

# Controlling Traffic on Linux

## Fun with “tc” command

Thanks to Linux Foundation

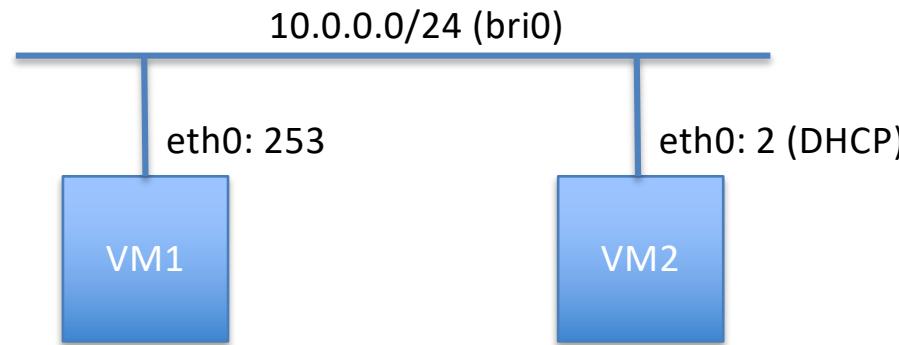
# What we do here?

- You may want “Internet-like” network for evaluating your network software.
- How? Several options cost something.
  - Buy AWS? Paying money.
  - Run on Simulator? Exporting software to simulator.
  - Do something on your LAN? You may disturb others.
- Well....

# “tc” comamnd with netem

- “Netem provides” [Network Emulation](#) functionality for testing protocols by emulating the properties of wide area networks. The current version emulates variable delay, loss, duplication and re-ordering.
- Netem is controlled by the command line tool 'tc' which is part of the [iproute2](#) package of tools. The tc command uses shared libraries and data files in the /usr/lib/tc directory.

# Let's make a network



- Tools: **tc**, ping, iperf, wireshark, tcpdump

```
kotaro@server1:~$ ping -c 10 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.364 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.763 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.782 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.642 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.639 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.610 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.672 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.775 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.678 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.666 ms
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9129ms
rtt min/avg/max/mdev = 0.364/0.659/0.782/0.114 ms
kotaro@server1:~$
```

```
kotaro@server1:~$ iperf -c 10.0.0.2 -i 1
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 3.62 MByte (default)
[ 3] local 10.0.0.253 port 57120 connected with 10.0.0.2
[ ID] Interval Transfer Bandwidth
[ 3] 0.0- 1.0 sec 3.54 GBytes 30.4 Gbits/sec
[ 3] 1.0- 2.0 sec 3.94 GBytes 33.8 Gbits/sec
[ 3] 2.0- 3.0 sec 3.57 GBytes 30.7 Gbits/sec
[ 3] 3.0- 4.0 sec 4.39 GBytes 37.7 Gbits/sec
[ 3] 4.0- 5.0 sec 3.91 GBytes 33.6 Gbits/sec
[ 3] 5.0- 6.0 sec 3.93 GBytes 33.8 Gbits/sec
[ 3] 6.0- 7.0 sec 3.88 GBytes 33.3 Gbits/sec
[ 3] 7.0- 8.0 sec 3.74 GBytes 32.2 Gbits/sec
[ 3] 8.0- 9.0 sec 3.70 GBytes 31.8 Gbits/sec
[ 3] 9.0-10.0 sec 4.36 GBytes 37.5 Gbits/sec
[ 3] 0.0-10.0 sec 39.0 GBytes 33.5 Gbits/sec
kotaro@server1:~$
```

```
kotaro@server1:~$ iperf -c 10.0.0.2 -u -b 2G -i 1
Client connecting to 10.0.0.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 5.48 us (kalman adjus
UDP buffer size: 208 KByte (default)
[ 3] local 10.0.0.253 port 32922 connected with 10.0.0.2 port
[ ID] Interval Transfer Bandwidth
[ 3] 0.0- 1.0 sec 256 MBytes 2.15 Gbits/sec
[ 3] 1.0- 2.0 sec 236 MBytes 1.98 Gbits/sec
[ 3] 2.0- 3.0 sec 264 MBytes 2.21 Gbits/sec
[ 3] 3.0- 4.0 sec 268 MBytes 2.25 Gbits/sec
[ 3] 4.0- 5.0 sec 256 MBytes 2.15 Gbits/sec
[ 3] 5.0- 6.0 sec 256 MBytes 2.15 Gbits/sec
[ 3] 6.0- 7.0 sec 254 MBytes 2.13 Gbits/sec
[ 3] 7.0- 8.0 sec 258 MBytes 2.16 Gbits/sec
[ 3] 8.0- 9.0 sec 246 MBytes 2.06 Gbits/sec
[ 3] 0.0-10.0 sec 2.48 GBytes 2.13 Gbits/sec
[ 3] Sent 1814325 datagrams
[ 3] Server Report:
[ 3] 0.0-10.0 sec 2.48 GBytes 2.13 Gbits/sec 0.004 ms 8
```

