we develop a kernel module to do this job. It dynamically probes current dropping rate, destination packet, number of packet dropped. It can also set these values.

```
static struct proc_dir_entry *dropperInfo, *fileEntry;
static int proc_reset_num(struct file *file,const char *
                        buffer,unsigned long count,void *data);
static int proc_read_num(char *buf, char **start,
                off_t off, int count,int *eof, void *data);
static int proc_read_dropper(char *buf, char **start,
                off_t off, int count,int *eof, void *data);
static int proc_write_dropper(struct file *file,const char
                        *buffer,unsigned long count,void *data);

int init_module()
{
        int rv = 0;
        EXPORT_NO_SYMBOLS;
        /* create directory */
        dropperInfo = proc_mkdir("dropperInfo", NULL);
        if(dropperInfo == NULL) {
                printk("<1> dropperInfo failed\n");
                rv = -ENOMEM;
                goto out;
        }
        dropperInfo->owner = THIS_MODULE;

        /* create "dropper" and "numDropped" files */
        fileEntry = create_proc_entry("dropper", 0644, dropperInfo);
        if(fileEntry == NULL) {
```

```
                rv = -ENOMEM;
                goto error;
        }
        fileEntry->read_proc = proc_read_dropper;
        fileEntry->write_proc = proc_write_dropper;
        fileEntry->owner = THIS_MODULE;

        fileEntry = create_proc_entry("numDropped", 0644, dropperInfo);
        if(fileEntry == NULL) {
                rv = -ENOMEM;
                goto error;
        }
        fileEntry->read_proc = proc_read_num;
        fileEntry->write_proc = proc_reset_num;
        fileEntry->owner = THIS_MODULE;

        /* everything OK */
        printk(KERN_INFO "%s  initialized\n", MODULE_NAME );

        return 0;
        error:
        remove_proc_entry(MODULE_NAME, NULL);
        out:    return rv;
}
```

Inserting this module to kernel will create a directory **/proc/dropperInfo**. In this directory there will be 2 files:

dropper:
    Use “cat /proc/dropperInfo/dropper” to see the rate, ip, # of packet dropped.
    Use “input ***rate ip_addr***” to reset rate and ip_addr dynamically.
numDropped:
    Use “cat /proc/dropperInfo/numDropped” to see # of packet dropped since last reset.
    Use “reset” to set “# of packet dropped” to 0

For the complete proc operation code,
please refer to *http://www.cs.unh.edu/cnrg/lin/linuxProject/phase3/proc/*

For the proc file system handling, see
http://www.cs.unh.edu/cnrg/lin/linuxProject/resource/proc.pdf

2.4 Tool used to synchronize router and source host

For some the experiments, as far as the control is on the same machine, we can write a shell script to let the experiments run again and again. But it’s almost impossible to write a script to control remote machines. In here, it’s really hard to write script to synchronize the router and the sending host. We need to run blast test for a few times at a given dropping rate. Then on router side, we need to set a new dropping rate and let experiments go on.

Our goal is to reduce the human interactions to the minimum when we do these experiments. The automation tool will also make it easy to repeat the experiments with the same settings sometime later.

Based on this, we develop a simple tool to synchronize the router and sending host. This tool contains 2 applications

"dublin" is to be run as server on dublin.cs.unh.edu:5678.
"madrid" is to be run as client on madrid.cs.unh.edu.

Step 1. Dublin installs "packet dropper" module.

Step 2. Dublin installs "proc file system manipulation" module.

Step 3. Dublin opens server port 5678 and waits for connection. And after connection is created, it waits for control message from Madrid and do as instructed. Control messages are:

SET_RATE ***rate***: Dublin sets dropping rate to ***rate.*** It also creates a directory with the name ***rate.*** Number of packet loss at this ***rate*** will be put under this directory.

RECORD_LOSS: Dublin checks the proc file system, get the number of packets dropped since last reset. And then it resets the number_of_packet to 0.

SET_ITERATION ***times***: (used only in regular dropping test with dropping rate 1/6). Dublin creates a directory with the name ***times.*** Number_of_packet loss will be put under this directory.

Step 4 Prague opens "blast test server"(using port 1026 because it's hard coded in this tools)

Step 5.Madrid opens tool client. It sends control message to Dublin to set rate. After it gets acknowledgement, it does experiment(blastclient 1026 192.168.2.2 100000 1448). After each experiment, Madrid sends control message to Dublin to record the packet loss. For each specific rate, Madrid does 10 times experiments and then increment the rate and set it to Dublin.

Note: Don't forget to redirect the result to a file. We will use this file to do statistics.

Step 6 On Dublin, run script "collectRawDropNum" whose content is:

```
cd dirName
for d in `ls`;
do
      cd $d
```

```
                echo $d >> /home/lin/Tools/rawDrop
                 for f in `ls`;
                do
                        cat $f >> /home/lin/Tools/rawDrop
                done
                cd ..
        done
        cd ..
```

(Suppose that ***dirName*** is the directory created by "dublin") we will get a file with the number of packets dropped for each experiment.

Run "getNum ***rawDrop dropStat***", we will get the statistical result of "number of packet dropped during each experiment"

Step 7. On Madrid, Suppose that we redirect the experiment result to a file "***rawElapse***", run "getElapse ***rawElapse elapseStat***" we can get the statistical result. By this result we can run gnuplot to draw the graph.

For the complete AutoTools code,
please refer to *http://www.cs.unh.edu/cnrg/lin/linuxProject/phase3/AutoTools/*

Chapter 3 Performance Test Result without Packet Dropper

Before we introduce the packet dropper module to the router, we would like to gather the basic performance (such as system throughput, packet delay) of our infrastructure.

3.1 Max Throughput got from the infrastructure

No packet dropper is installed in router (Dublin.cs.unh.edu) so far.

3.1.1 Get throughput by TCP blast test

Table 3.1 shows that the maximum throughput we can get from my Linux Environment is: 12374712 Bytes/Sec. It's very close (about 1% less than) to 100Mb/sec.

Blast test when request size is optimal (1448 Bytes) PingTest													
	1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**	**AVG**	**Var**	**Throughput(Byte/Sec)**
ElapseTime (Sec)	123.21	123.19	123.26	123.35	123.22	123.64	123.32	123.33	123.47	123.17	**123.316**	**0.106**	**12374712**

Table 3.1 Performance test data by TCP blast test.

