# 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.

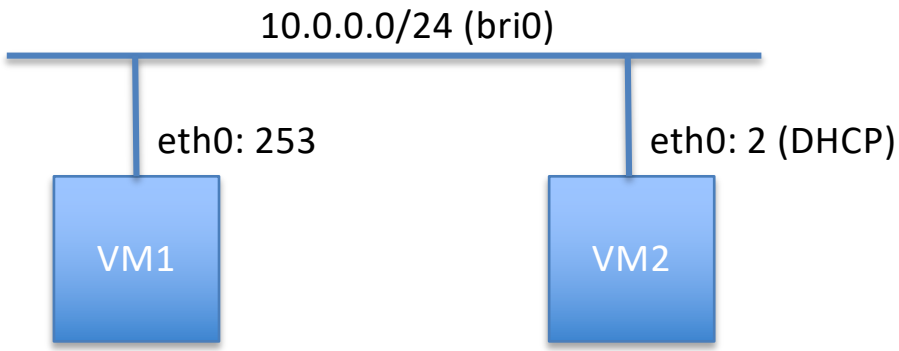
- How? Several options cost something.
  - Buy AWS? Paying money.
  - Run on Simulator? Exporting software to simulator.
  - Do something on your LAN? You may disturb others.

- 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: ~

```
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: ~

```
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 p
[ 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: ~

```
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

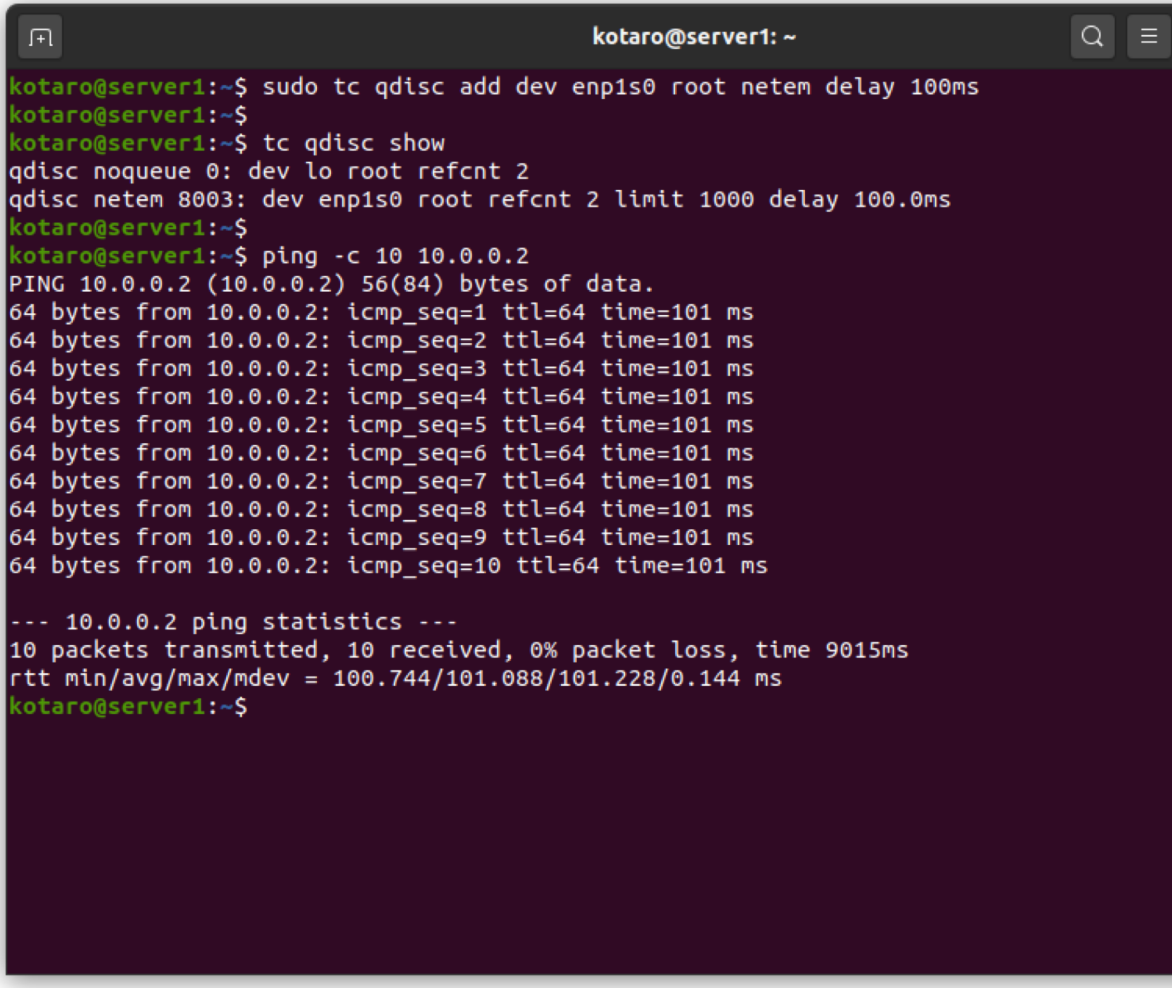