# Applying delay to all outgoing packets on a NIC

- Execute the following command on VM1

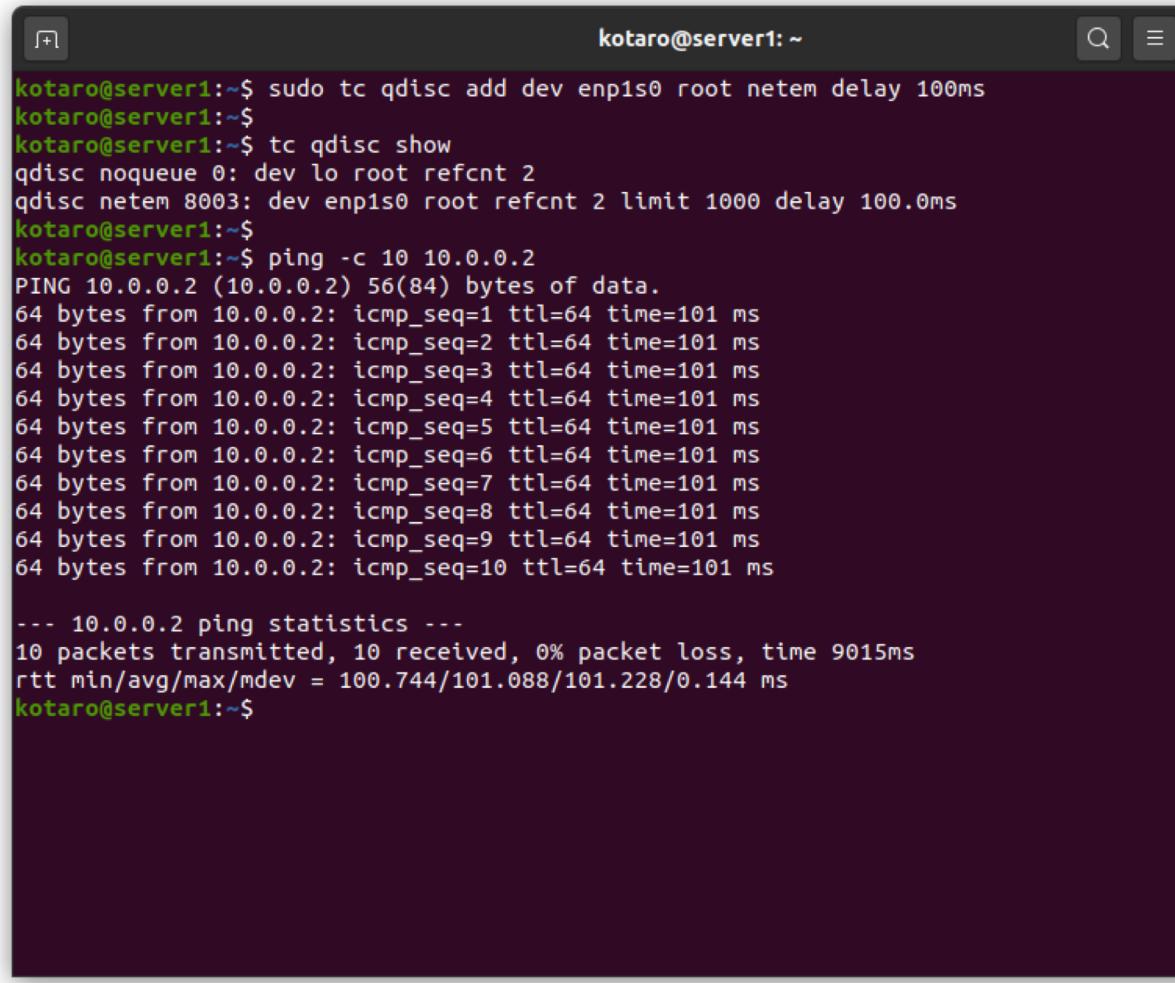
```
# tc qdisc add dev eth0 root netem delay 100ms
```

- Adding a queue
  - qdisc: Queue Discipline where packets to a NIC to go through
  - root: All 1 (one) for 16bits ID of queue
  - netem: Enable Network Emulation
  - delay 100 ms: Delay the outgoing packets by 100ms

# Showing the queue and seeing how it affects

```
# tc qdisc show
```

- Use ping command to see how the RTT is affected



A screenshot of a terminal window titled "kotaro@server1: ~". The window displays the following command-line session:

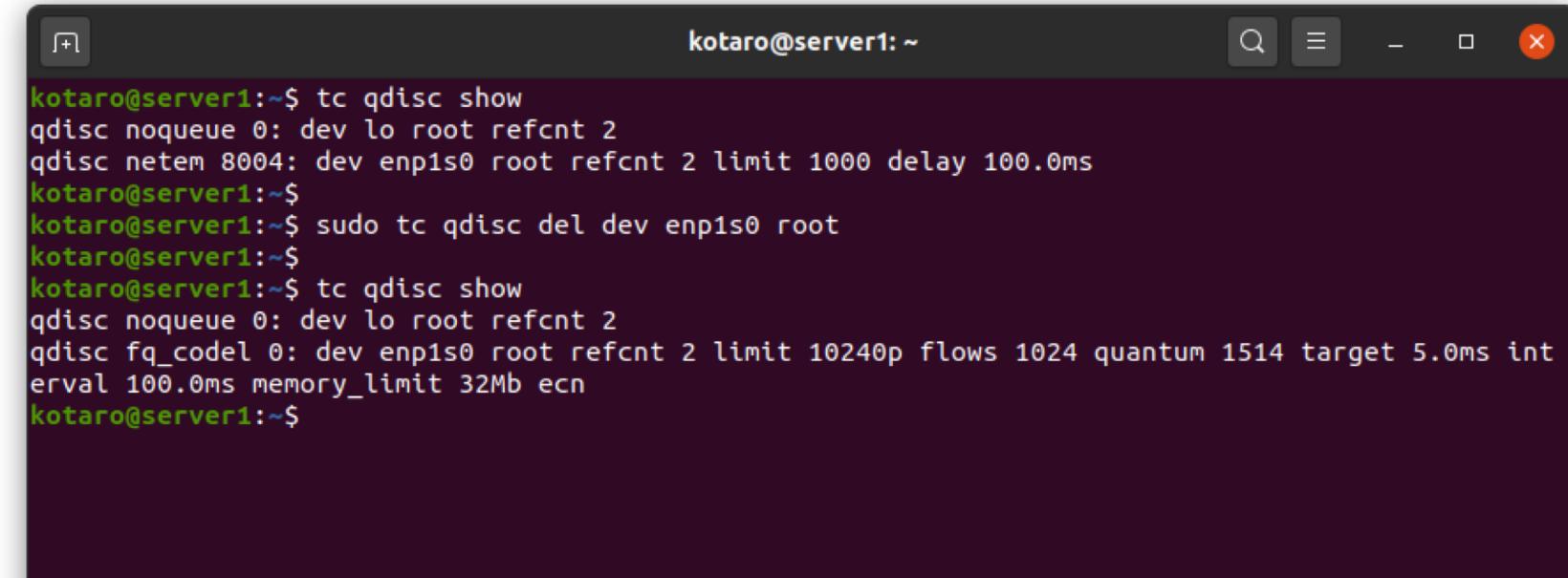
```
kotaro@server1:~$ sudo tc qdisc add dev enp1s0 root netem delay 100ms
kotaro@server1:~$ tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 8003: dev enp1s0 root refcnt 2 limit 1000 delay 100.0ms
kotaro@server1:~$ ping -c 10 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=101 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=101 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9015ms
rtt min/avg/max/mdev = 100.744/101.088/101.228/0.144 ms
kotaro@server1:~$
```

# Deleting the queue

```
# tc qdisc del dev eth0 root
```

- The additional delay is removed from the NIC.



A screenshot of a terminal window titled "kotaro@server1: ~". The terminal shows the following sequence of commands:

```
kotaro@server1:~$ tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc netem 8004: dev enp1s0 root refcnt 2 limit 1000 delay 100.0ms
kotaro@server1:~$ sudo tc qdisc del dev enp1s0 root
kotaro@server1:~$ tc qdisc show
qdisc noqueue 0: dev lo root refcnt 2
qdisc fq_codel 0: dev enp1s0 root refcnt 2 limit 10240p flows 1024 quantum 1514 target 5.0ms interval 100.0ms memory_limit 32Mb ecn
kotaro@server1:~$
```

# Emulating Random Delay

```
# tc qdisc add dev eth0 root netem delay 100ms 10ms
    – Randomly delaying outgoing packets by 100ms +/- 10ms
```

- You can also “change” the configuration instead of “del and add”

```
# tc qdisc change dev eth0 root netem delay 100ms 10ms 25%
    – Introducing correlation to delay
    – The next random element is depending 25% on the last one
```

```
# tc qdisc change dev eth0 root netem delay 100ms 20ms
distribution normal
    – Applying normal distribution to delay
    – 100ms: Average ( $\mu$ ), 20ms: Standard Deviation ( $\sigma$ )
```

# Emulating Random Loss

```
# tc qdisc change dev eth0 root netem loss 5%
```

- Randomly drop packets by 5% ( $P = 0.05$ )
- Check with ping command, too!

