Traffic Control Manual For Lab1

For these guys who want to understand this manual very well, please refer to:
http://www.tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/

This manual has three parts. In the first part, it shows you how to emulate the delay, packet loss, duplication, re-ordering in our lab network with NetEM(Network Emulator), The second part shows you how to limit bandwidth with TBF(Token Bucket Filter). The third part will show you how to combine NetEm and TBF together and prove that the combination works.

All these “shows” will be presented by examples. Its environment is exactly the one we have set up in the virtual machine where the ubuntu1 and the FreeBSD are connected by the ubuntu2(router).

Except specified, all the command (the **bold** characters) will be input on the terminal of ubuntu2(router).

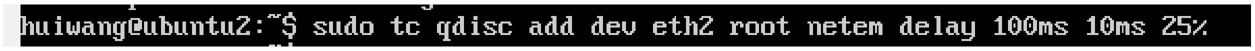
Part 1 Emulating the delay, packet loss,etc with NetEM

1. Delay

sudo tc qdisc add dev eth2 root netem delay 100ms 10ms 25%

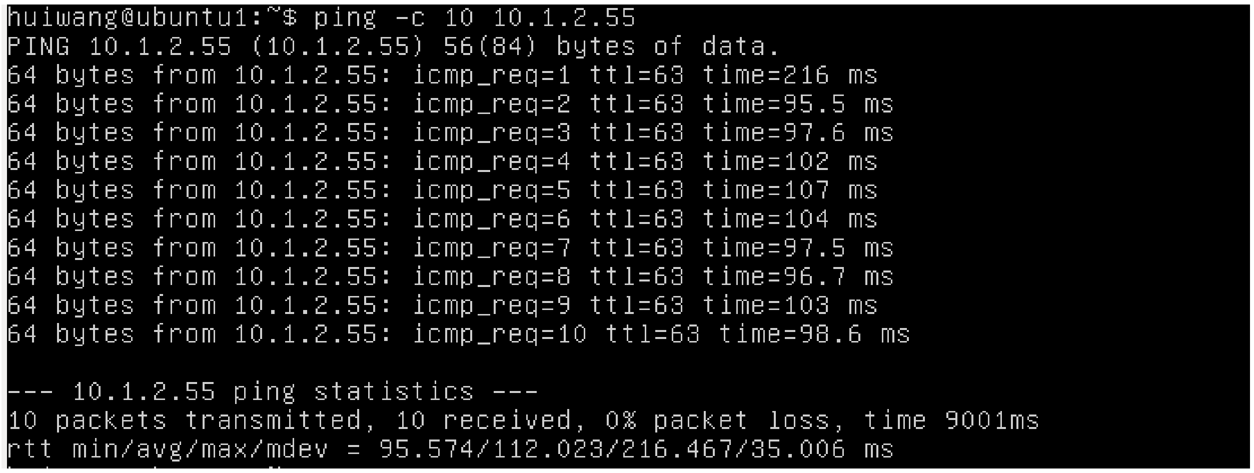
This causes the added delay to be 100ms ± 10ms with the next random element depending 25% on the last one. This isn't true statistical correlation, but an approximation.

I test it in my ubuntu2:



```
huiwang@ubuntu2:~$ sudo tc qdisc add dev eth2 root netem delay 100ms 10ms 25%
```

Then I use ping to prove that it works:



```
huiwang@ubuntu1:~$ ping -c 10 10.1.2.55
PING 10.1.2.55 (10.1.2.55) 56(84) bytes of data.
64 bytes from 10.1.2.55: icmp_req=1 ttl=63 time=216 ms
64 bytes from 10.1.2.55: icmp_req=2 ttl=63 time=95.5 ms
64 bytes from 10.1.2.55: icmp_req=3 ttl=63 time=97.6 ms
64 bytes from 10.1.2.55: icmp_req=4 ttl=63 time=102 ms
64 bytes from 10.1.2.55: icmp_req=5 ttl=63 time=107 ms
64 bytes from 10.1.2.55: icmp_req=6 ttl=63 time=104 ms
64 bytes from 10.1.2.55: icmp_req=7 ttl=63 time=97.5 ms
64 bytes from 10.1.2.55: icmp_req=8 ttl=63 time=96.7 ms
64 bytes from 10.1.2.55: icmp_req=9 ttl=63 time=103 ms
64 bytes from 10.1.2.55: icmp_req=10 ttl=63 time=98.6 ms

--- 10.1.2.55 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9001ms
rtt min/avg/max/mdev = 95.574/112.023/216.467/35.006 ms
```

we found that all the rtt is between 100ms ± 10ms except the first one which is because the arp request and response packets also take a rtt. (In the following part of manual, You can also verify if the tc rule works in this way by yourself.)