```
# tc qdisc add dev eth0 root netem delay 100ms
```

- 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`

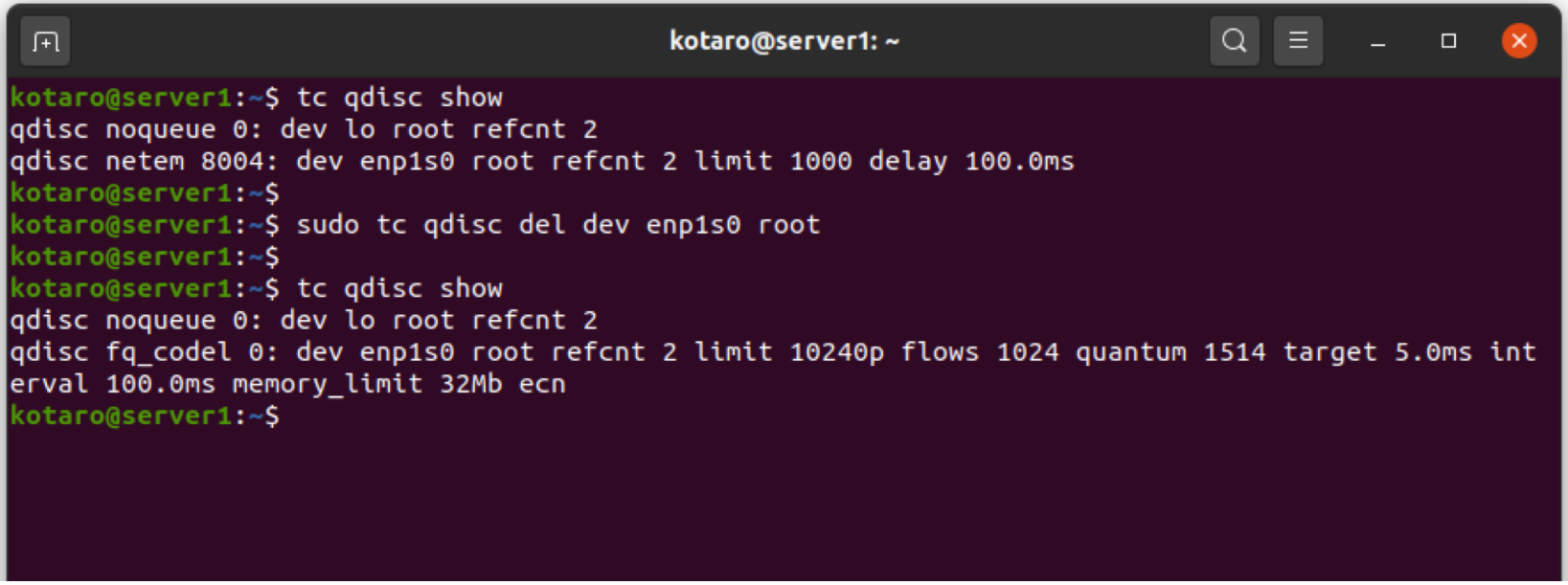
- Use ping command to see how the RTT is affected



# Deleting the queue

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

- The additional delay is removed from the NIC.



# 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

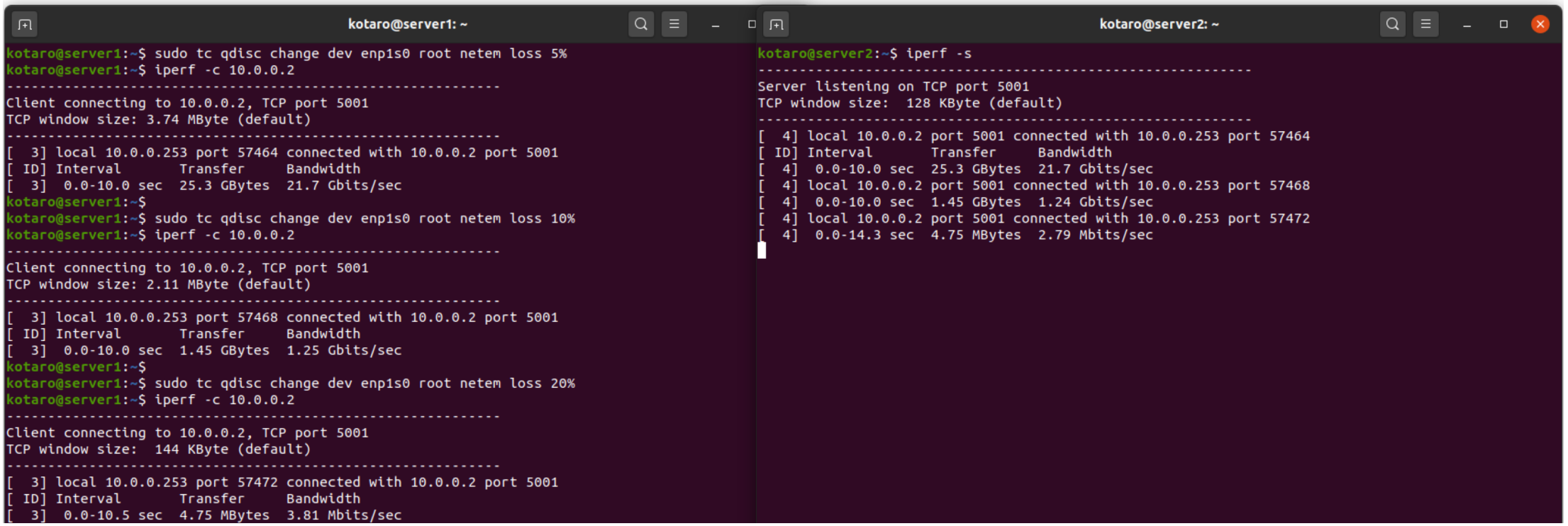
```
# tc qdisc change dev eth0 root netem delay 100ms 20ms
distribution normal
```

- Applying normal distribution to delay
- 100ms: Average (μ), 20ms: Standard Deviation (σ)