The image shows two terminal windows side-by-side. The left window, titled 'kotaro@server1: ~', displays the configuration of a network interface 'enp1s0' with 'netem' discipline and 5% loss. It then runs an 'iperf -c 10.0.0.2' client test, which connects to a server at 10.0.0.2 port 5001. The client shows a transfer rate of approximately 2.11 MByte/sec over 3 seconds. The right window, titled 'kotaro@server2: ~', shows the server configuration where it is listening on port 5001. It then runs an 'iperf -s' command, which lists four connections from the client. The bandwidth for these connections is shown as 21.7 Gbits/sec, 1.24 Gbits/sec, 2.79 Mbytes/sec, and 3.81 Mbytes/sec respectively. Both terminals have a dark background with light-colored text.

```
kotaro@server1:~$ sudo tc qdisc change dev enp1s0 root netem loss 5%
kotaro@server1:~$ iperf -c 10.0.0.2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 3.74 MByte (default)
[ 3] local 10.0.0.253 port 57464 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer     Bandwidth
[ 3]  0.0-10.0 sec  25.3 GBytes   21.7 Gbits/sec
kotaro@server1:~
kotaro@server1:~$ sudo tc qdisc change dev enp1s0 root netem loss 10%
kotaro@server1:~$ iperf -c 10.0.0.2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 2.11 MByte (default)
[ 3] local 10.0.0.253 port 57468 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer     Bandwidth
[ 3]  0.0-10.0 sec  1.45 GBytes   1.25 Gbits/sec
kotaro@server1:~
kotaro@server1:~$ sudo tc qdisc change dev enp1s0 root netem loss 20%
kotaro@server1:~$ iperf -c 10.0.0.2
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 144 KByte (default)
[ 3] local 10.0.0.253 port 57472 connected with 10.0.0.2 port 5001
[ ID] Interval      Transfer     Bandwidth
[ 3]  0.0-10.5 sec  4.75 MBytes   3.81 Mbits/sec

kotaro@server2:~$ iperf -s
-----
Server listening on TCP port 5001
TCP window size: 128 KByte (default)
[ 4] local 10.0.0.2 port 5001 connected with 10.0.0.253 port 57464
[ ID] Interval      Transfer     Bandwidth
[ 4]  0.0-10.0 sec  25.3 GBytes   21.7 Gbits/sec
[ 4] local 10.0.0.2 port 5001 connected with 10.0.0.253 port 57468
[ 4]  0.0-10.0 sec  1.45 GBytes   1.24 Gbits/sec
[ 4] local 10.0.0.2 port 5001 connected with 10.0.0.253 port 57472
[ 4]  0.0-14.3 sec  4.75 MBytes   2.79 Mbits/sec
```

# Emulating Bursty Loss?

```
# tc qdisc change dev eth0 root netem loss 0.3% 25%
```

- Randomly drop packets 0.3% ( $P = 0.003$ )
- Each successive probability depends 25% on the last one (Bursty loss)

- This is a known technique, but not very effective.
- Refer the following article if interested.

**Article**

Definition of a general and intuitive loss model for packet networks and its implementation in the Linux kernel Netem

January 2012

Authors:

F Ludovici A Ordine

[Download citation](#) [Copy link](#)

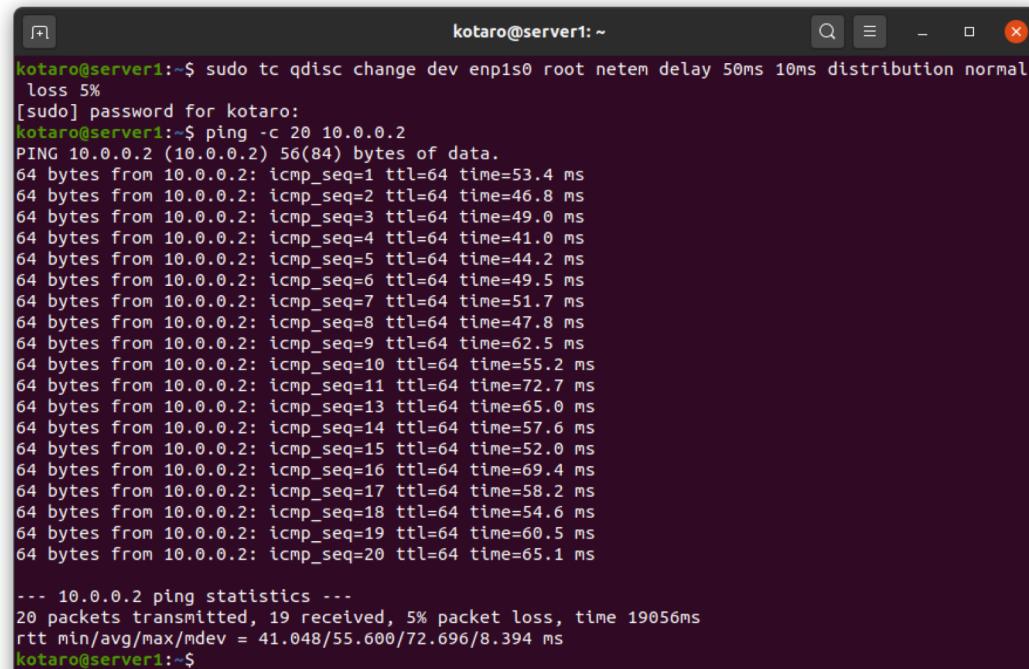
 Request full-text PDF

To read the full-text of this research, you can request a copy directly from the authors.

# Combining Delay and Loss

```
# tc qdisc change dev eth0 root netem delay 50ms 10ms  
distribution normal loss 5%
```

- 50ms Normal Distribution Delay and 5% Loss
- Networks can be more like Internet (?)



