Study of Congestion Control and Its Impact on Network Throughput

Karthik Vegesna

Abstract

Networks today are seeing high growth in traffic. With the increase in data consumption and internet usage, the need for networks that can adequately adapt and ensure high throughput, especially in situations where bandwidth and data requirements being higher than switch capacity, is at an all-time high. However, despite the pressing need of such congestion management, many networks seem to have missed out on adding congestion control to the switches within their network. Network congestion control can be implemented by creating multiple queues for various traffic types to ensure efficient link sharing there by increasing overall network performance. The purpose of the project is to simulate multiple queues in switches and show the benefit of bandwidth reservation in such a network as compared to a network with switches with a single queue. A successful undertaking of this project and adoption of congestion control within both traditional and Software Defined Networking (SDN) networks would have broad implications on network performance.

Introduction

Network congestion is a reality in most networks today as networks are trying hard to keep up with ever increasing network traffic. In these situations, use of congestion control mechanisms based on implementing multiple queues with bandwidth reservation for the various traffic types can lead to high switch efficiency and better network performance. This project quantitatively showed these advantages by automatically generating traffic via Python scripts run on Mininet and implementing multiple queues with bandwidth reservation to achieve better performance. However, with the addition of multiple queues and bandwidth reservation, the network was able to better react to the congestion in the network leading to superior quality of service.

Project Setup and Configuration

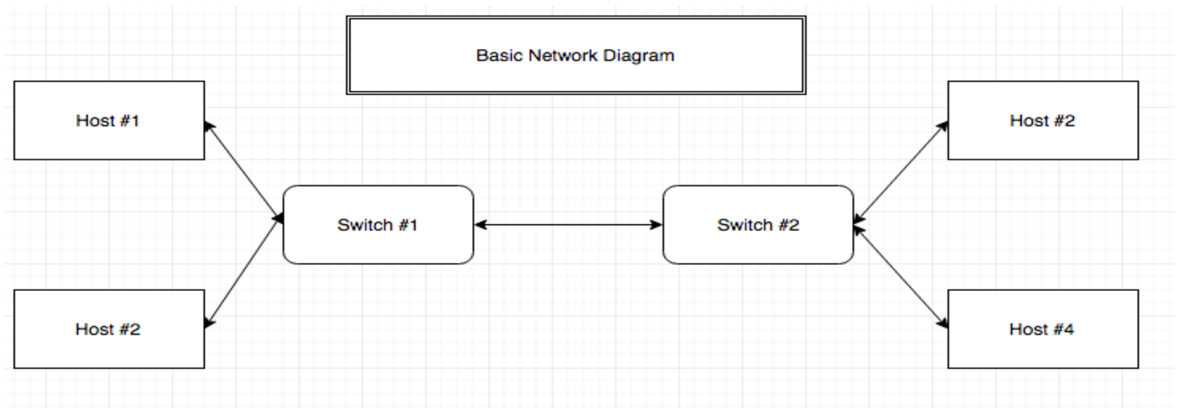
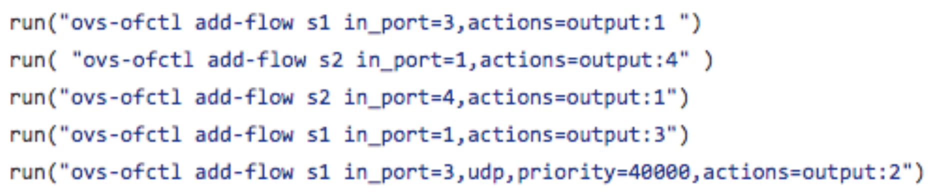


Figure 1 - Network Topology

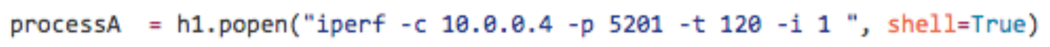
The project created a virtual network with four hosts and two switches as shown in Figure 1 using Mininet, a software to create a complete virtual network topology. Within the Mininet network topology, iPerf is used to generate both TCP and UDP traffic of varying bandwidth between hosts with traffic flowing from Host 1 to Host 4. The Python script creates 2 hosts and 2 OVSs, that are connected via a TCLink, and used traffic control to limit bandwidth to 5 Mbps, 5 ms delay and 0% packet loss. The traffic consists of a long TCP flow for 120 seconds, and recurring 10 second UDP flows of increasing bandwidth, from 0 MB to 12 MB. This created a scenario for network congestion due to increasing UDP traffic that severely impacted the throughput of the TCP traffic.