The original data are
http://www.cs.unh.edu/cnrg/lin/linuxProject/phase1/blastOptimal.htm

How do we get **12374712 Byte/sec?**

Although, we just send useful data **1448** Byte on each request, we actually send TCP and IP header and Ethernet Wrapper. We actually send **1526** Bytes for each frame. (1500Bytes for layer 2 data, 26Bytes for Ethernet packet wrapper, for header detail, see here). So actual throughput = 1526*1000000/123.316 = 12374712 Bytes/sec

It is (12500000 - 12374712)/12500000 = 1.0023% less than the Ideal Theoretical throughput 100Mb/sec

Note:

We know 1 Mb/sec in Ethernet specification is 1000000 bit/sec, 1KB = 1024 Byte, 1 Byte = 8 bits

`Fast Ethernet 100Mb/Sec = 10^8 bits/Sec =` **`12500000 Bytes/sec`**

3.1.2 Get throughput by UDP blast test

Table 3.2 shows that by UDP test, the maximum throughput we can get from my Linux environment is: 12374712 Bytes/Sec. It's very close (about 1% less than) to 100Mb/sec

UDP test when request size optimal(1472 Byte) Sending 1000000 udp packets																		
	1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**	**11th**	**12th**	**13th**	**14th**	**15th**	**AVG**	**Var**	**Throughput(Byte/sec)**
ElapseTime (Sec)	123.27	123.48	123.35	123.45	123.45	123.34	123.17	123.19	123.37	123.61	123.20	123.35	123.08	123.11	123.32	**123.316**	**0.1168**	**12374712**
Packet Loss	42	0	0	0	0	108	0	0	0	0	49	0	51	0	61			

Table 3.2 Performance got by UDP test.

The elapse time original data is:
http://www.cs.unh.edu/cnrg/lin/linuxProject/phase1/clientInfoOptimal
The packet loss original data is:
http://www.cs.unh.edu/cnrg/lin/linuxProject/phase1/serverInfoOptimal

It's pretty interesting that the Average Elapse Time is the same as that in TCP optimal case. Incredible!!

So actual throughput = 1526*1000000/123.316 = 12374712 Bytes/Sec

It is (12500000 - 12374712)/12500000 = 1.0023% less than the Ideal Theoretical throughput 100Mb/sec

3.2 Infrastructure Delay

In this section we will find the delay of the infrastructure that we will use to do experiments. This parameter will also be used in NS (network simulator).

We will use ICMP ping packet with different packet size to find out the packet delay in our infrastructure.

```
madrid$ ping -U -c 12 -s requestSize 192.168.2.2
```

Table 3.3 shows the result of ping requests with different ICMP packet request size.

ICMP RequestSize	**56byte**	**64Byte**	**128Byte**	**256byte**	**384Byte**	**512Byte**	**640byte**	**768Byte**	**896Byte**	**1024Byte**	**1152 Byte**	**1280Byte**	**1408Byte**	**1472Byte**
1st	158	173	197	255	291	351	379	420	473	507	570	587	642	659
2nd	150	182	222	245	282	333	370	410	453	496	541	593	629	660
3rd	160	173	197	257	288	327	379	421	468	528	558	636	649	651
4th	174	195	224	239	284	337	376	410	464	501	538	597	666	659
5th	160	176	202	244	311	321	376	451	468	518	549	585	645	651
6th	151	189	204	244	286	329	367	413	482	494	538	595	629	676
7th	163	172	193	263	303	321	375	421	463	506	550	583	639	647
8th	159	179	202	250	284	332	371	420	453	500	540	600	632	658
9th	157	177	210	247	295	335	399	447	461	508	562	593	651	650
10th	160	182	215	239	283	332	373	410	453	497	537	591	627	658
Average	159.2	179.8	206.6	248.3	290.7	331.8	376.5	422.3	463.8	505.5	548.3	596	640.9	656.9
Variation	4.2	5.76	8.92	6.36	7.44	5.84	5.5	10.68	7.2	7.9	9.5	9	9.7	5.72

Table 3.3 Data from ICMP ping test

Figure 3.1 shows the relation between RTT and packet request size:

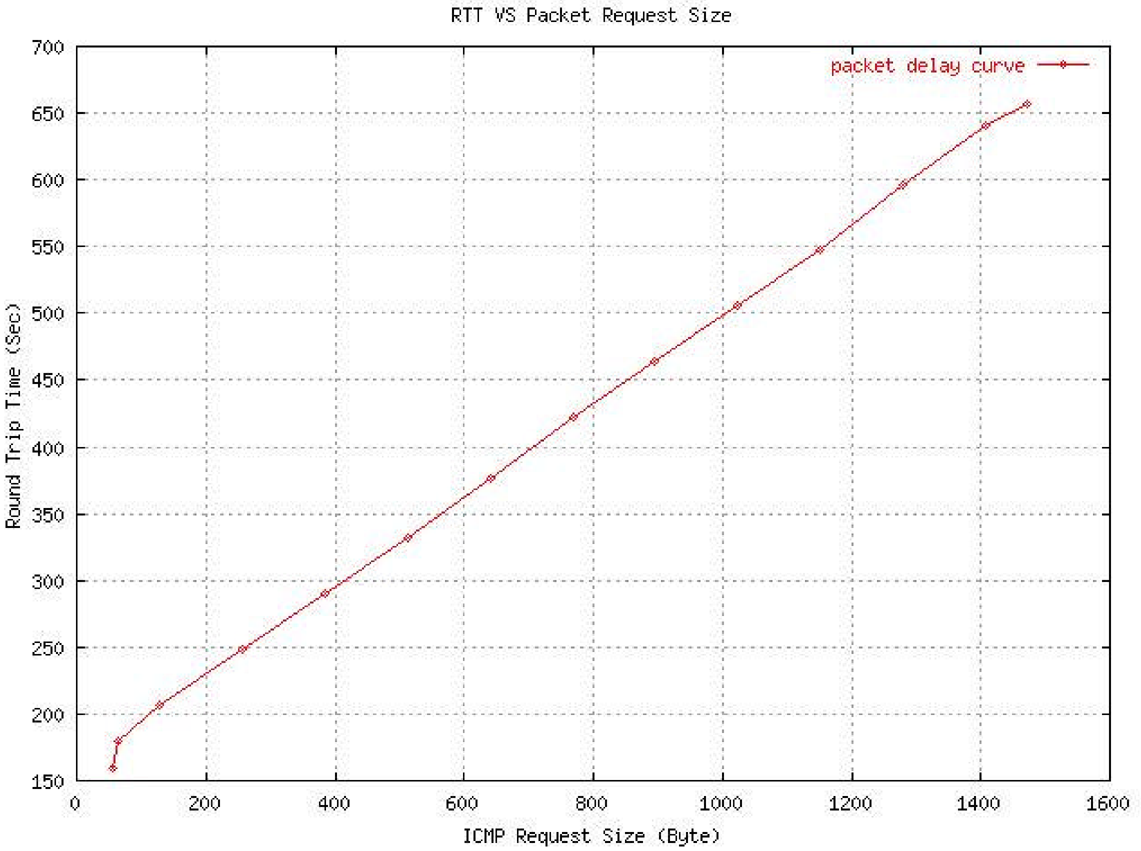

Figure 3.1 Relations between ICMP Packet Request Size and RTT

In Figure 3.1, we can use the line passing point (256,248.3) and point (1408,640.9) to approximate this line