A screenshot of a terminal window titled "kotaro@server1: ~". The terminal shows the command "sudo tc qdisc change dev enp1s0 root netem delay 50ms 10ms distribution normal loss 5%" being run, followed by a password prompt "[sudo] password for kotaro:". Below this, a "ping -c 20 10.0.0.2" command is executed, displaying 20 ICMP echo requests sent to 10.0.0.2 with various round-trip times (rtt) ranging from 41.0 ms to 72.7 ms. The output concludes with ping statistics showing 20 packets transmitted, 19 received, 5% packet loss, and an average round-trip time (rtt) of 55.600 ms.

```
kotaro@server1:~$ sudo tc qdisc change dev enp1s0 root netem delay 50ms 10ms distribution normal  
loss 5%  
[sudo] password for kotaro:  
kotaro@server1:~$ ping -c 20 10.0.0.2  
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.  
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=53.4 ms  
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=46.8 ms  
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=49.0 ms  
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=41.0 ms  
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=44.2 ms  
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=49.5 ms  
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=51.7 ms  
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=47.8 ms  
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=62.5 ms  
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=55.2 ms  
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=72.7 ms  
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=65.0 ms  
64 bytes from 10.0.0.2: icmp_seq=14 ttl=64 time=57.6 ms  
64 bytes from 10.0.0.2: icmp_seq=15 ttl=64 time=52.0 ms  
64 bytes from 10.0.0.2: icmp_seq=16 ttl=64 time=69.4 ms  
64 bytes from 10.0.0.2: icmp_seq=17 ttl=64 time=58.2 ms  
64 bytes from 10.0.0.2: icmp_seq=18 ttl=64 time=54.6 ms  
64 bytes from 10.0.0.2: icmp_seq=19 ttl=64 time=60.5 ms  
64 bytes from 10.0.0.2: icmp_seq=20 ttl=64 time=65.1 ms  
  
--- 10.0.0.2 ping statistics ---  
20 packets transmitted, 19 received, 5% packet loss, time 19056ms  
rtt min/avg/max/mdev = 41.048/55.600/72.696/8.394 ms  
kotaro@server1:~$
```

# Bandwidth Throttling

```
# tc qdisc add dev eth0 root handle 1:0 tbf  
limit 20kb buffer 20kb rate 100Mbit
```

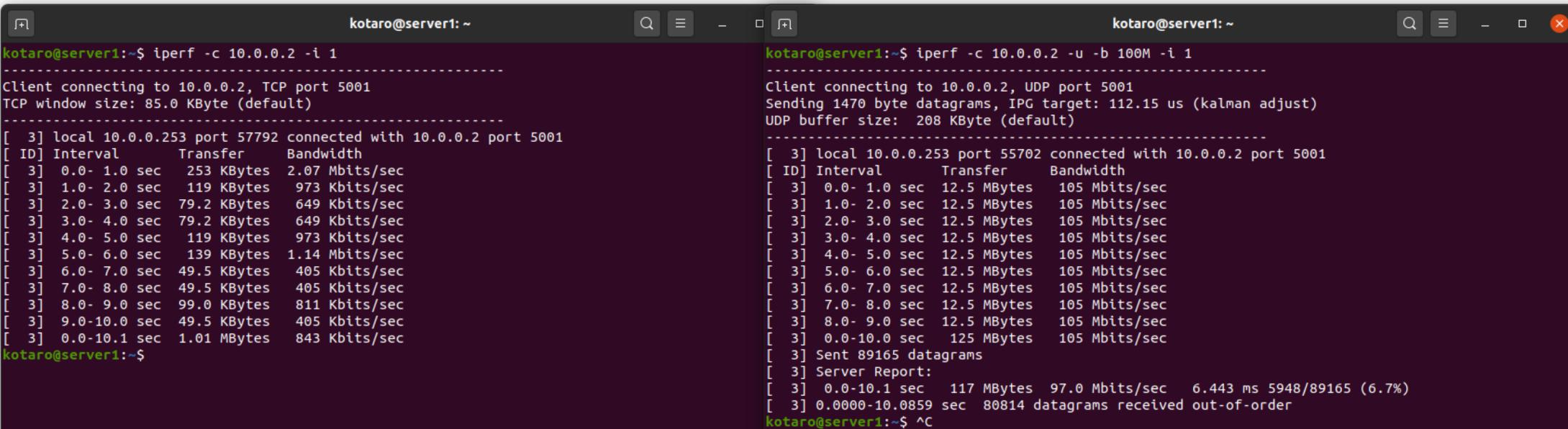
- tb~~f~~: The Token Bucket Filter is suited for slowing traffic down to a precisely configured rate. Scales well to large bandwidths.
- 100Mbit: 100Mbps

The screenshot shows two terminal windows on a Linux system. The left window displays the command to add a Token Bucket Filter (tb~~f~~) to the eth0 interface, setting a limit of 20kb, a buffer of 20kb, and a rate of 100Mbit. The right window shows the iperf test results between two hosts on port 5001, indicating a bandwidth of 105 Mbit/sec.

```
kotaro@server1:~$ sudo tc qdisc add dev enp1s0 root handle 1:0 tbf limit 20kb buffer 20kb rate 100Mbit  
kotaro@server1:~$ iperf -c 10.0.0.2 -u -b 100M -i 1  
-----  
Client connecting to 10.0.0.2, UDP port 5001  
Sending 1470 byte datagrams, IPG target: 112.15 us (kalman adjust)  
UDP buffer size: 208 KByte (default)  
-----  
[ 3] local 10.0.0.253 port 41379 connected with 10.0.0.2 port 5001  
[ ID] Interval Transfer Bandwidth  
[ 3] 0.0- 1.0 sec 11.6 MBytes 97.5 Mbits/sec  
[ 3] 1.0- 2.0 sec 11.4 MBytes 95.4 Mbits/sec  
[ 3] 2.0- 3.0 sec 11.4 MBytes 95.4 Mbits/sec  
[ 3] 3.0- 4.0 sec 11.5 MBytes 96.5 Mbits/sec  
[ 3] 4.0- 5.0 sec 11.4 MBytes 95.4 Mbits/sec  
[ 3] 5.0- 6.0 sec 11.4 MBytes 95.4 Mbits/sec  
[ 3] 6.0- 7.0 sec 11.4 MBytes 95.4 Mbits/sec  
[ 3] 7.0- 8.0 sec 11.5 MBytes 96.5 Mbits/sec  
[ 3] 8.0- 9.0 sec 11.4 MBytes 95.4 Mbits/sec  
[ 3] 9.0-10.0 sec 11.4 MBytes 95.4 Mbits/sec  
[ 3] 0.0-10.0 sec 114 MBytes 95.8 Mbits/sec  
kotaro@server1:~$  
-----  
[ 3] local 10.0.0.253 port 41379 connected with 10.0.0.2 port 5001  
[ ID] Interval Transfer Bandwidth  
[ 3] 0.0- 1.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 1.0- 2.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 2.0- 3.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 3.0- 4.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 4.0- 5.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 5.0- 6.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 6.0- 7.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 7.0- 8.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 8.0- 9.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 0.0-10.0 sec 125 MBytes 105 Mbits/sec  
[ 3] Sent 89165 datagrams  
[ 3] Server Report:  
[ 3] 0.0-10.0 sec 116 MBytes 97.2 Mbits/sec 0.016 ms 6474/89165 (7.3%)  
kotaro@server1:~$
```

# Combining Bandwidth Throttling, Delay and Loss

```
# tc qdisc add dev eth0 root handle 1:0 tbm limit  
20kb buffer 20kb rate 100Mbit  
  
# tc qdisc add dev eth0 parent 1:0 delay 50ms 10ms  
distribution normal loss 5%  
• 100Mbps AND Apprx. 50ms Delayed AND 5% Loss
```



The image shows two terminal windows side-by-side, both titled "kotaro@server1:~". The left window displays the output of an TCP iperf test, while the right window displays the output of a UDP iperf test. Both tests were run from server1 to a host at 10.0.0.2.