# 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!

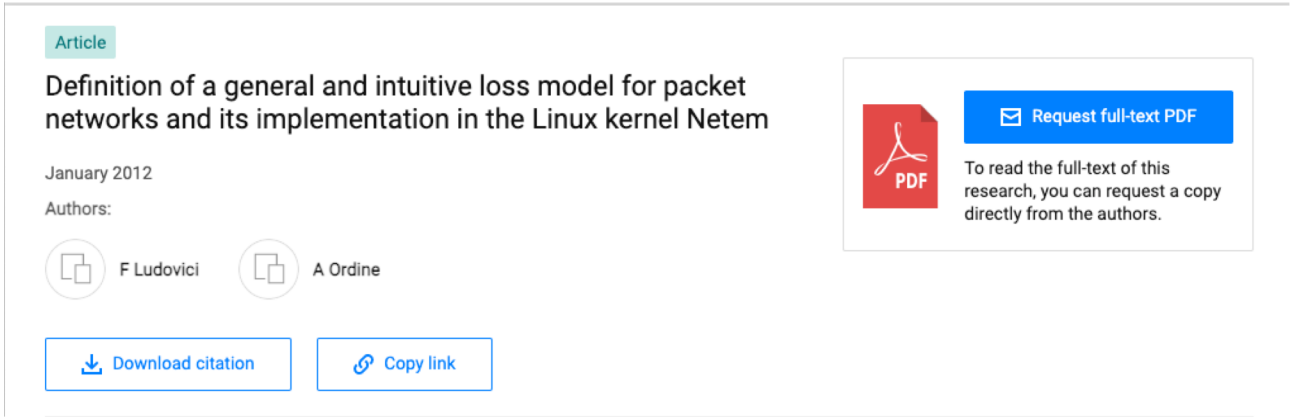


# Emulating Bursty Loss?

```
# 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)

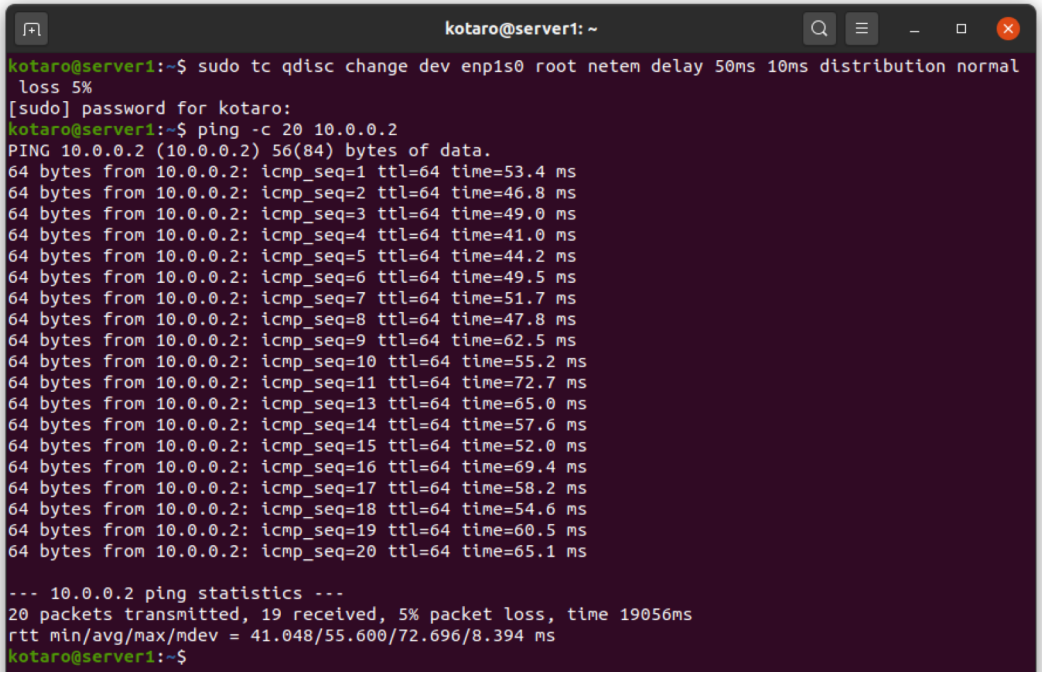
- This is a known technique, but not very effective.
- Refer the following article if interested.



# Combining Delay and Loss

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

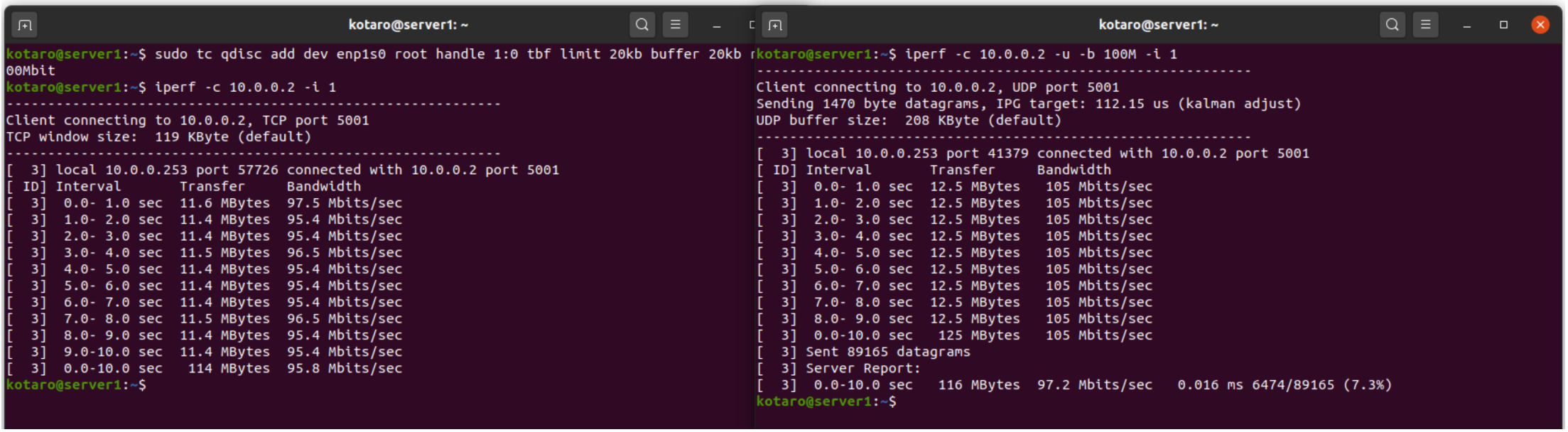
- 50ms Normal Distribution Delay and 5% Loss
- Networks can be more like Internet (?)



