Dropping Packets in Ubuntu Linux using tc and iptables

By Steven Gordon on Tue, 18/01/2011 - 8:13pm

There are two simple ways to randomly drop packets on a Linux computer: using `tc`, the program dedicated for controlling traffic; and using `iptables`, the built-in firewall. A good description of using `tc` to drop packets (as well as other features) is available via the Linux Foundation [3] site. There is a online manual [4] for `iptables`; the `man` pages also describe the statistic module. Here I will explain the use of both.

1. Dropping Packets with tc

First, lets use `tc`. `tc` controls the transmit queues of your kernel. Normally when applications on your computer generate data to send, the data is passed to your kernel (via TCP and IP) for transmission on the network interface. The packets are transmitted in a first-in-first-out (FIFO) order. `tc` allows you to change the queuing mechanisms (e.g. giving priority to specific type of packets), as well as emulate links by delaying and dropping packets. Here we will use `tc` to drop packets. Because `tc` controls the transmit queues, we use it on a source computer (normally `tc` doesn't impact on what is received by your computer, but there are exceptions).

Assume we want to transfer data from A to B. Computer A transmits on interface `eth0` (and in the output below, has IP address 192.168.1.116; B is 192.168.1.5). To randomly drop packets sent by computer A, on computer A run the command:

```
$ sudo tc qdisc add dev eth0 root netem loss 3%
```

netem is a special type of queuing discipline used for emulating networks. The above command tells the Linux kernel to drop on average 3% of the packets in the transmit queue. You can use different values of loss (e.g. 10%).

When using `tc` you can show the current queue disciplines using:

```
$ sudo tc qdisc show dev eth0
qdisc netem 8001: root refcnt 2 limit 1000 loss 3%
```

To show that it works, lets run an `iperf` UDP test. On computer B (the computer where `tc` is NOT used) run:

```
$ iperf -s -u
```

And now on computer A run a UDP test, sending at 1Mb/s for 60 seconds:

```
$ iperf -c 192.168.1.5 -u -t 60
------------------------------------------------------------
Client connecting to 192.168.1.5, UDP port 5001
```

```
Sending 1470 byte datagrams
UDP buffer size:   110 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.116 port 57240 connected with 192.168.1.5 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec
[  3] Sent 5351 datagrams
[  3] Server Report:
[  3]  0.0-60.0 sec  7.26 MBytes  1.02 Mbits/sec  0.348 ms  170/ 5351 (3.2%)
```

Out of the 5351 packets sent by A, 170 were dropped (not received by by). This is 3.2%. Of course `tc` selects packets to drop randomly, and so we do not always get exactly 3% drops over 60 seconds.

To delete the above queue discipline use the *del* command instead of *add*:

```
$ sudo tc qdisc del dev eth0 root netem loss 3%
$ sudo tc qdisc show dev eth0
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
```

(The above output shows the default FIFO queue that the kernel uses).

A potential problem with dropping packets in the kernel at the sender (i.e. using `tc`), is that the TCP implementation in the kernel may be notified of the drop and respond differently to when a drop occurs in the network or receiver. That is, TCP is smart enough to recognise this packet loss is not due to congestion, and responds accordingly. To avoid this (because often we want to see how TCP performs in response to congestion) we have two options:

1. Use a third computer, C, running `tc` on that computer instead of the source A. The topology would be A--C--B. If computer C drops packets, then TCP at source A will definitely respond as if congestion increased.
2. Drop packets at the destination, B. However as `tc` operates on *transmitted* packets, we cannot use it to drop packets at B.

If we don't have a third computer available, then we can use `iptables`, the default firewall software in Linux, to drop packets at the destination B. `iptables`, and firewalls in general, can be very complex. Here I will not try to explain them, but rather illustrate how they can be used to achieve our goal of dropping packets at B.

Update (2013-01-30): On Ubuntu 12.04 LTS (and possibly other versions) there is a bug [5] that means `iptables` cannot use the `statistic` module in the following commands. If the iptables command below that uses the `-m statistic` option doesn't work, then you need to patch iptables to workaround this bug. Here is how to apply the patch (my instructions are following directly from one of the comments [6] in the bug report):

```
$ sudo apt-get install quilt
$ sudo apt-get build-dep iptables
$ sudo apt-get source iptables
$ wget https://launchpadlibrarian.net/104349144/xtables-lm-noasneeded.patch
```

```
$ cd iptables-1.4.12
$ sudo quilt import ../xtables-lm-noasneeded.patch
$ sudo quilt push -a
$ sudo dpkg-buildpackage -b
$ cd ..
$ sudo dpkg -i iptables_1.4.12-1ubuntu4_amd64.deb
$ echo "iptables hold" | sudo dpkg --set-selections
```

2. Dropping Packets with iptables

`iptables` allows you to create rules that specify how packets coming into your computer and going out of your computer are treated (and for routers, also forwarded by the router). The rules for packets coming in are in the INPUT *chain*, packets going out are OUTPUT, and packets forwarded are in the FORWARD chain. We will only use the INPUT chain.

The rules can filter packets based on common packet identifiers (IP addresses, ports, protocol numbers) as well as other matching criteria. We will use a special *statistic* matching module. For each packet that matches the filter, some action is applied (e.g. DROP the packet, ACCEPT the packet, or some more complex operation). We want to DROP packets, in particular a specified percentage of packets.