**Left Terminal (TCP Test):**

```
kotaro@server1:~$ iperf -c 10.0.0.2 -i 1  
Client connecting to 10.0.0.2, TCP port 5001  
TCP window size: 85.0 KByte (default)  
[ 3] local 10.0.0.253 port 57792 connected with 10.0.0.2 port 5001  
[ ID] Interval Transfer Bandwidth  
[ 3] 0.0- 1.0 sec 253 KBytes 2.07 Mbits/sec  
[ 3] 1.0- 2.0 sec 119 KBytes 973 Kbytes/sec  
[ 3] 2.0- 3.0 sec 79.2 KBytes 649 Kbits/sec  
[ 3] 3.0- 4.0 sec 79.2 KBytes 649 Kbits/sec  
[ 3] 4.0- 5.0 sec 119 KBytes 973 Kbits/sec  
[ 3] 5.0- 6.0 sec 139 KBytes 1.14 Mbits/sec  
[ 3] 6.0- 7.0 sec 49.5 KBytes 405 Kbits/sec  
[ 3] 7.0- 8.0 sec 49.5 KBytes 405 Kbits/sec  
[ 3] 8.0- 9.0 sec 99.0 KBytes 811 Kbits/sec  
[ 3] 9.0-10.0 sec 49.5 KBytes 405 Kbits/sec  
[ 3] 0.0-10.1 sec 1.01 MBytes 843 Kbytes/sec  
kotaro@server1:~$
```

**Right Terminal (UDP Test):**

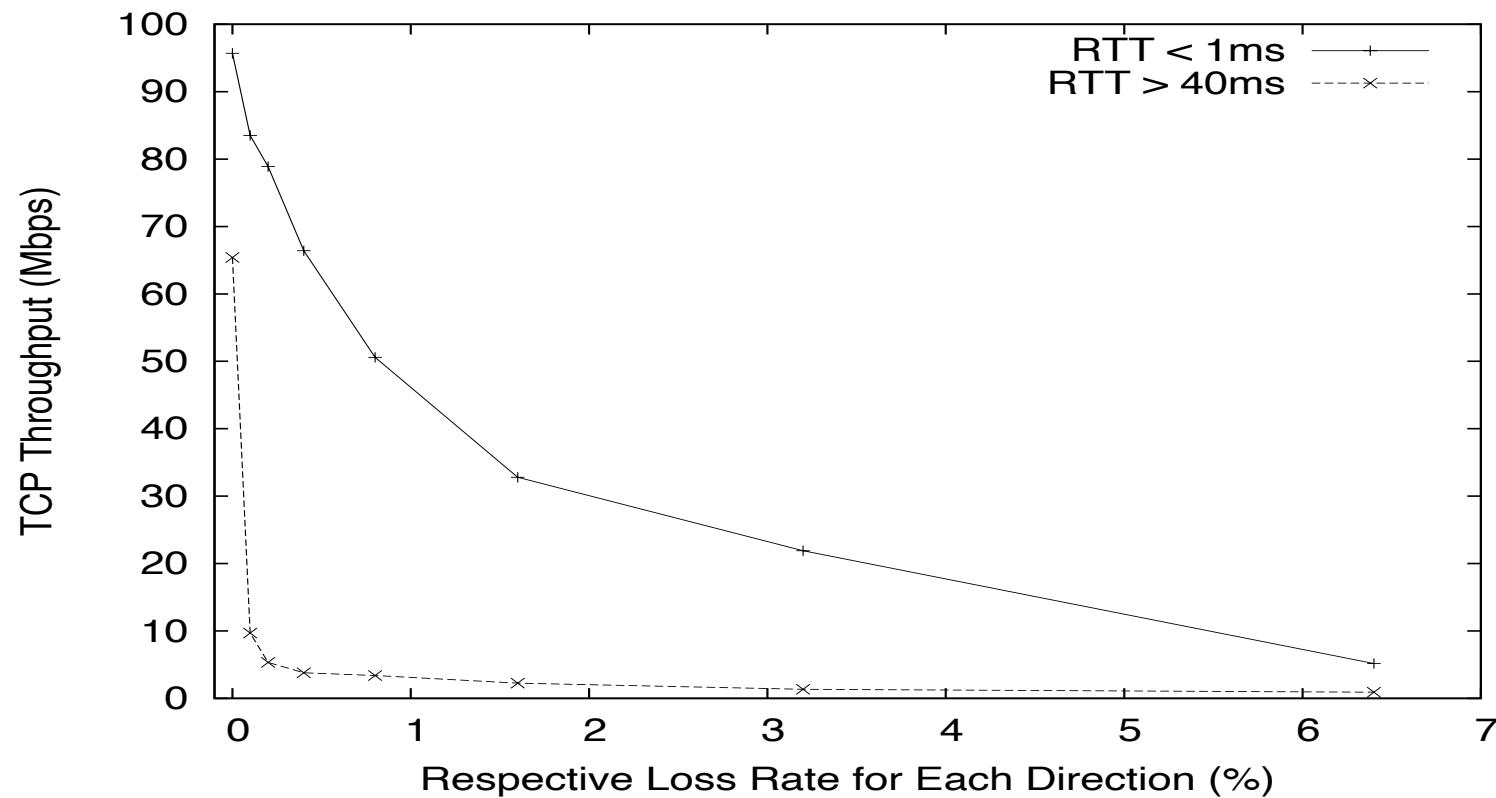
```
kotaro@server1:~$ iperf -c 10.0.0.2 -u -b 100M -i 1  
Client connecting to 10.0.0.2, UDP port 5001  
Sending 1470 byte datagrams, IPG target: 112.15 us (kalman adjust)  
UDP buffer size: 208 KByte (default)  
[ 3] local 10.0.0.253 port 55702 connected with 10.0.0.2 port 5001  
[ ID] Interval Transfer Bandwidth  
[ 3] 0.0- 1.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 1.0- 2.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 2.0- 3.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 3.0- 4.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 4.0- 5.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 5.0- 6.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 6.0- 7.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 7.0- 8.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 8.0- 9.0 sec 12.5 MBytes 105 Mbits/sec  
[ 3] 0.0-10.0 sec 125 MBytes 105 Mbits/sec  
[ 3] Sent 89165 datagrams  
[ 3] Server Report:  
[ 3] 0.0-10.1 sec 117 MBytes 97.0 Mbits/sec 6.443 ms 5948/89165 (6.7%)  
[ 3] 0.0000-10.0859 sec 80814 datagrams received out-of-order  
kotaro@server1:~$ ^C
```