The following flow entries are implemented in the switches.



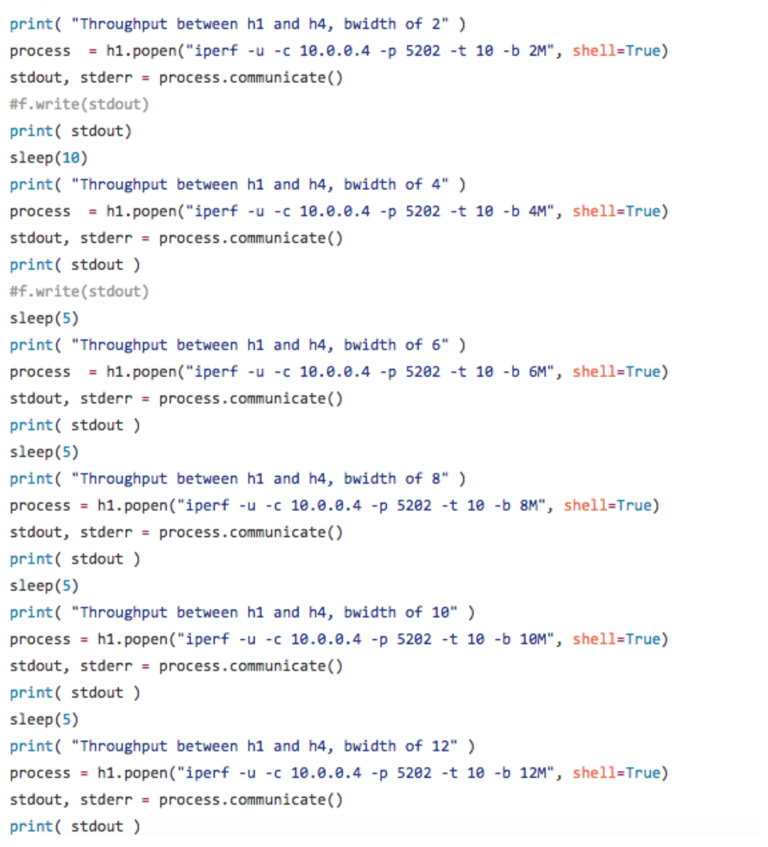
```
run("ovs-ofctl add-flow s1 in_port=3,actions=output:1 ")
run( "ovs-ofctl add-flow s2 in_port=1,actions=output:4" )
run("ovs-ofctl add-flow s2 in_port=4,actions=output:1")
run("ovs-ofctl add-flow s1 in_port=1,actions=output:3")
run("ovs-ofctl add-flow s1 in_port=3,udp,priority=40000,actions=output:2")
```

I generated one long running TCP flow for 120 seconds, as well as recurring UDP flows.



```
processA  = h1.popen("iperf -c 10.0.0.4 -p 5201 -t 120 -i 1 ", shell=True)
```

Further, I started adding UDP flows of 5 second intervals of increasing bandwidth, as seen here:



```
print( "Throughput between h1 and h4, bwidth of 2" )
process  = h1.popen("iperf -u -c 10.0.0.4 -p 5202 -t 10 -b 2M", shell=True)
stdout, stderr = process.communicate()
#f.write(stdout)
print( stdout)
sleep(10)
print( "Throughput between h1 and h4, bwidth of 4" )
process  = h1.popen("iperf -u -c 10.0.0.4 -p 5202 -t 10 -b 4M", shell=True)
stdout, stderr = process.communicate()
print( stdout )
#f.write(stdout)
sleep(5)
print( "Throughput between h1 and h4, bwidth of 6" )
process  = h1.popen("iperf -u -c 10.0.0.4 -p 5202 -t 10 -b 6M", shell=True)
stdout, stderr = process.communicate()
print( stdout )
sleep(5)
print( "Throughput between h1 and h4, bwidth of 8" )
process = h1.popen("iperf -u -c 10.0.0.4 -p 5202 -t 10 -b 8M", shell=True)
stdout, stderr = process.communicate()
print( stdout )
sleep(5)
print( "Throughput between h1 and h4, bwidth of 10" )
process = h1.popen("iperf -u -c 10.0.0.4 -p 5202 -t 10 -b 10M", shell=True)
stdout, stderr = process.communicate()
print( stdout )
sleep(5)
print( "Throughput between h1 and h4, bwidth of 12" )
process = h1.popen("iperf -u -c 10.0.0.4 -p 5202 -t 10 -b 12M", shell=True)
stdout, stderr = process.communicate()
print( stdout )
```

The program was run without any congestion control and the results were captured using iPerf. The output was parsed and a graph generated for TCP throughput with an increase in UDP bandwidth. This was done by using the matplotlib and pandas Python libraries.