# Bandwidth Throttling

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

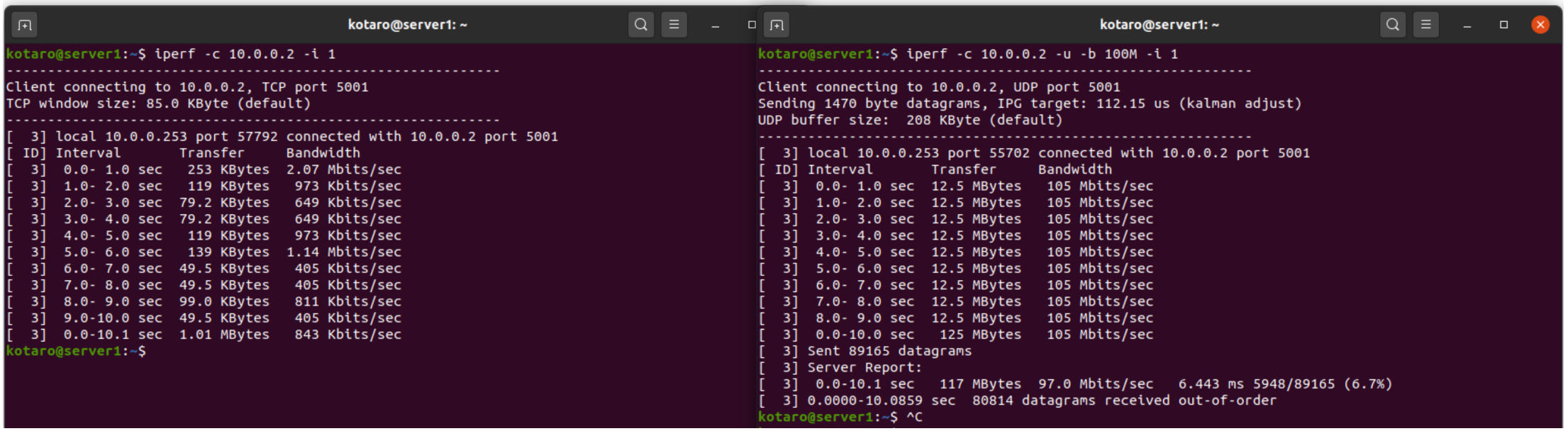
- tbf: The Token Bucket Filter is suited for slowing traffic down to a precisely configured rate. Scales well to large bandwidths.
- `100Mbit`: 100Mbps



# Combining Bandwidth Throttling, Delay and Loss

```