# Many other functions in tc

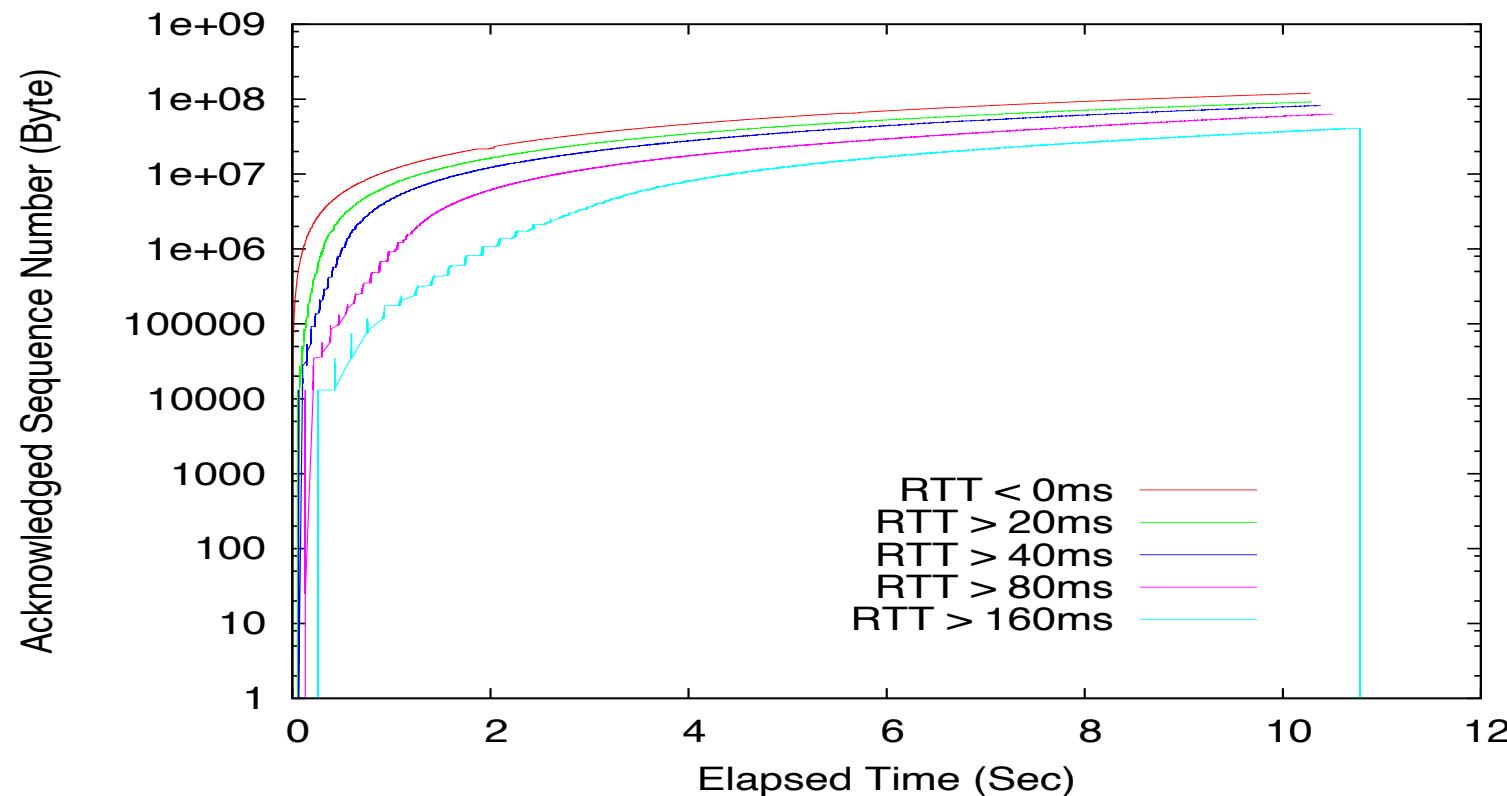
- Packet duplication
- Packet corruption
- Packet re-ordering
- Queuing
- Reference:  
<http://www.linuxfoundation.org/collaborate/workgroups/networking/netem>

Q. How does TCP behave in different conditions?

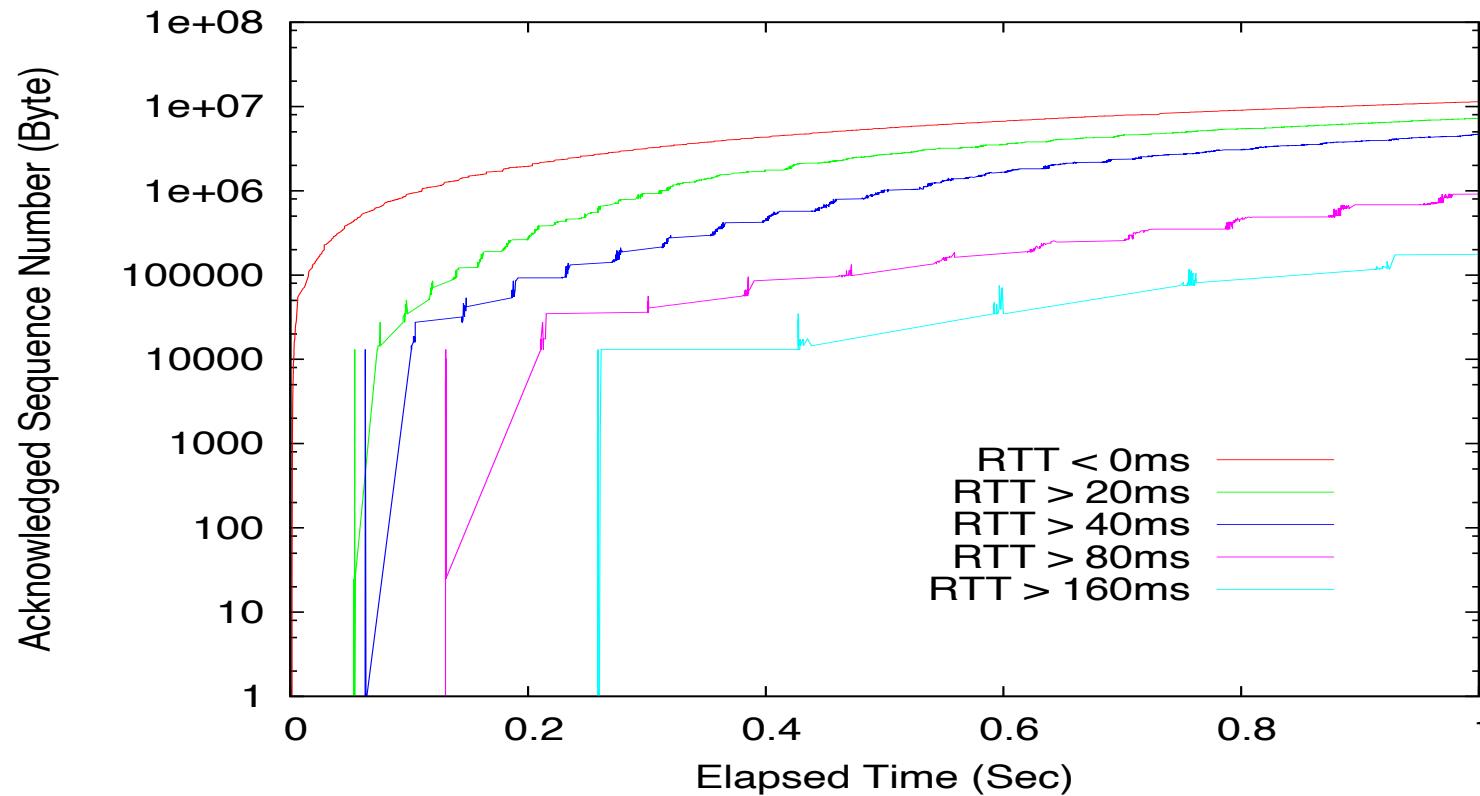
# 1. How does TCP throughput get affected when loss increases?