RTT = a * Request_Size + b

so
248.3 = a*256 + b
640.9 = a*1408 + b

Solve the above equations, we can get a = 0.3408 , b = 161.0556

Theoretically,

RTT/2 = delay + 2*(Packet_Size/Rate)

(Packet_Size = Request_size + ICMP header(8) + IP header (20)+Ethernet Wrapper(26))

RTT = 2*delay + 4*((Request_Size+8+20+26)/Rate)
RTT = Request_Size * (4/Rate) + (2*delay + 216/Rate)

Where Rate = 100Mb/sec = 12.5MB/sec = 12.5 B/µs

4/Rate = 4/12.5 = 0.32 which is very close to the data calculated from the experiment 0.3408

b = 2*delay + 216/Rate = 161.0566

So delay = (161.0556 - 216/12.5)/2 = 71.8878 (µs**)** **=72** µs

Chapter 4 Performance Test Result with Packet Dropper

In this chapter, we will introduce packet dropper, both random dropping version and regular dropping version.

4.1 Random Dropping

On receiving each packet, the Dropper checks the destination IP address. If the destination is the target IP, we get a 16-bit random number, if the random number is less than the dropping threshold value which is ***rate****65535, we drop this packet, otherwise, process it normally.

The random number is generated in this way. Because the assembly instruction “rdtsc” is specific to Intel processor, this way is only good for it.

```
Get a CPU cycle
        unsigned l, h;
        __asm__ volatile("rdtsc": "=a" (l), "=d" (h));
Get the lower 16 bit from the CPU cycle. Swap the lower byte with the higher byte.
```

In the following calculation:

The AVG refers to the average value of 10 times’ experiments result (either elapse time or number of packet dropped).

The Var is the variation of the 10 times’ experiment result.

It is: sqrt(((t1-avg)**2 + (t2-avg)**2 + ...+ (t10-avg)**2)/10) Where ti is the experiment result (either elapse time or number of packets dropped). And i is 1…10.

The throughput is: 1526*100000/ AVG

Figure 4.1 shows the relation chart of throughput and dropping rate.

Table 4.1 shows the detail data of the test result.

Drop Rate	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	AVG	VAR	Throughput(B/s)
0.000	12.32	12.31	12.31	12.31	12.31	12.31	12.31	12.31	12.31	12.31	12.311	0.003	12395419
0.004	12.58	12.59	12.60	12.78	12.40	12.59	12.79	12.79	12.59	12.59	12.630	0.117	12082344
0.008	13.63	13.43	13.22	13.92	13.21	13.78	13.81	13.40	14.38	14.37	13.715	0.402	11126504
0.012	14.93	14.43	16.20	15.59	15.91	16.09	16.76	15.41	15.60	15.98	15.690	0.629	9725940
0.016	18.21	17.94	16.63	18.59	18.19	15.77	18.68	16.64	18.65	18.91	17.821	1.026	8562932
0.020	19.03	19.29	21.32	20.36	18.80	20.64	20.73	22.64	21.26	19.39	20.346	1.159	7500246
0.024	23.72	25.32	22.86	23.00	23.81	26.15	23.38	24.46	22.97	23.08	23.875	1.055	6391623
0.028	27.50	27.87	30.08	27.19	28.50	31.01	27.45	33.13	32.90	30.96	29.659	2.155	5145150
0.032	34.62	37.09	35.42	37.10	31.75	33.26	29.71	33.38	35.41	35.08	34.282	2.198	4451316
0.036	40.21	36.55	38.61	37.46	39.47	36.10	43.48	40.70	39.05	42.04	39.367	2.224	3876343
0.040	47.50	43.83	49.85	48.76	51.44	50.21	48.39	48.02	51.76	46.92	48.668	2.222	3135531
0.044	57.63	58.25	56.20	58.41	60.10	57.71	59.47	61.01	63.59	57.05	58.942	2.064	2588986
0.048	60.53	62.69	60.53	62.10	60.77	64.72	69.53	63.17	61.82	64.32	63.018	2.585	2421530
0.052	69.94	71.16	77.68	71.36	81.50	78.39	75.87	83.58	80.62	72.28	76.238	4.610	2001627
0.056	81.00	81.88	87.80	88.41	80.69	83.04	86.63	82.62	82.11	83.47	83.765	2.672	1821763
0.060	94.69	88.28	85.34	98.35	100.85	91.36	93.02	100.69	91.31	99.59	94.348	5.138	1617416
0.064	102.81	113.83	112.53	119.74	118.44	107.60	111.76	101.93	108.02	110.67	110.733	5.584	1378090
0.068	128.40	118.65	107.80	123.88	121.82	122.60	129.09	121.18	132.55	120.71	122.668	6.459	1244008
0.072	137.22	140.05	129.92	136.85	142.88	138.54	133.10	148.12	135.43	143.60	138.571	5.066	1101241
0.076	149.76	138.79	158.41	152.24	167.07	154.32	150.96	164.03	154.89	157.76	154.823	7.489	985642
0.080	157.50	168.43	172.89	171.53	184.27	166.37	155.31	173.43	160.68	160.15	167.056	8.432	913466
0.084	194.28	196.97	186.59	181.34	190.61	189.30	193.47	201.84	206.89	194.90	193.619	6.964	788146
0.088	209.22	205.81	224.02	217.21	214.00	216.19	227.36	210.13	207.06	221.41	215.241	6.975	708973
0.092	228.31	220.76	224.77	241.97	223.95	229.48	227.43	213.72	205.09	230.78	224.626	9.485	679351
0.096	250.06	252.02	270.32	248.97	244.95	237.94	249.31	249.61	266.07	307.15	257.640	18.789	592299
0.100	268.61	275.46	280.16	279.69	283.42	268.90	288.08	253.52	274.59	271.92	274.435	9.147	556052
0.104	317.36	276.15	307.06	292.50	310.21	326.06	321.80	330.96	318.36	305.04	310.550	15.630	491386
0.108	325.51	344.83	340.28	295.20	302.02	334.39	327.88	351.86	285.12	306.06	321.315	21.620	474923
0.112	359.97	352.99	353.36	370.75	352.03	347.78	319.44	353.36	346.82	375.23	353.173	14.319	432083
0.116	405.68	385.94	393.06	419.70	373.77	391.31	379.12	370.57	368.25	387.88	387.528	15.269	393778
0.120	426.30	422.77	401.62	424.46	426.22	376.22	420.19	400.01	418.08	403.23	411.910	15.461	370469
0.124	446.80	508.11	445.87	503.94	491.30	440.11	444.95	435.18	460.41	464.42	464.109	25.861	328802
0.128	486.39	469.44	539.99	473.81	467.21	535.21	497.34	466.53	506.08	434.08	487.608	31.134	312956
0.132	527.16	532.07	586.13	511.39	543.10	536.87	563.76	492.69	491.36	546.67	533.120	28.166	286239
0.136	600.00	566.77	558.85	608.03	556.17	592.22	558.70	579.79	570.13	625.97	581.663	22.720	262351
0.140	644.62	616.15	631.62	633.08	633.37	624.63	579.93	594.24	594.61	572.06	612.431	24.013	249171
0.144	692.91	616.93	645.45	626.70	630.92	643.30	628.59	767.10	695.61	683.01	663.052	44.209	230148
0.148	720.46	696.01	706.25	703.32	717.07	744.12	728.69	675.32	713.80	724.99	713.003	18.108	214024