On computer B (the destination), to view the current set of rules:

```
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

There are no rules in either of the three chains. Note that the default policy (if a packet does not match any rule) is to ACCEPT packets.

Now to add a rule to the INPUT chain to drop 3% of incoming packets on computer B:

```
$ sudo iptables -A INPUT -m statistic --mode random --probability 0.03 -j DROP
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere            statistic mode random pr

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

If the above did not work, then check the update from 2013-01-30 higher up in this page.

To demonstrate the packet dropping, run another `iperf` UDP test on the source A:

```
$ iperf -c 192.168.1.5 -u -t 60
------------------------------------------------------------
Client connecting to 192.168.1.5, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:   110 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.116 port 33427 connected with 192.168.1.5 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec
[  3] Sent 5351 datagrams
[  3] Server Report:
[  3]  0.0-60.0 sec  7.30 MBytes  1.02 Mbits/sec  0.343 ms  146/ 5351 (2.7%)
```

About 3% of the sent packets were dropped.

Returning to computer B, to delete a rule you can use the -D option:

```
$ sudo iptables -D INPUT -m statistic --mode random --probability 0.03 -j DROP
```

(or you can refer to rules by number, e.g. `iptables -D INPUT 1` to delete rule 1 from the INPUT chain).

Finally, the above rule dropped a percentage of packets. Alternatively we can specify to drop every *n* packets, starting from packet *p. And we can combine with the standarding filtering mechanisms of firewalls to only drop packets belong to a particular source/destination pair or application.*

Here is the output of an `iperf` test when no rules are configured at the destination B. There are no packets dropped. The `-P` option tells the client to create 2 parallel connections (i.e. using different source port numbers, 45045 and 46770 in this case). There is one packet lost.

```
$ iperf -c 192.168.1.5 -u -t 60 -P 2
------------------------------------------------------------
Client connecting to 192.168.1.5, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:   110 KByte (default)
------------------------------------------------------------
[  3] local 192.168.1.116 port 45045 connected with 192.168.1.5 port 5001
[  4] local 192.168.1.116 port 46770 connected with 192.168.1.5 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec
[  3] Sent 5351 datagrams
[  4]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec
[  4] Sent 5351 datagrams
[SUM]  0.0-60.0 sec  15.0 MBytes  2.10 Mbits/sec
[  3] Server Report:
[  3]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec  0.349 ms    0/ 5351 (0%)
[  4] Server Report:
[  4]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec  0.345 ms    1/ 5351 (0.019%)
```

Now start the `iperf` test again, quickly note the source port numbers and on the destination B

add the following rule:

```
$ sudo iptables -A INPUT -p udp --sport 39912 -m statistic --mode nth --every 200 --pa
```

This rule should drop packet 100, 300, 500, 700, ... for only one of the connections (with source port 39912). The result at the source is that for only one of the UDP connections there are 26 packets lost (note I added the firewall rule slightly after starting `iperf` client; thats way 27 packets are not dropped).

```
$ iperf -c 192.168.1.5 -u -t 60 -P 2
------------------------------------------------------------
Client connecting to 192.168.1.5, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:   110 KByte (default)
------------------------------------------------------------
[  4] local 192.168.1.116 port 39912 connected with 192.168.1.5 port 5001
[  3] local 192.168.1.116 port 54717 connected with 192.168.1.5 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec
[  4] Sent 5351 datagrams
[  3]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec
[  3] Sent 5351 datagrams
[SUM]  0.0-60.0 sec  15.0 MBytes  2.10 Mbits/sec
[  4] Server Report:
[  4]  0.0-60.0 sec  7.47 MBytes  1.04 Mbits/sec  0.376 ms   26/ 5351 (0.49%)
[  3] Server Report:
[  3]  0.0-60.0 sec  7.50 MBytes  1.05 Mbits/sec  0.335 ms    1/ 5351 (0.019%)
```

So now you can drop packets for specific connections, and by comparing the performance of the different connections investigate the impact of packet drops (especially on TCP).

Interest: Ubuntu Linux [7]
iperf [8]
iptables [9]
tc [10]
Topic: Networking [11]
Content: Howto [12]

Source URL: http://sandilands.info/sgordon/dropping-packets-in-ubuntu-linux-using-tc-and-iptables

Links:
[1] http://sandilands.info/sgordon/dropping-packets-in-ubuntu-linux-using-tc-and-iptables
[2] http://sandilands.info/sgordon/user/2
[3] http://www.linuxfoundation.org/collaborate/workgroups/networking/netem
[4] http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO-7.html
[5] https://bugs.launchpad.net/ubuntu/+source/iptables/+bug/982961
[6] https://bugs.launchpad.net/ubuntu/+source/iptables/+bug/982961/comments/6
[7] http://sandilands.info/sgordon/taxonomy/term/302
[8] http://sandilands.info/sgordon/taxonomy/term/306
[9] http://sandilands.info/sgordon/taxonomy/term/311
[10] http://sandilands.info/sgordon/taxonomy/term/312
[11] http://sandilands.info/sgordon/taxonomy/term/7
[12] http://sandilands.info/sgordon/taxonomy/term/212