Next, queues for congestion control and associated bandwidth reservations were implemented on the two Open vSwitches. The same process was repeated and the iPerf results with congestion control were captured.

Project Results

Without congestion control, the captured results showed a fall in TCP throughput in the switches with increasing UDP congestion. But with congestion control in place, TCP throughput was steady despite of the recurring increased bandwidth of the UDP flows. All the results are tabulated below. The graphical representation of these results as captured in Figure 2 clearly depicted the throughput results with and without congestion control.

Q - With Queues
NQ - Without Queues

UDP Bandwidth – 0 (NQ)

Time (s)	TCP Throughput
1	1.06 MB
2	1.14 MB
3	1.13 MB
4	1.08 MB
5	1.09 MB
6	1.11 MB
7	1.15 MB
8	1.03 MB
9	1.06 MB
10	1.05 MB

UDP Bandwidth – 0 (Q)

Time (s)	TCP Throughput
1	542 KB
2	588 KB
3	583 KB
4	577 KB
5	580 KB
6	571 KB
7	566 KB
8	588 KB
9	577 KB
10	592 KB

UDP Bandwidth – 2 (NQ)

Time (s)	TCP Throughput
1	939 KB
2	928 KB
3	939 KB
4	922 KB
5	939 KB
6	919 KB
7	854 KB
8	930 MB
9	930 MB
10	930 MB

UDP Bandwidth – 2 (Q)

Time (s)	TCP Throughput
1	567 KB
2	563 KB
3	587 KB
4	562 KB
5	581 KB
6	542 KB
7	571 KB
8	588 KB
9	572 KB
10	581 KB

UDP Bandwidth – 4 (NQ)

Time (s)	TCP Throughput
1	653 KB
2	585 KB
3	690 KB
4	687 KB
5	699 KB
6	693 KB
7	653 KB
8	602 KB
9	693 KB
10	701 KB

UDP Bandwidth – 4 (Q)

Time (s)	TCP Throughput
1	559 KB
2	572 KB
3	585 KB
4	583 KB
5	591 KB
6	583 KB
7	579 KB
8	591 KB
9	585 KB
10	588 KB

UDP Bandwidth – 6 (NQ)

Time (s)	TCP Throughput
1	411 KB
2	387 KB
3	436 KB
4	469 KB
5	464 KB
6	484 KB
7	421 KB
8	390 KB
9	420 KB
10	427 KB

UDP Bandwidth – 6 (Q)

Time (s)	TCP Throughput
1	556 KB
2	578 KB
3	589 KB
4	567 KB
5	569 KB
6	563 KB
7	585 KB
8	583 KB
9	583 KB
10	591 KB

UDP Bandwidth – 8 (NQ)

Time (s)	TCP Throughput
1	267 KB
2	266 KB
3	300 KB
4	334 KB
5	474 KB
6	247 KB
7	255 KB
8	288 KB
9	269 KB
10	270 KB

UDP Bandwidth – 8 (Q)

Time (s)	TCP Throughput
1	566 KB
2	577 KB
3	581 KB
4	563 KB
5	567 KB
6	567 KB
7	582 KB
8	588 KB
9	587 KB
10	593 KB

UDP Bandwidth – 10 (NQ)

Time (s)	TCP Throughput
1	267 KB
2	266 KB
3	229 KB
4	234 KB
5	244 KB
6	247 KB
7	255 KB
8	228 KB
9	239 KB
10	260 KB

UDP Bandwidth – 10 (Q)

Time (s)	TCP Throughput
1	567 KB
2	589 KB
3	587 KB
4	574 KB
5	583 KB
6	580 KB
7	575 KB
8	569 KB
9	574 KB
10	563 KB