Table 4.1 Performance result of random packet dropping.

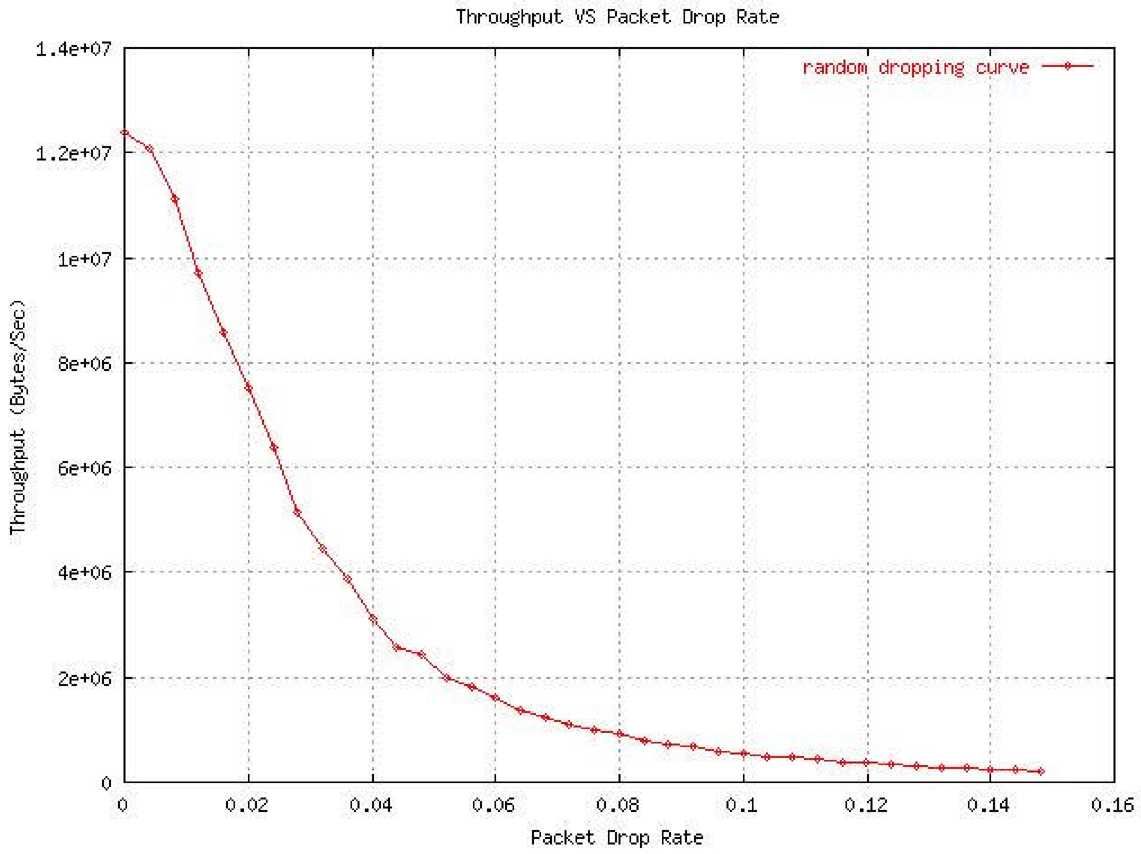

Figure 4.1 Performance Curve of Random dropping.

Through /proc file system, we can see the actual number of TCP packets that are dropped. Table 5.2 shows the packet loss.

The IDEAL is the ideal number of packets dropped during the experiments with a specific dropping rate.