# tc qdisc add dev eth0 root handle 1:0 tbf 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



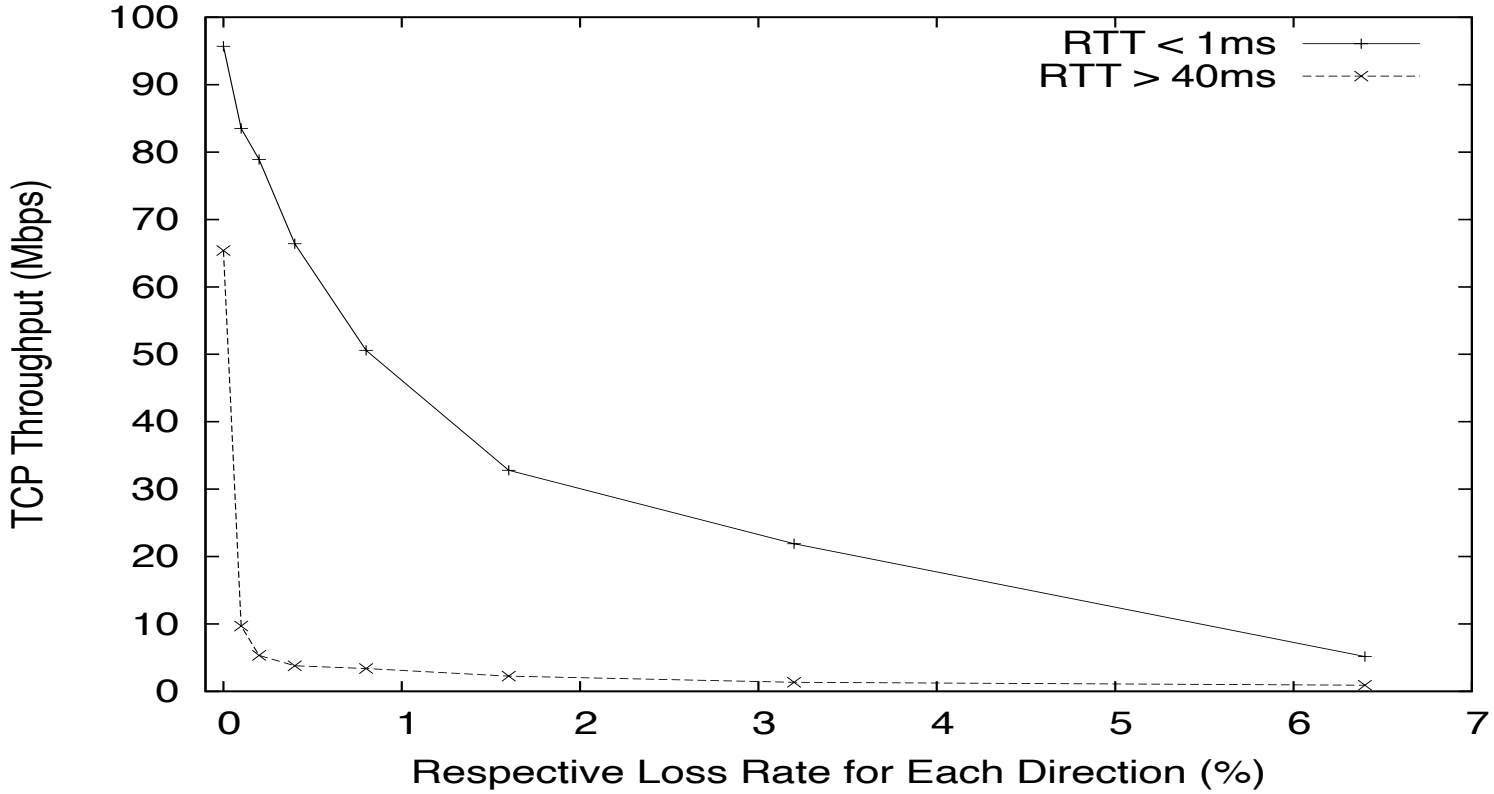
# Many other functions in tc

- Packet duplication
- Packet corruption
- Packet re-ordering
- Queuing

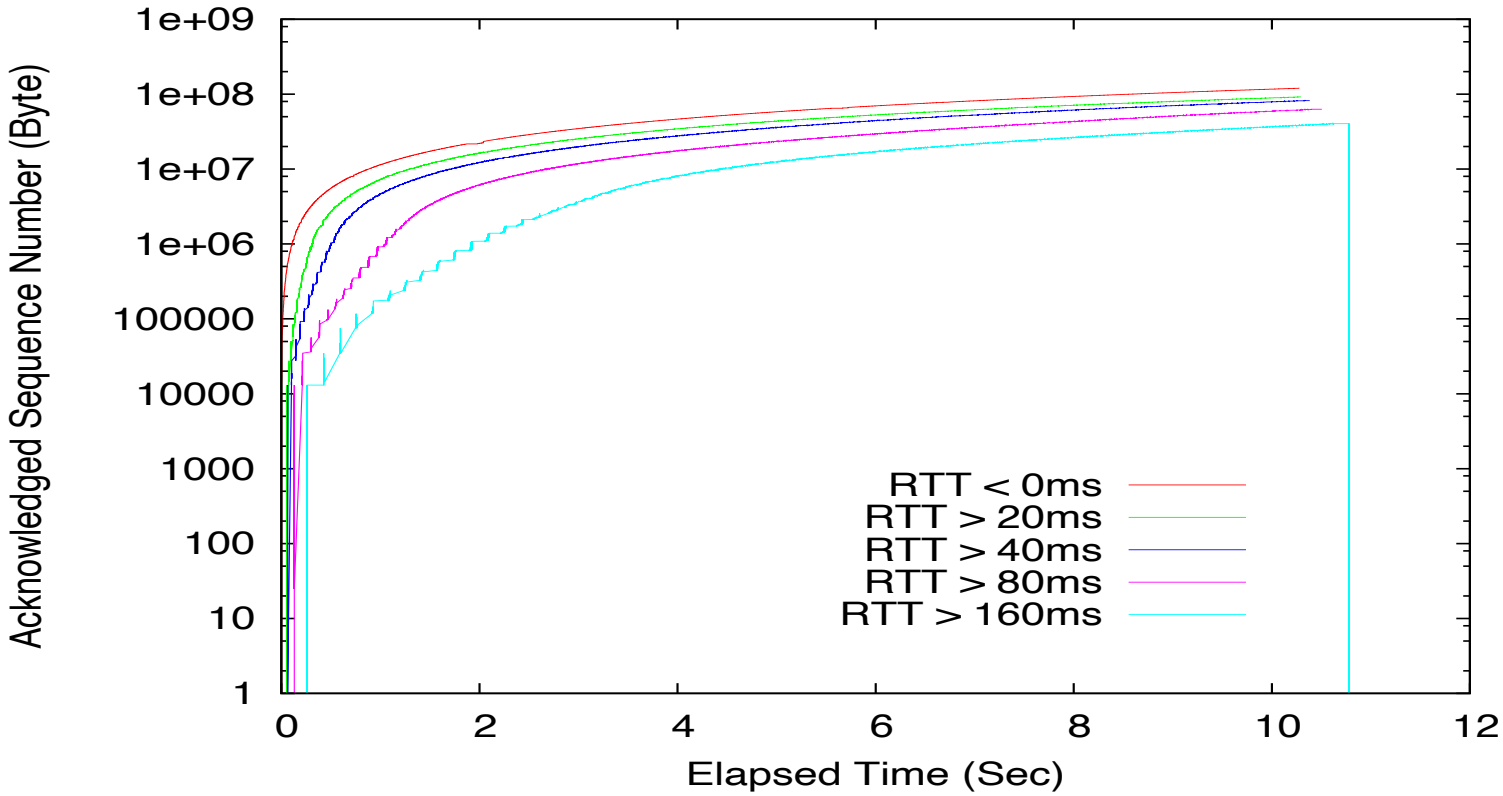
- Reference: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