Typically, the delay in a network is not uniform. It is more common to use a something like a normal distribution to describe the variation in delay. The netem discipline can take a table to specify a non-uniform distribution:
sudo tc qdisc change dev eth2 root netem delay 100ms 20ms distribution normal

This command will replace the previous tc rule and make the distribution of delay is normal distribution between the range of 100ms ± 20ms . The actual tables (normal, pareto, paretonormal) are generated as part of the iproute2 compilation and placed in /usr/lib/tc; so it is possible with some effort to make your own distribution based on experimental data.

2. Packet loss

Random packet loss is specified in the 'tc' command in percent. The smallest possible non-zero value is: 232 = 0.0000000232%

sudo tc qdisc change dev eth2 root netem loss 5%

This causes 1/10th of a percent (i.e 1 out of 1000) packets to be randomly dropped.
An optional correlation may also be added. This causes the random number generator to be less random and can be used to emulate packet burst losses.
sudo tc qdisc change dev eth2 root netem loss 5% 25%

This will cause 5% of packets to be lost, and each successive probability depends by a quarter on the last one.
Prob(n) = .25 * Prob(n-1) + .75 * Random

3. Packet corruption

Random noise can be emulated with the corrupt option. This introduces a single bit error at a random offset in the packet.
sudo tc qdisc change dev eth2 root netem corrupt 5%

4.Packet Re-ordering

There are two different ways to specify reordering. The first method gap uses a fixed sequence and reorders every Nth packet. A simple usage of this is:
sudo tc qdisc change dev eth2 root netem gap 5 delay 10ms

This causes every 5th (10th, 15th, ...) packet to go to be sent immediately and every other packet to be delayed by 10ms. This is predictable and useful for base protocol testing like reassembly.

The second form reorder of re-ordering is more like real life. It causes a certain percentage of the packets to get mis-ordered.
sudo tc qdisc change dev eth2 root netem delay 10ms reorder 25% 50%

In this example, 25% of packets (with a correlation of 50%) will get sent immediately, others will be delayed by 10ms.

Newer versions of netem will also re-order packets if the random delay values are out of order. The following will cause some reordering:
sudo tc qdisc change dev eth2 root netem delay 100ms 75ms

If the first packet gets a random delay of 100ms (100ms base - 0ms jitter) and the second packet is sent 1ms later and gets a delay of 50ms (100ms base - 50ms jitter); the second packet will be sent first. This is because the queue discipline tfifo inside netem, keeps packets in order by time to send.

Part 2 Limit Bandwidth with TBF

There is no rate control built-in to the netem discipline, instead use one of the other disciplines that does do ratecontrol. In this example, we use Token Bucket Filter (TBF) to limit output.

First, we need to clean up the setting we made in the first part.
sudo tc qdisc del dev eth2 root

This will delete the tc rule we made in the first part of this manual.

Then,
sudo tc qdisc add dev eth2 root tbf rate 256kbit burst 1600 limit 3000

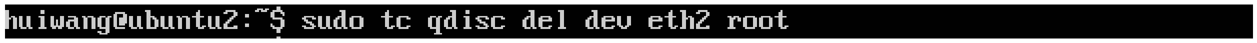
This will limit the one-direction bandwidth from the ubuntu1 to FreeBSD to 256 kbps, just replace 256kbit with other value without changing other parts of this command if you want to specify a different bandwidth. If you want to limit the bandwidth from FreeBSD to ubuntu1 at the same time, you should (just change the interface from eth2 to eth1):
sudo tc qdisc add dev eth1 root tbf rate 256kbit burst 1600 limit 3000

Part 3 Combination of NetEM and TBF

First, we need to clean up the queue discipline we made in the first part.
sudo tc qdisc del dev eth2 root

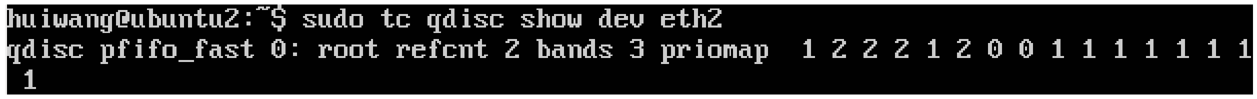
Then**,**
sudo tc qdisc add dev eth2 root handle 1: tbf rate 256kbit buffer 1600 limit 3000
sudo tc qdisc add dev eth2 parent 1:1 handle 10: netem delay 100ms

Below is the proof that the combination of NetEM and TBF works:
First, I clean up the tc rule we made in the first part.