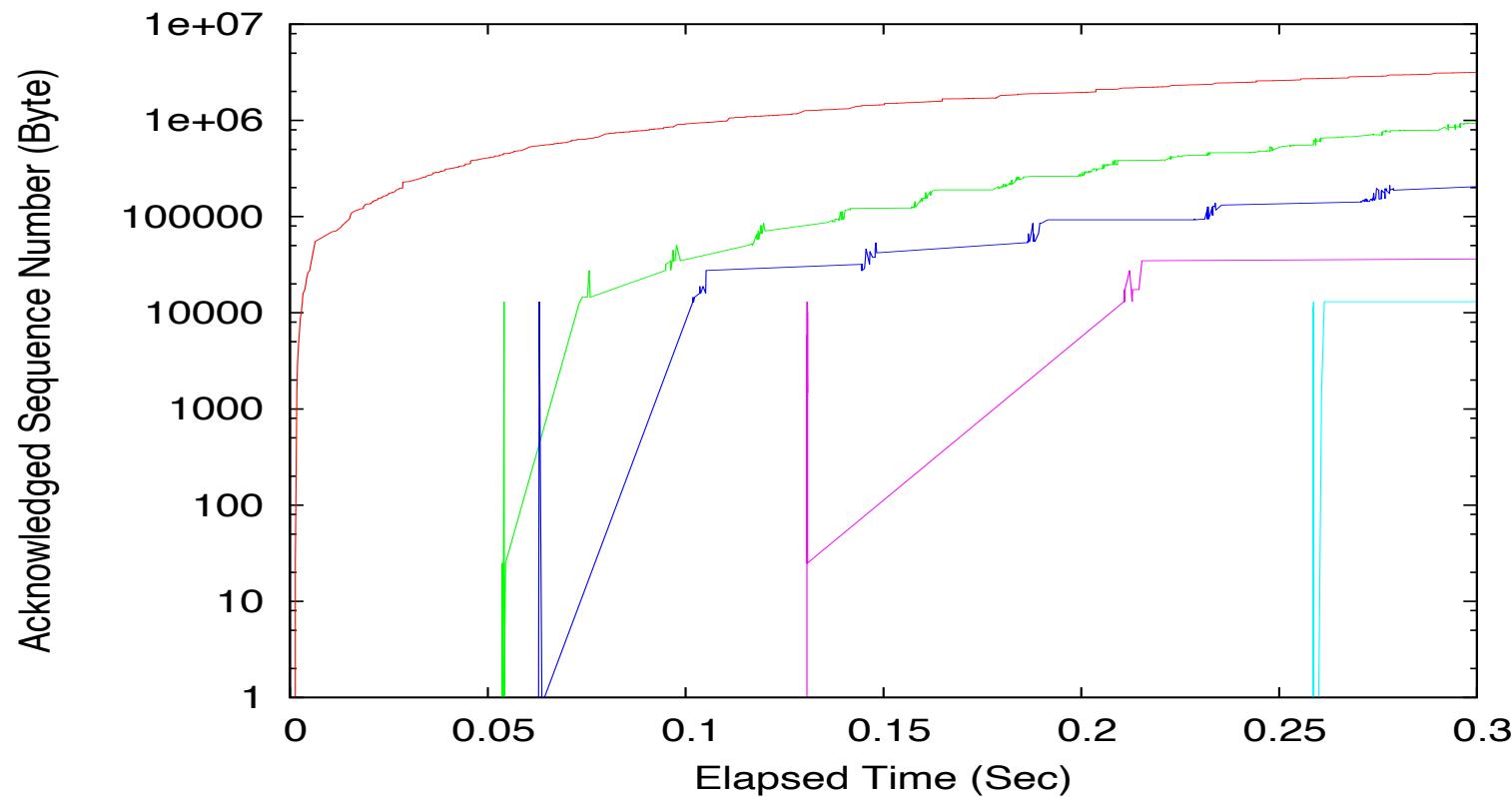
## 2-1. How does acked sequence number grow when RTT increases? (0 to 12 sec, no loss)