# Q. How does TCP behave in different conditions?

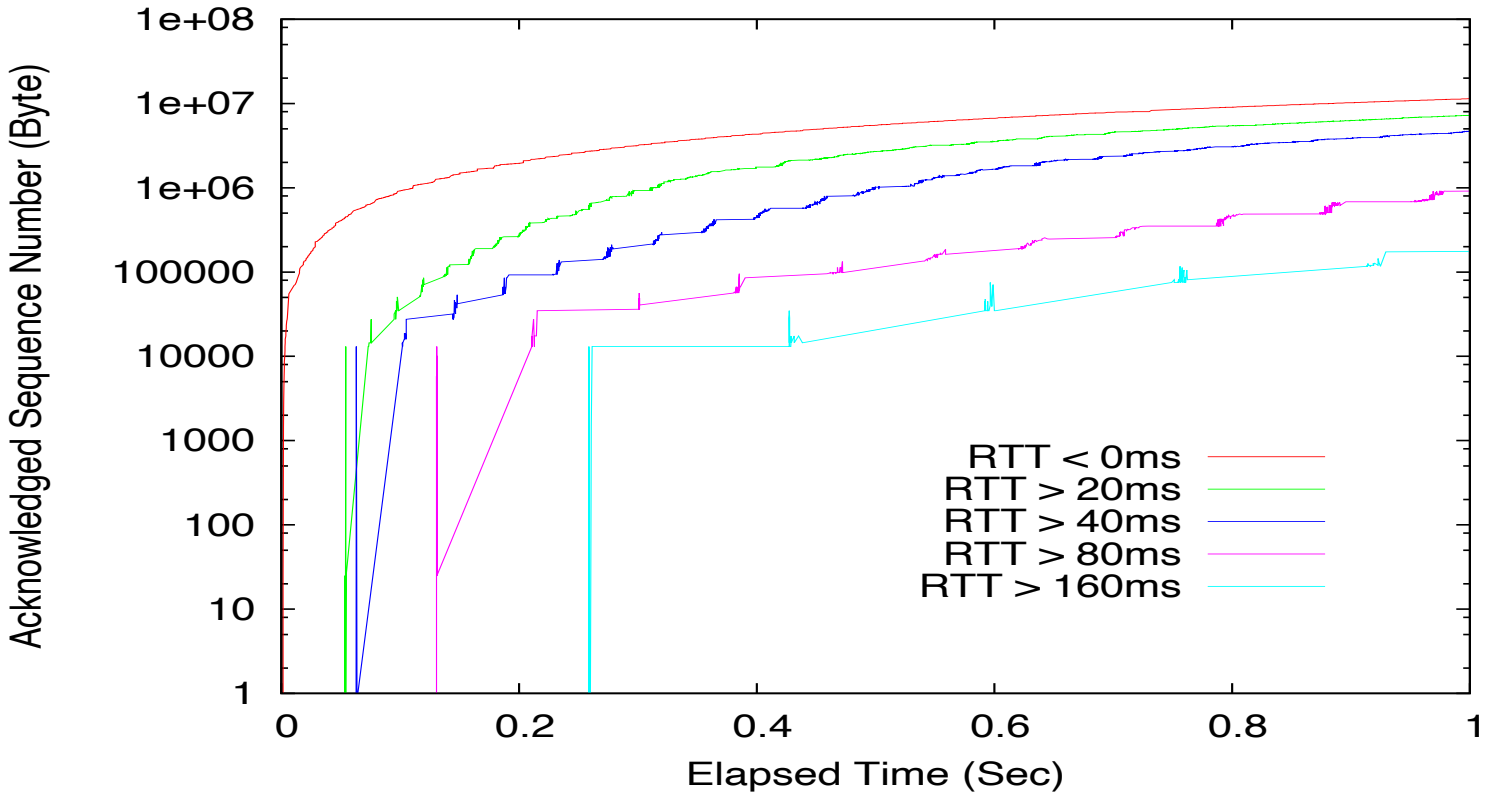
# 1. How does TCP throughput get affected when loss increases?



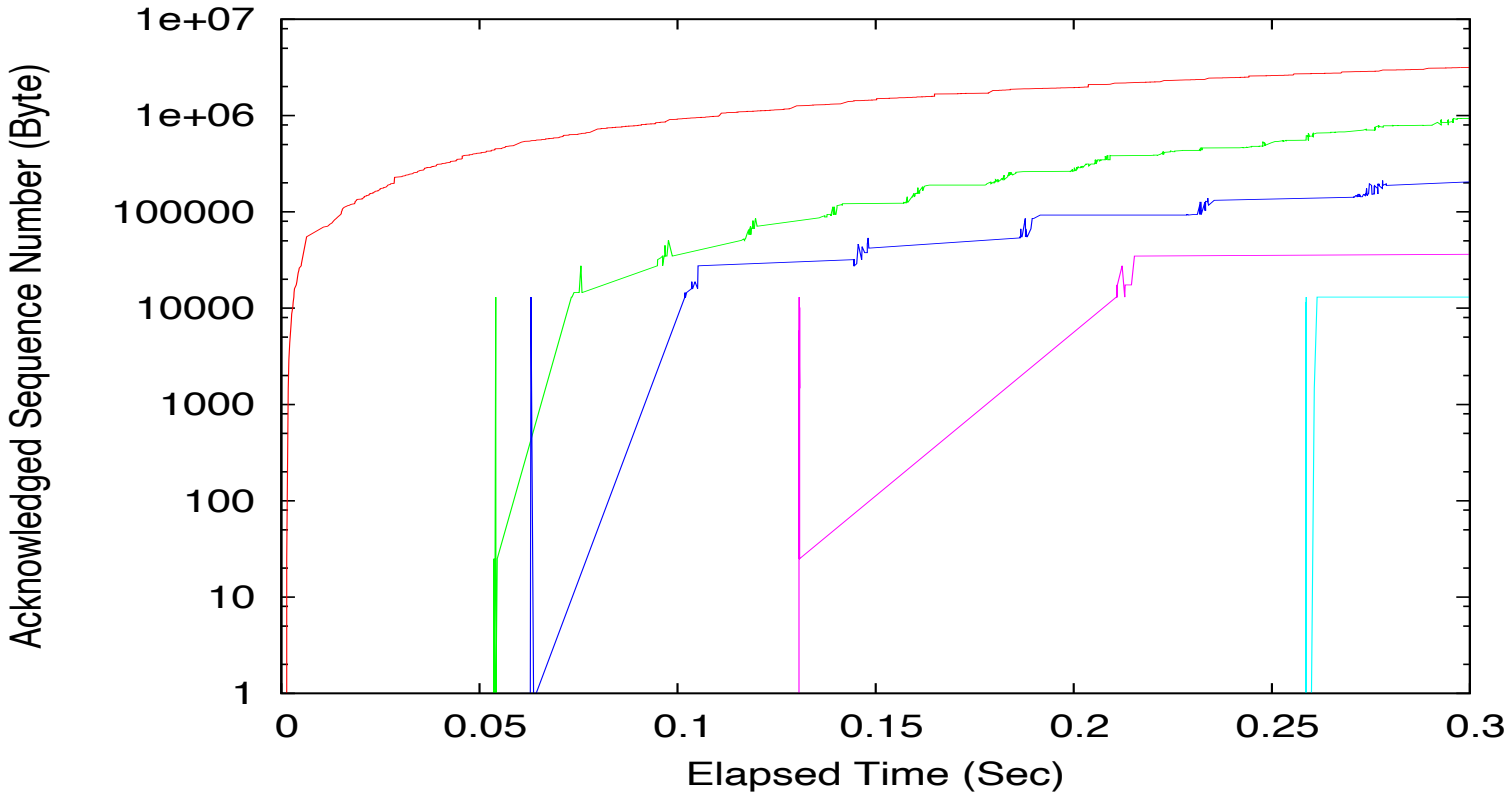