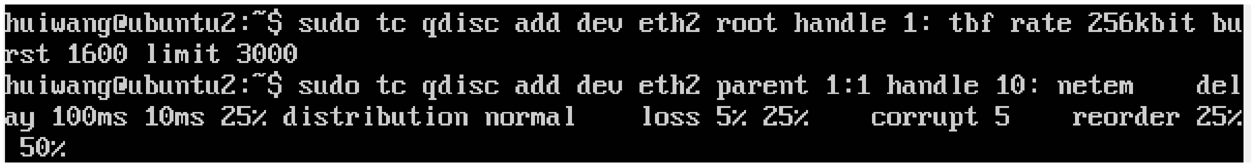
```
huiwang@ubuntu2:~$ sudo tc qdisc del dev eth2 root
```

and use
sudo tc qdisc show dev eth2
to check if we delete it successfully. If we do, we will get something like these:

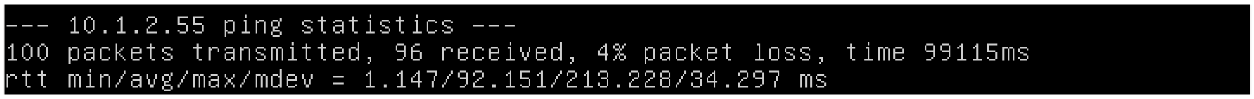


```
huiwang@ubuntu2:~$ sudo tc qdisc show dev eth2
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1
 1
```

Then, emulate the network between ubuntu1 and FreeBSD by inputting two commands in the terminal of ubuntu2:

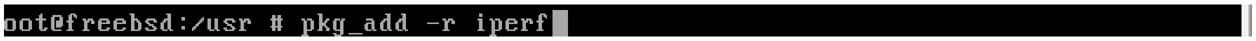

```
huiwang@ubuntu2:~$ sudo tc qdisc add dev eth2 root handle 1: tbf rate 256kbit bu
rst 1600 limit 3000
huiwang@ubuntu2:~$ sudo tc qdisc add dev eth2 parent 1:1 handle 10: netem    del
ay 100ms 10ms 25% distribution normal    loss 5% 25%    corrupt 5    reorder 25%
 50%
```

Final, I use ping command and iperf to make sure this combination of NetEM and TBF works:
 1. ping the em1(interface 1) of FreeBSD from the terminal of ubuntu1 for 1000 times):
ping -c 100 10.1.2.55
The ping result is:

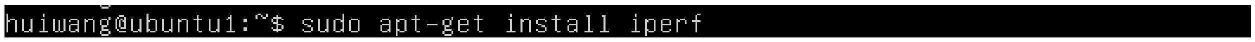

```
--- 10.1.2.55 ping statistics ---
100 packets transmitted, 96 received, 4% packet loss, time 99115ms
rtt min/avg/max/mdev = 1.147/92.151/213.228/34.297 ms
```

which is the result from the mixed effect of delay, packet loss corruption and reordering.

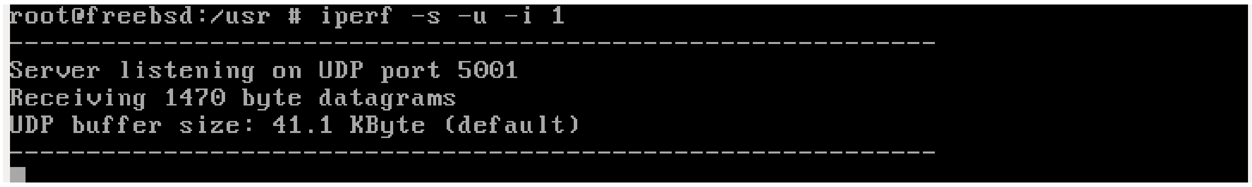
2. user iperf to test the bandwidth between ubuntu1 and FreeBSD:
 (1) install iperf in FreeBSD :


```
oot@freebsd:/usr # pkg_add -r iperf
```

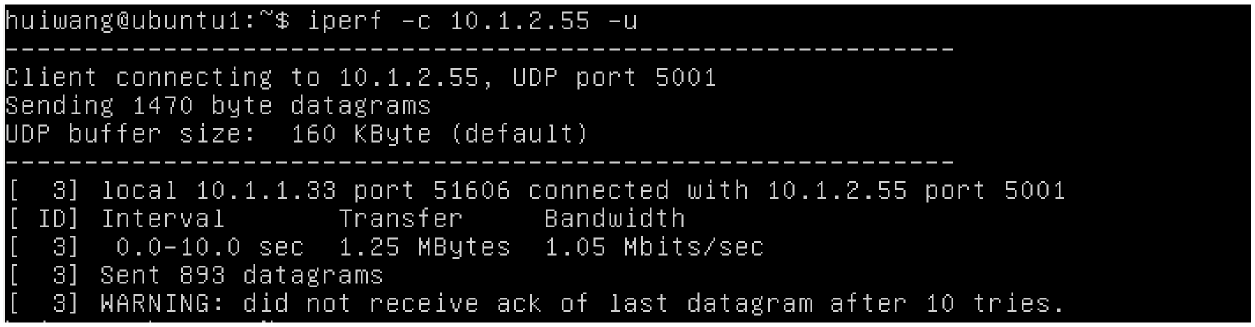
(2). install iperf in ubuntu1:


```
huiwang@ubuntu1:~$ sudo apt-get install iperf
```