The deviation is (IDEAL - AVG)/IDEAL

Drop Rate	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	AVG	VAR	IDEAL	DEV
0.000	0	0	0	0	0	0	0	0	0	0	0	0	0	0.000
0.004	392	403	409	366	381	407	424	418	414	370	398	19	402	-0.008
0.008	840	864	814	842	785	812	861	814	774	797	820	29	806	0.017
0.012	1214	1200	1198	1229	1235	1227	1205	1191	1237	1191	1213	17	1215	-0.002
0.016	1674	1606	1598	1635	1661	1621	1566	1672	1662	1684	1638	37	1626	0.007
0.020	2087	2056	2003	2081	2019	2097	2003	2007	1972	2047	2037	40	2041	-0.002
0.024	2521	2480	2552	2465	2396	2583	2468	2418	2449	2458	2479	55	2459	0.008
0.028	2794	3029	2845	2857	2824	2868	2881	3003	2927	2859	2889	72	2881	0.003
0.032	3312	3333	3247	3320	3246	3274	3310	3216	3420	3334	3301	55	3306	-0.001
0.036	3827	3682	3766	3693	3607	3648	3835	3792	3763	3741	3735	72	3734	0.000
0.040	4204	4153	4196	4186	4274	4247	4121	4241	4231	4218	4207	43	4167	0.010
0.044	4589	4604	4646	4675	4602	4577	4639	4772	4738	4553	4640	67	4603	0.008
0.048	4873	5070	5011	5069	4940	4990	5203	4947	5056	5094	5025	89	5042	-0.003
0.052	5418	5573	5629	5366	5582	5497	5530	5674	5501	5430	5520	92	5485	0.006
0.056	5968	6009	5961	6055	5834	5833	5928	5946	5814	5791	5914	86	5932	-0.003
0.060	6307	6441	6375	6468	6454	6375	6308	6276	6423	6347	6377	64	6383	-0.001
0.064	6722	6866	6823	6873	7029	6859	6926	6780	6711	7033	6862	106	6838	0.004
0.068	7427	7325	7056	7346	7313	7287	7325	7202	7379	7429	7309	105	7296	0.002
0.072	7712	7846	7572	7802	7727	7879	7811	7734	7768	7764	7762	81	7759	0.000
0.076	8217	8206	8371	8384	8315	8264	8163	8369	8249	8175	8271	79	8225	0.006
0.080	8496	8726	8652	8690	8815	8718	8523	8825	8763	8670	8688	104	8696	-0.001
0.084	9487	9263	9250	9197	9325	9188	9333	9219	9189	9263	9271	87	9170	0.011
0.088	9733	9820	9672	9619	9854	9906	9666	9733	9751	9806	9756	86	9649	0.011
0.092	10184	10175	10195	10297	10167	10132	10193	10067	9974	10291	10168	91	10132	0.003
0.096	10650	10612	10899	10644	10607	10674	10718	10476	10744	10752	10678	106	10619	0.005
0.100	11185	11268	10985	11324	11158	11127	11158	11185	11003	11209	11160	99	11111	0.004
0.104	11888	11621	11557	11810	11731	11811	11890	11840	11612	11606	11737	121	11607	0.011
0.108	12226	12187	12329	12120	12147	12328	12026	12218	12084	12099	12176	96	12108	0.006
0.112	12687	12707	12818	12793	12623	12826	12555	12737	12708	12968	12742	110	12613	0.010
0.116	13371	13215	13102	13361	13265	13061	13321	13222	13111	13192	13222	103	13122	0.008
0.120	13968	13859	13857	13893	13895	13690	13962	13651	13894	13686	13836	111	13636	0.015
0.124	14342	14444	14443	14563	14688	14505	14331	14334	14330	14389	14437	113	14155	0.020
0.128	14801	14781	14965	14999	14790	15046	15006	14716	14853	14657	14861	128	14679	0.012
0.132	15512	15429	15634	15257	15419	15592	15293	15346	15322	15687	15449	143	15207	0.016
0.136	15752	15881	16198	16112	16067	15917	15868	16144	16007	16005	15995	133	15741	0.016
0.140	16654	16292	16725	16609	16633	16638	16405	16552	16522	16614	16564	122	16279	0.018
0.144	17166	16893	17095	17004	16839	16998	16720	17209	16888	17138	16995	151	16822	0.010
0.148	17740	17630	17515	17664	17515	17566	17392	17570	17424	17671	17569	105	17371	0.011

Table 4.2 Number of packets dropped by router (Dublin).

4.2 Regular Dropping

On receiving each packet, the Dropper checks the destination IP address. If the destination is the target IP, we increment the counter, if the counter equals to cutoff (an integer), we drop this packet, otherwise, process it normally.

```
/******************************************************* packet_dropper
 * this is what dev_queue_xmit will call while this module is installed
 *********/

unsigned rate = 6;
static unsigned int count = 0;
unsigned number_of_packet_dropped=0;

int packet_dropper(struct sk_buff *skb) {
  if (skb->nh.iph->daddr == target) {
        count ++;
        if ( count == rate)
        {
                count = 0;
                number_of_packet_dropped++;
                return 1;
        }
  }
  return 0;                       /* continue with normal routine */
}  /* packet_dropper */
```

The source code is
http://www.cs.unh.edu/cnrg/lin/linuxProject/phase3/regular_dropper/

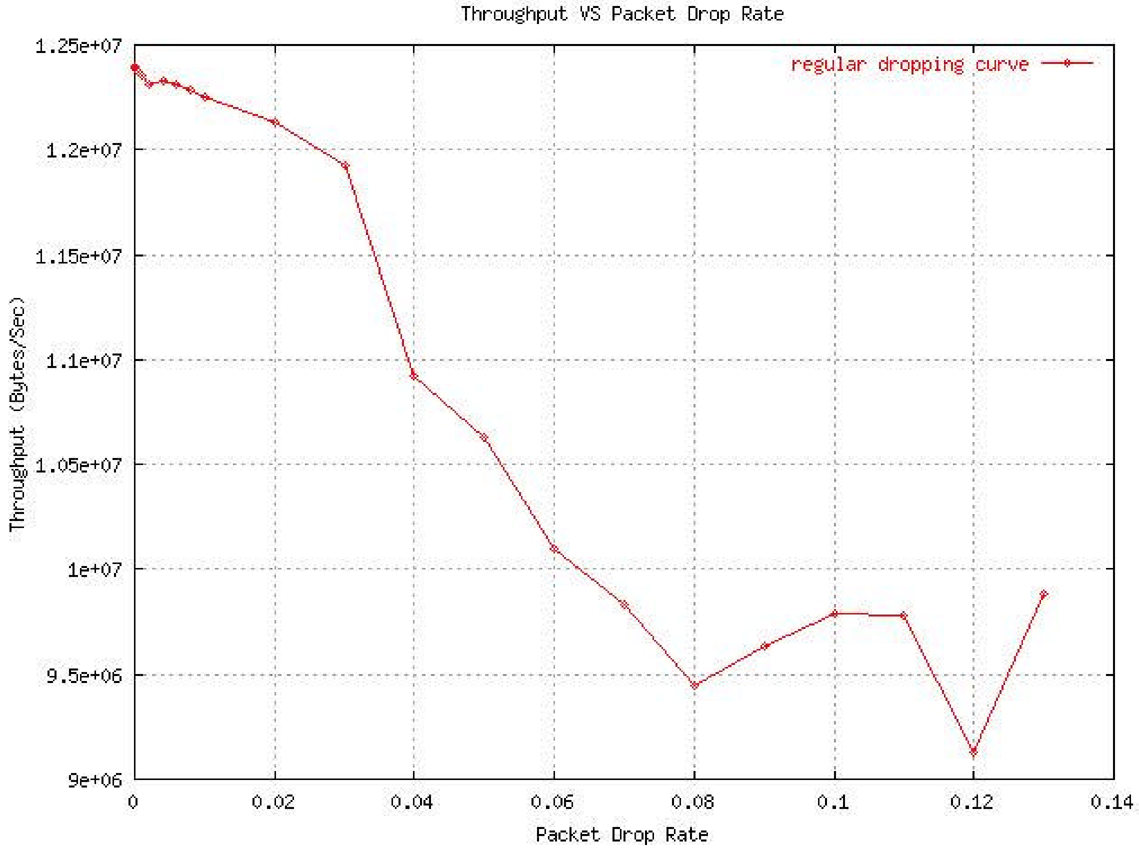

Figure 4.2 Performance curve of regular dropping.