# 2-1. How does acked sequence number grow when RTT increases? (0 to 12 sec, no loss)



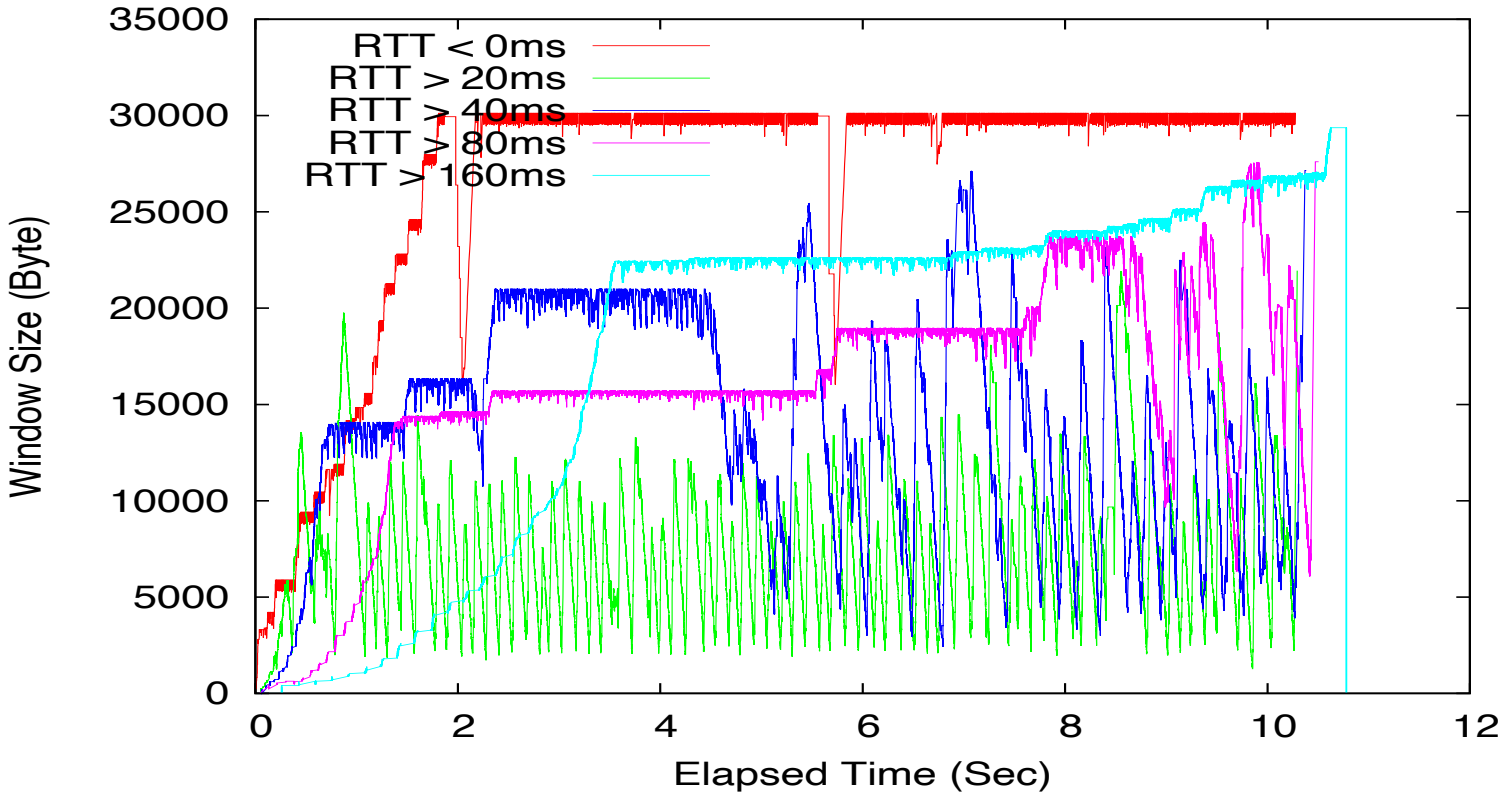
# 2-2. How does acked sequence number grow when RTT increases? (0 to 1 