Graphical Representation of the Fall in Throughput

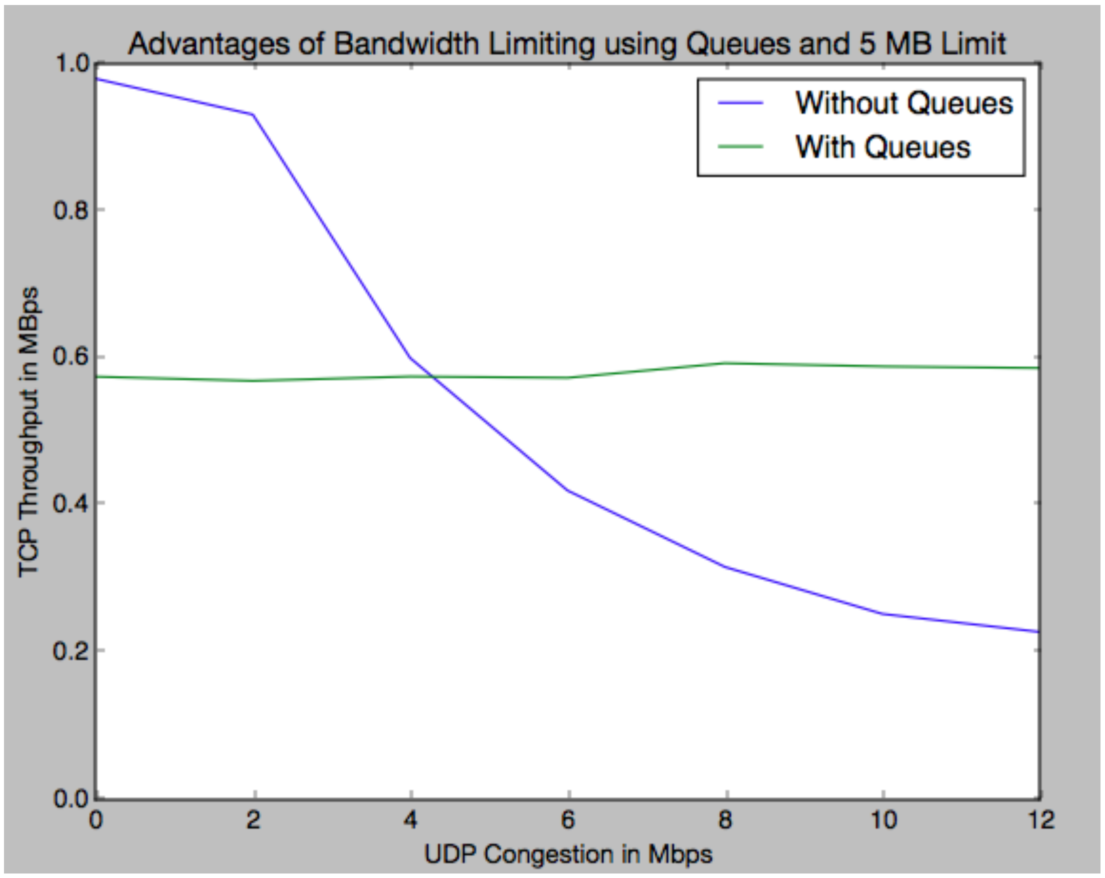


Figure 2 TCP Throughput w/o Congestion Control with increasing UDP Congestion

Future Implications

In summary, it is shown that congestion control mechanisms can lead to positive impact on TCP traffic throughput. These results apply equally to both traditional and SDN based networks. In SDN networks, the network controller can not only automate the creation of the flow entries in the switches but it can also detect congestion and can take automated congestion control mechanisms such as creation to queues and bandwidth reservation to improve network performance.

In networks where congestion is very common especially during peak traffic periods, adoption of congestion control within both traditional and Software Defined Networking (SDN) networks would have broad implications on network performance.

My work and code in full form is available at: https://github.com/kvegesna/congestion-control-with-queues.

Acknowledgements

The project was done at Open Networking Laboratory (ON.Lab) located in Menlo Park, California that builds tools and platforms to enable and accelerate the use of Software Defined Networking.I would specifically like to thank Ms. Sandhya Narayan and my supervisor Mr. Brian O'Connor for all their help during this project.

References:

- http://mininet.org/api/hierarchy.html
- http://www.colorado.edu/itp/sites/default/files/attached-files/70129-130943_-_derrick_dsouza_-_apr_25_2016_859_pm_-_team_5_improving_qos_in_sdn_redo.pdf
- https://github.com/mininet/mininet/wiki/Bufferbloat