Figure 4.2 shows the performance after the regular packet dropper is applied on the router.

Table 4.3 shows the experiment detail data. In Table 4.3, column "***cutoff***" means "drop 1 packet out of every ***cutoff*** packets". For the original data, see http://www.cs.unh.edu/cnrg/lin/linuxProject/phase2/nov5Tcp/

Experiment on packet_dropper. packets are dropped one out of every 1/rate packets														
drop rate	**cutoff**	**Elapse time (sec) Tcp request size 1448 bytes, iteration time 100000**												**Throughput(B/s)**
		1st	**2nd**	**3rd**	**4th**	**5th**	**6th**	**7th**	**8th**	**9th**	**10th**	**AVG**	**Var**	
0	**-**	12.33	12.31	12.31	12.31	12.31	12.31	12.31	12.31	12.31	12.31	**12.312**	0.004	**12394411.96**
0.0001	**10000**	12.33	12.32	12.32	12.32	12.32	12.32	12.32	12.32	12.32	12.32	**12.321**	0.002	**12385358.33**
0.0005	**2000**	12.34	12.33	12.33	12.33	12.33	12.33	12.33	12.33	12.33	12.33	**12.331**	0.002	**12375314.25**
0.001	**1000**	12.36	12.35	12.35	12.35	12.35	12.35	12.35	12.39	12.36	12.35	**12.356**	0.008	**12350275.17**
0.002	**500**	12.40	12.39	12.39	12.39	12.39	12.39	12.39	12.39	12.39	12.39	**12.391**	0.002	**12315390.20**
0.004	**250**	12.39	12.38	12.38	12.38	12.38	12.38	12.38	12.38	12.38	12.38	**12.381**	0.002	**12325337.21**
0.006	**166**	12.40	12.39	12.39	12.39	12.39	12.39	12.39	12.39	12.39	12.39	**12.391**	0.002	**12315390.20**
0.008	**125**	12.43	12.42	12.42	12.42	12.42	12.42	12.42	12.42	12.42	12.42	**12.421**	0.002	**12285645.28**
0.01	**100**	12.45	12.45	12.44	12.44	12.44	12.50	12.45	12.45	12.45	12.45	**12.452**	0.010	**12255059.43**
0.02	**50**	12.59	12.58	12.58	12.58	12.58	12.58	12.58	12.58	12.58	12.58	**12.581**	0.002	**12129401.48**
0.03	**33**	12.92	12.73	12.71	12.78	12.71	12.84	13.04	12.78	12.72	12.74	**12.797**	0.082	**11924669.84**
0.04	**25**	14.02	13.96	13.95	13.94	13.96	13.95	14.14	13.96	13.93	13.96	**13.977**	0.041	**10917936.61**
0.05	**20**	14.36	14.34	14.33	14.32	14.53	14.32	14.33	14.33	14.35	14.33	**14.354**	0.036	**10631182.95**
0.06	**16**	15.12	15.16	15.11	15.10	15.12	15.12	15.11	15.10	15.10	15.11	**15.115**	0.012	**10095931.19**
0.07	**14**	15.56	15.67	15.54	15.47	15.53	15.47	15.47	15.48	15.52	15.44	**15.515**	0.049	**9835642.93**
0.08	**12**	16.30	16.06	16.06	16.26	16.07	16.06	16.28	16.07	16.06	16.27	**16.149**	0.103	**9449501.52**
0.09	**11**	15.92	15.75	15.93	15.73	15.93	15.75	15.94	15.74	15.95	15.72	**15.836**	0.098	**9636271.79**
0.10	**10**	15.79	15.42	15.58	15.81	15.43	15.61	15.42	15.77	15.61	15.41	**15.585**	0.133	**9791466.15**
0.11	**9**	15.79	15.41	15.78	15.62	15.42	15.79	15.43	15.59	15.80	15.41	**15.604**	0.152	**9779543.71**
0.12	**8**	16.66	17.06	17.63	17.83	15.88	16.08	16.46	16.43	17.25	15.89	**16.717**	0.580	**9128432.13**
0.13	**7**	15.51	15.30	15.70	15.30	15.49	15.30	15.49	15.49	15.50	15.30	**15.438**	0.110	**9884700.09**

Table 4.3 Performance of regular dropping.

Figure 4.3 shows that given the same dropping rate, the regular dropping has a much better performance than the random dropping.