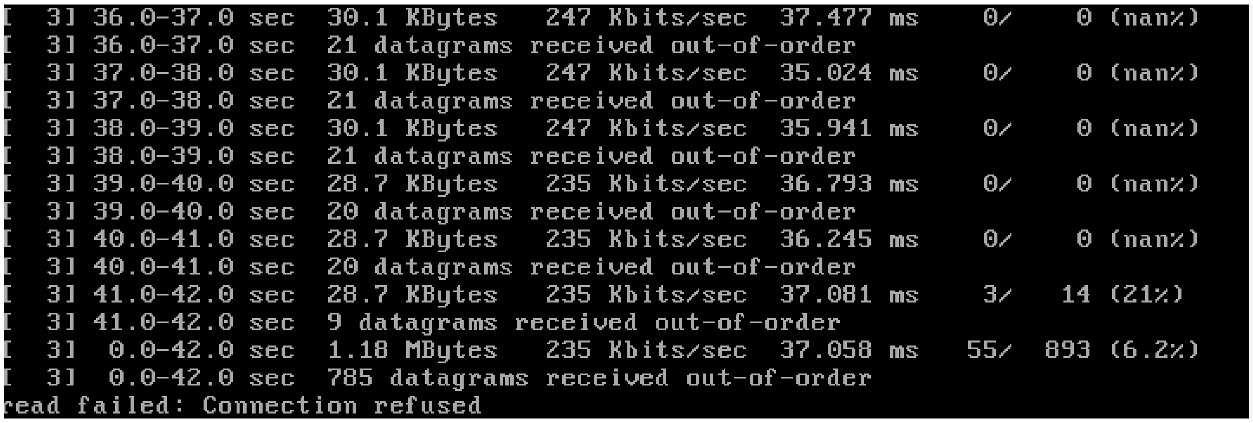
(3). run iperf as a server in FreeBSD:


```
root@freebsd:/usr # iperf -s -u -i 1
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 41.1 KByte (default)
------------------------------------------------------------
```

(4). run iperf as a client in ubuntu1:


```
huiwang@ubuntu1:~$ iperf -c 10.1.2.55 -u
------------------------------------------------------------
Client connecting to 10.1.2.55, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 10.1.1.33 port 51606 connected with 10.1.2.55 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
```

In the FreeBSD, we get the result:



```
[  3] 36.0-37.0 sec  30.1 KBytes   247 Kbits/sec  37.477 ms    0/    0 (nan%)
[  3] 36.0-37.0 sec  21 datagrams received out-of-order
[  3] 37.0-38.0 sec  30.1 KBytes   247 Kbits/sec  35.024 ms    0/    0 (nan%)
[  3] 37.0-38.0 sec  21 datagrams received out-of-order
[  3] 38.0-39.0 sec  30.1 KBytes   247 Kbits/sec  35.941 ms    0/    0 (nan%)
[  3] 38.0-39.0 sec  21 datagrams received out-of-order
[  3] 39.0-40.0 sec  28.7 KBytes   235 Kbits/sec  36.793 ms    0/    0 (nan%)
[  3] 39.0-40.0 sec  20 datagrams received out-of-order
[  3] 40.0-41.0 sec  28.7 KBytes   235 Kbits/sec  36.245 ms    0/    0 (nan%)
[  3] 40.0-41.0 sec  20 datagrams received out-of-order
[  3] 41.0-42.0 sec  28.7 KBytes   235 Kbits/sec  37.081 ms    3/   14 (21%)
[  3] 41.0-42.0 sec  9 datagrams received out-of-order
[  3]  0.0-42.0 sec  1.18 MBytes   235 Kbits/sec  37.058 ms   55/  893 (6.2%)
[  3]  0.0-42.0 sec  785 datagrams received out-of-order
read failed: Connection refused
```

Now, we show that the bandwidth from ubuntu1 to FreeBSD is limited to 250kbit/s(you can also see other characters like Jitter and packer loss from the output in the terminal of FreeBSD.

Thus we proved that the combination of NetEM and TBF works.

--END OF SHOWS--
--- THANKS---

Acknowledge

A big part of this manual comes from the Internet and I compiled them and made some change so that it is specially for our lab.

Reference:

1. https://calomel.org/network_loss_emulation.html
2. http://mytestbed.net/projects/omf/wiki/NetEM_examples_of_rules
3. http://www.linuxfoundation.org/collaborate/workgroups/networking/netem