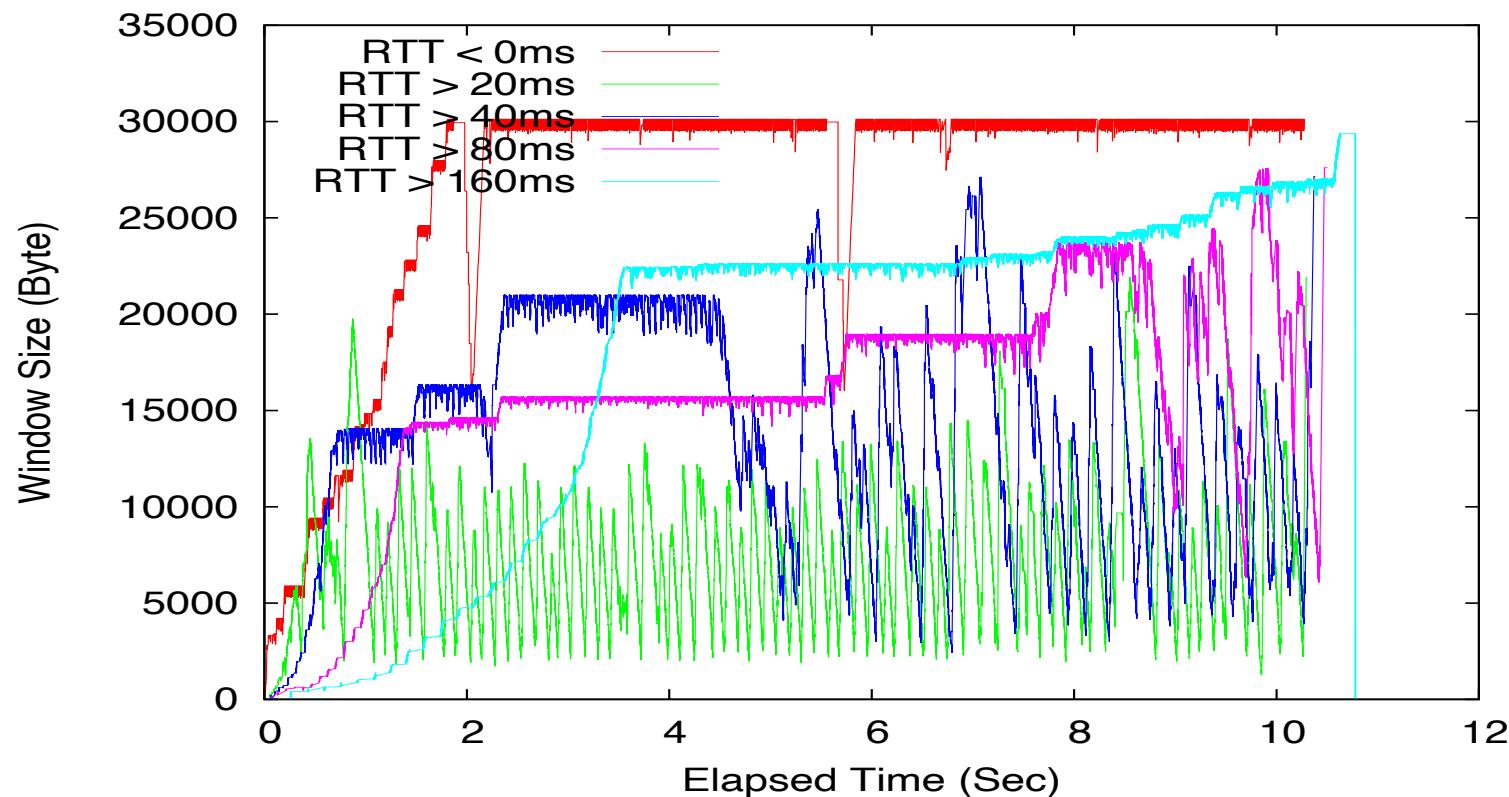
## 2-2. How does acked sequence number grow when RTT increases? (0 to 1 sec, no loss)



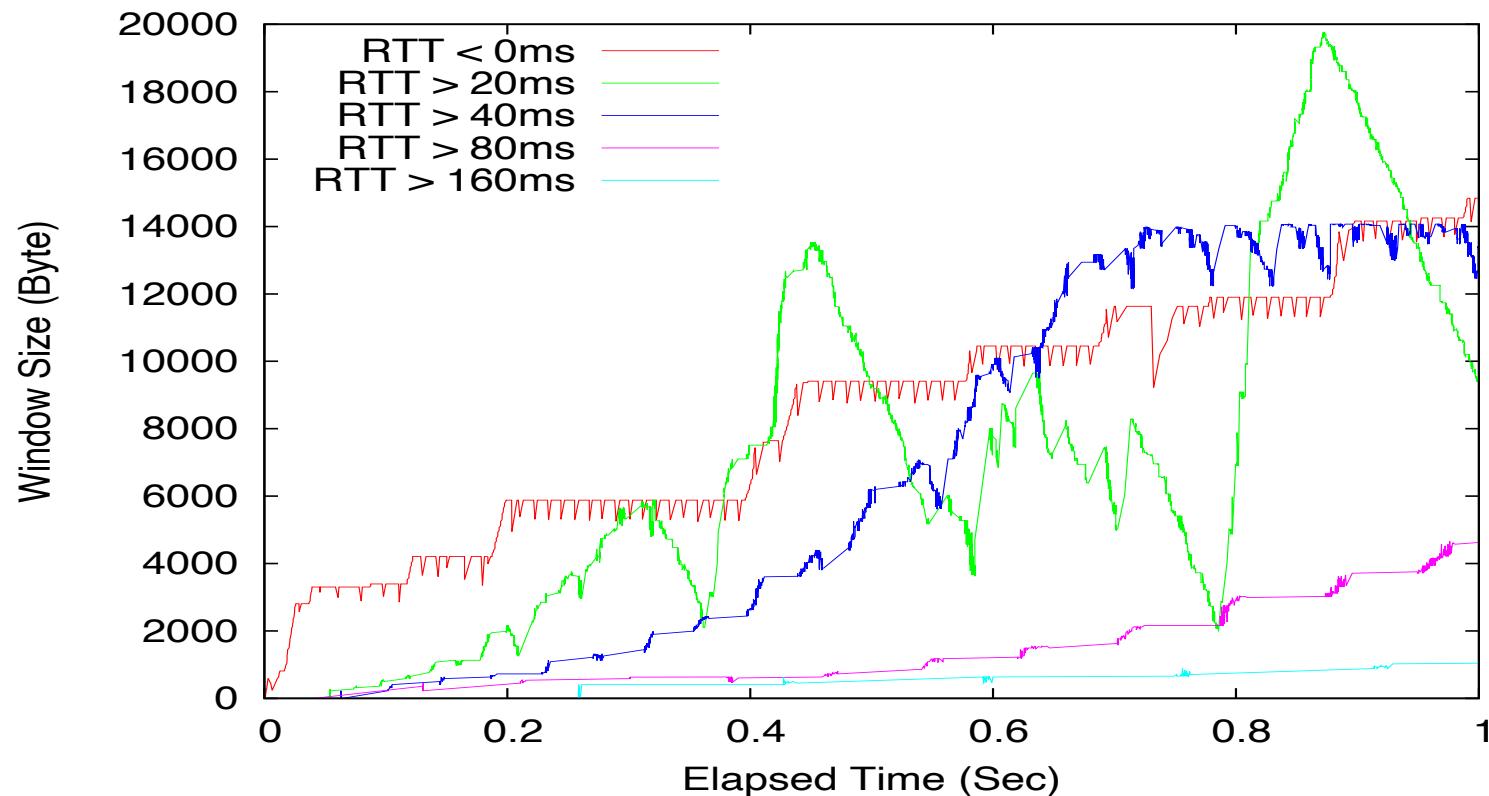
## 2-3. How does acked sequence number grow when RTT increases? (0 to 0.3 sec, no loss)



### 3-1. How does window size grow when RTT increases? (0 to 12 sec, no loss)



### 3-2. How does window size grow when RTT increases? (0 to 1 sec, no loss)



# Way forward?

- Many scenarios can be tested using “tc” and your network configuration / software can be tested in the Internet-like environment.

**DONE!!**