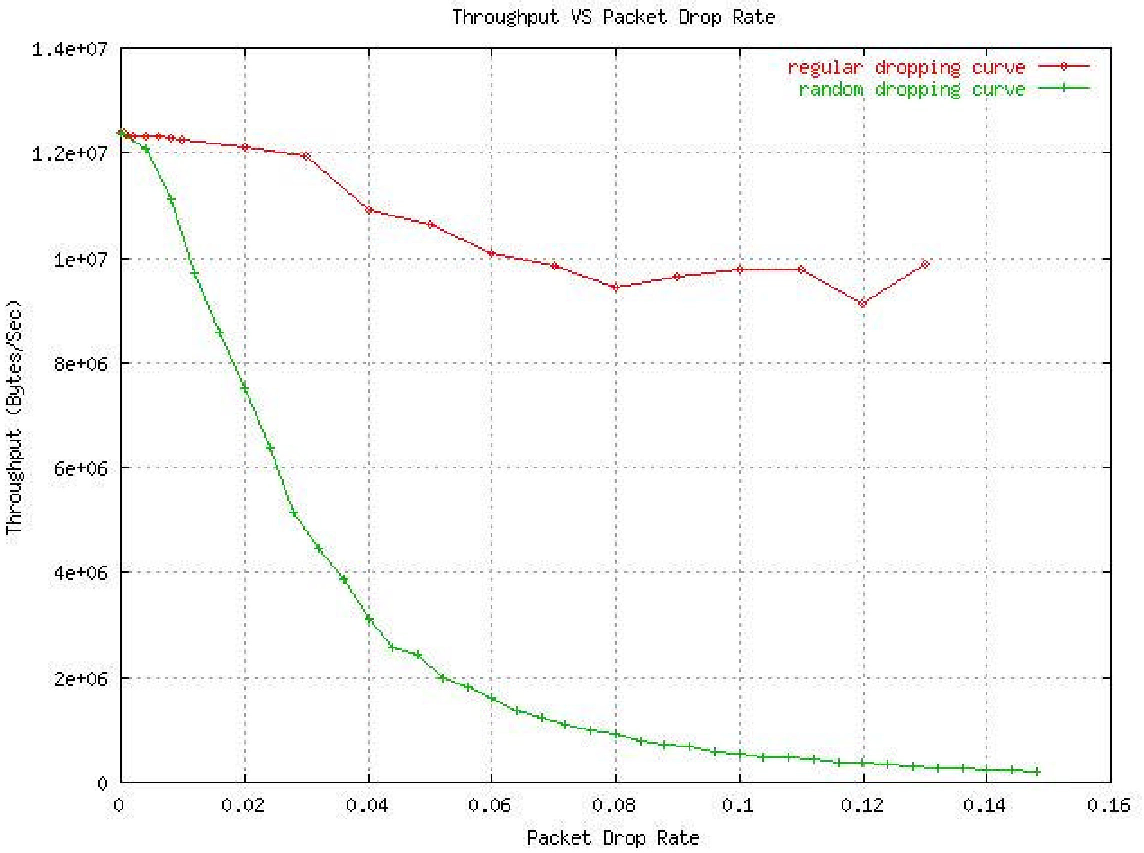

Figure 4.3 Performance comparison of random dropping and regular dropping.

4.3 Interesting Result on Regular Dropping

When we set regular dropping rate to 1/6 (regularly drop one packet out of every 6 packets). The elapse time varies significantly. Sometimes it takes only a few seconds to send 50000 packets. Sometimes it takes hundreds of seconds to finish it. And sometimes it even takes thousands of seconds to do that. Table 4.4 shows the experiment data.

Iteration Times	Time elapse (Second)											
	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	Random	
											1st	2nd
10000	1.77	**12.73**	1.77	1.77	1.78	1.77	1.77	**346.65**	1.78	1.77	**122.34**	**114.58**
20000	3.35	3.35	3.35	3.35	3.34	3.34	3.34	3.34	3.34	**455.36**	**181.67**	**189.52**
30000	4.91	4.92	4.92	4.92	**1971.68**	4.92	**652.67**	….....			**313.62**	**328.32**
40000	6.45	6.45	6.45	6.44	6.45	1129.76	….....				**385.51**	**392.27**
50000	8.01	**1597.12**	8.01	**1008.76**	**566.48**	8.01	….....				**611.35**	**528.09**
60000	9.58	9.57	9.57	**2006.39**	**2993.67**	9.60	….....				**583.14**	**614.00**
70000	10.95	**4946.53**	**1128.08**	**4944.91**	….....						**779.36**	**824.49**
80000	12.71	12.71	**4000.19**	**211.84**	….....						**765.21**	**819.63**

Table 4.4 Elapse time when regular dropping rate is set 1/6.

Chapter 5 Comparison between experimental and simulation results

In this chapter, we will compare the results we got from Random Dropper and from Network Simulator [2].

NS simulation script was written so that it mimics the experimental setup (same bit rate, same propaganda delay, same packet size and TCP version). Figure 5.1 shows that there is still significant difference between NS and experimental results. The Random Dropper has a much better performance than Network Simulator. Figure 5.2 shows the percentage.