sec, no loss)



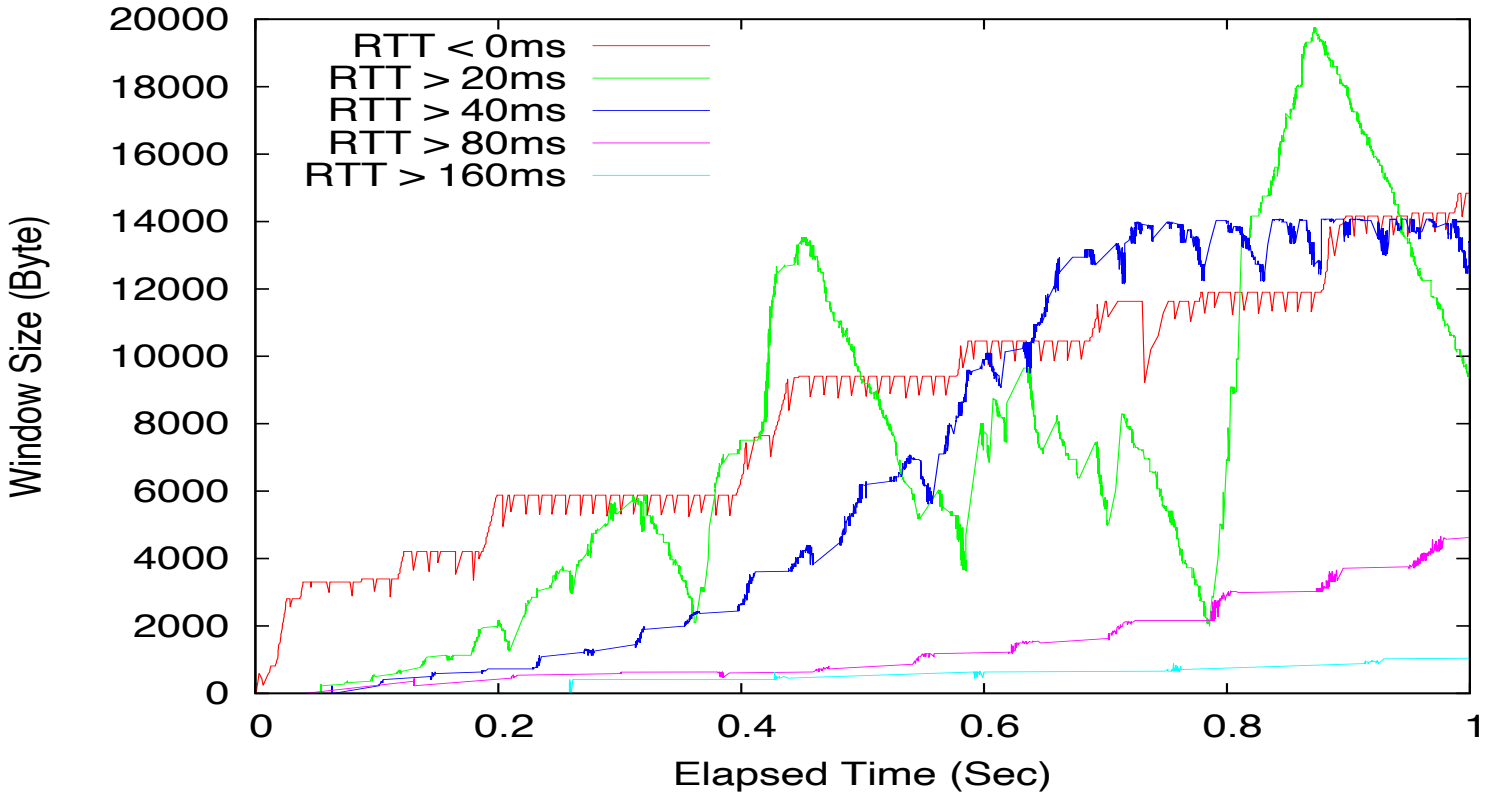
# 2-3. How does acked sequence number grow when RTT increases? (0 to 0.3 sec, no loss)



# 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!!