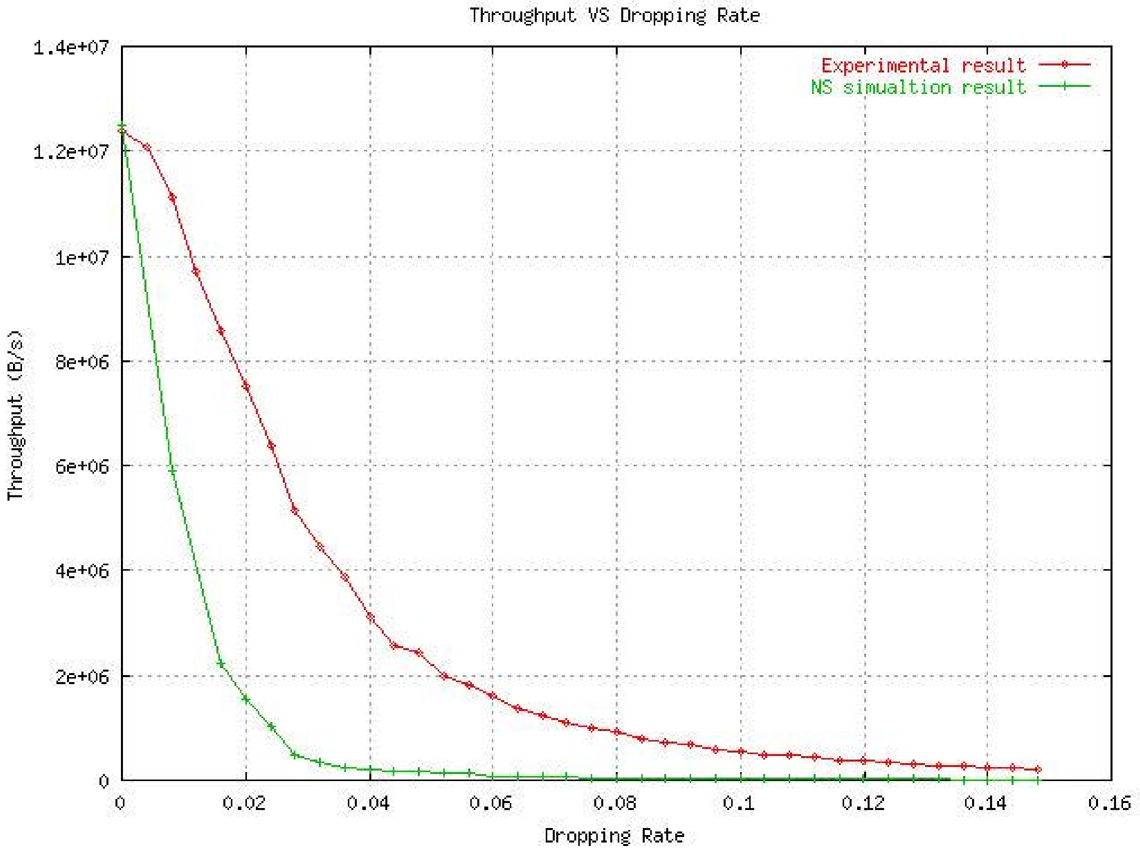

Figure 5.1 Comparison of random dropping and NS

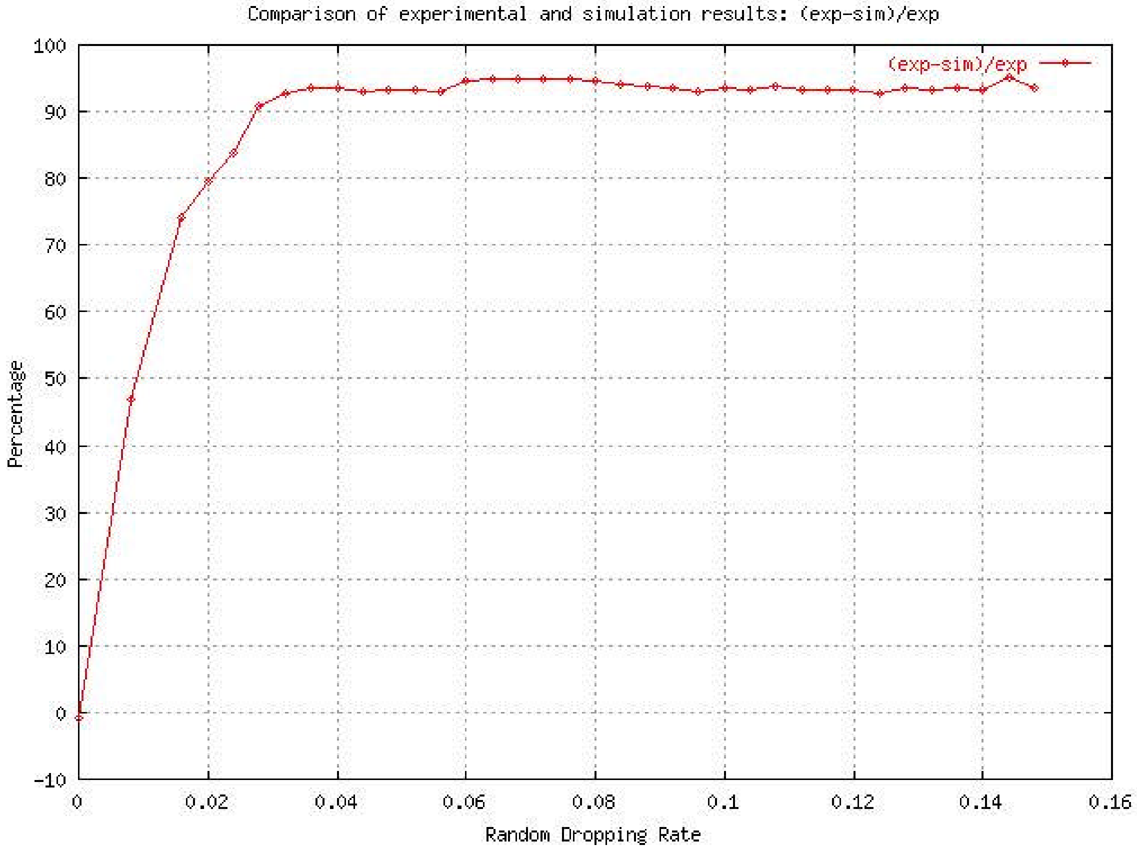

Figure 5.2 Comparison of experimental and simulation results in percentage. (exp -sim)/exp .

Chapter 6 Future work

For the limit of time, we can not do more stuff for packet delay and queue reordering. Here we conceptually talk about how to do packet delay.

Firstly, we define our own queue. Each item in this queue should have at least a timer and a structure—sk_buff. We also need to define some function to do "enqueue", "dequeue".

Secondly, on the Linux virtual DEVICE layer [1], we check the packet received (in file: /usr/src/linuc/net/core/dev.c function: int netif_rx(struct sk_buff *skb);). If the packet if not for the target connection, we just let it go normal. Otherwise, set the timer and put this packet to the queue we defined just now and process the next packet. When the timer in our queue goes off, it triggers some function to dequeue the packet and process it as what netif_rx() originally does (Note: don't call netif_rx directly since we have changed netif_rx()). Due to the shortage of time, we will leave the detail work for someone who is interested in this topics.

Chapter 7 Conclusion

In this document, we talk about the Linux routing environment, proc file system manipulation, experiment tools, performance before and after packet dropper is applied on the router. We also compare the experimental and simulation result. The experiment result shows much better performance than the simulation result. It also shows that the regular dropping has much less impact on performance than the random dropping at a given dropping rate.

Chapter 8 References

8.1 Documents

[1] G. Herrin, "Linux IP Networking, *a guide to the Implementation and Modification of the Linux Protocol Stack,"*, **TR 00-04**, *http://www.cs.unh.edu/cnrg/gherrin*

[2] E. Mouw, "Linux Kernel Procfs Guide," Delft University of Technology, Delft, The Netherlands. http://kernelnewbies.org/documents/kdoc/procfs-guide/lkprocfsguide.html

[3] R. Russell, "CS820 class note, CS Dept UNH",

8.2 Internet Sites

Linux Document Project	*http://www.linuxdoc.org*
Linux Kernel Project	*http://www.kernel.org*
Linux Router Project	*http://www.linuxrouter.org*
Red Hat Software	*http://www.redhat.com*
Request for comment	*http://www.rfc-editor.org/isi.html*

8.3 Books

A. Tanenbaum, "**Computer Networks"**, Prentice-Hall Inc., Upper Saddle River, NJ, 1996

W. Stalling, "High Speed Networks", Prentice-Hall Inc., Upper Saddle River, NJ, 1998

A. Rubini, "Linux Device Drivers", O'Reilly & Associates, Inc., Sebastopol, CA, 1998

M. Beck, "Linux Kernel Internals", Addison-Wesley, Harlow, England, 1997

Bovet & Desati, "Understanding Linux Kernel", O'Reilly & Associates, Inc., Sebastopol, CA, 2001

W.R Steven, "UNIX Network Programming", Vol.1 (2nd Ed.) Prentice-Hall Inc., Upper Saddle River, NJ, 1998