Using Virtual Queue Simulation and Bandwidth Reservation in Switches to show the enhanced QoS of such a network.

Karthik Vegesna

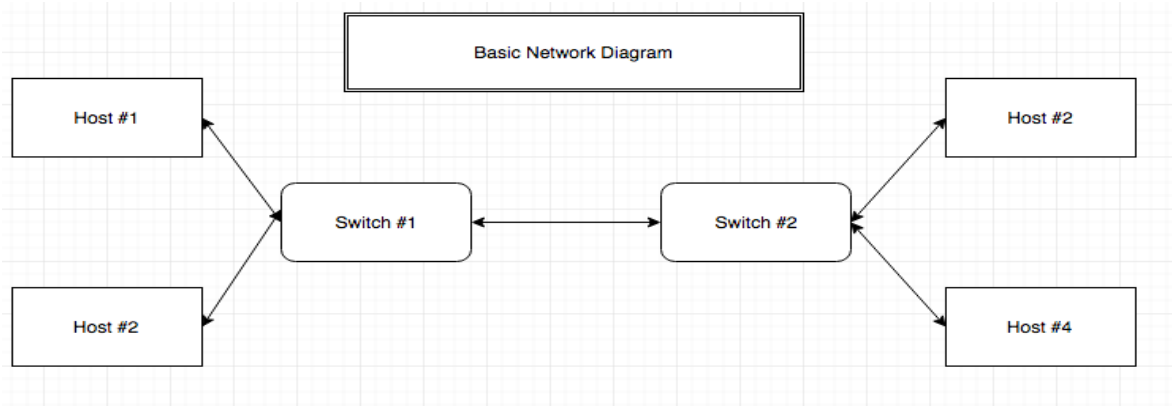
Abstract

Networks today see only a rise in overall traffic. With the increase in data consumption and internet usage, the need for networks that can adequately adapt and ensure high throughput, despite 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 adding congestion control to their network and the simple queue to ensure link sharing and an overall increase in performance. The purpose of my project was to simulate queues in switches and show the benefit of bandwidth reservation for different classes of applications that use the network of such switches as compared to switches without queues. A successful undertaking of this project could have much larger implications and can lead to adoption in future network designs.

Introduction

Bandwidth Reservation has been often regarded as a concept that can lead to high switch efficiency and better performance for congested networks. Additionally the use of queues often leads to higher overall performance, especially when there is congestion or a great deal of traffic coming from different sources and of different types. My project quantitatively showed these advantages by automatically generating traffic via Python scripts run on Mininet and using bandwidth reservation in queues to achieve better performance. 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 was one long TCP flow for 120 seconds, followed by recurring 10 second UDP flows of increasing bandwidth, from 0 MB to 12 MB. This created congestion and the TCP throughput was severely affected. However, with the addition of queues, the network was able to better react to the congestion in the network leading to superior quality of service.

Project Setup and Configuration



The project had a virtual network using Mininet, a software that allows you to create hosts, switches, and a complete network in general. Further, it used iPerf to generate both TCP and UDP traffic between hosts of varying bandwidth and time. I created four hosts with most of the traffic passing from Host 1 to Host 4. I also used two Open vSwitches through which I could create queues for congestion control and performance enhancements. Additionally, it is important to note that the network didn't have a centralized controller and instead wrote down all the flows to map traffic to the right places.

```
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, as well as recurring UDP flows as seen here.

This is the TCP flow I have created, which runs for a total of 120 seconds.

```
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 )
```

Finally, the program would run and generate the required output. Further, I parsed through the output and generated a graph depicting the fall in throughput with an increase in UDP bandwidth. This was done by using the matplotlib and pandas Python libraries.

Project Results

I was able to generate results as expected showing a fall in throughput in the switches without queues. These tables show a clear indication of the fall in throughput, with the increase in the overall UDP Bandwidth. Finally, a graph would be created, mapping the TCP Throughput with the UDP Bandwidth, thereby clearly showing the detrimental effects of congestion in networks 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

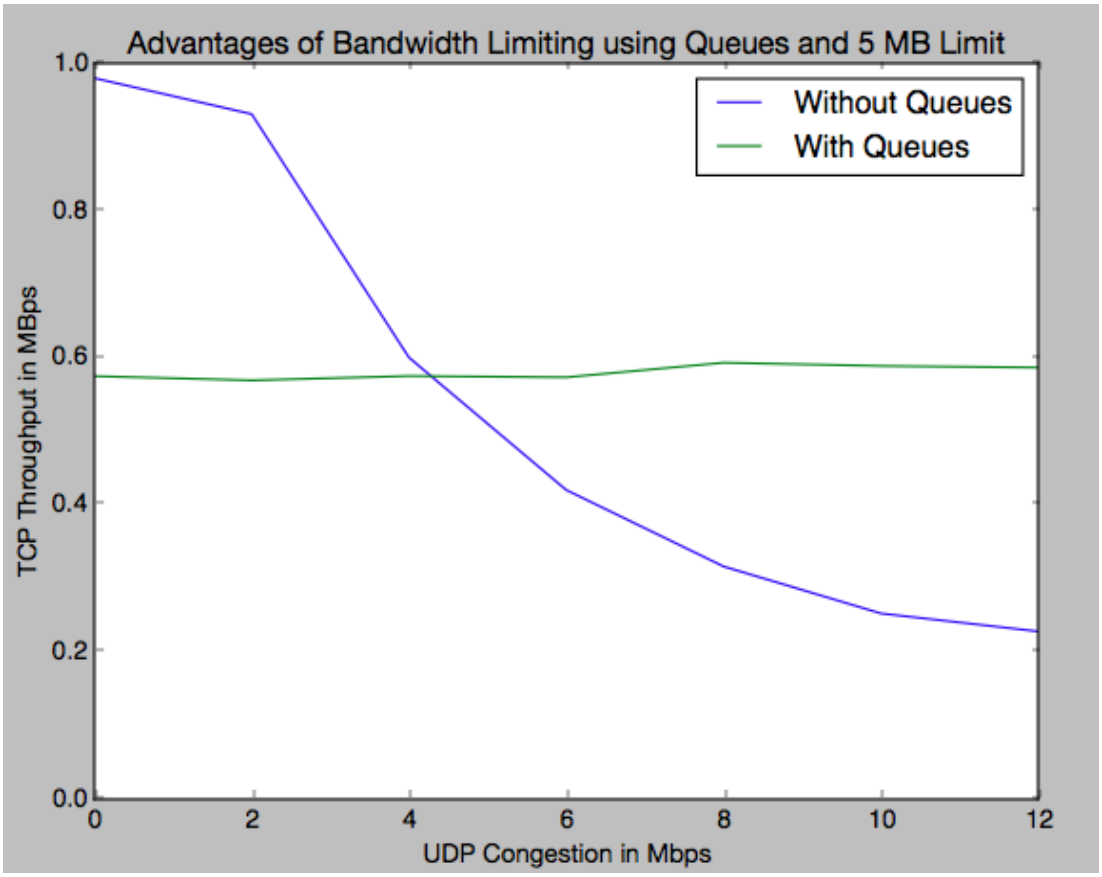
UDP Bandwidth – 10 (NQ)

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



Future Implications

Overall, the conclusions were quite clear. Adding queues to networks will have an extremely beneficial impact on a network and may lead to better congestion control.

It is also important to note that this project can have large future implications and may lead to more widespread use of queueing in SDN. Further, a network controller can detect congestion and then automatically implement queues of some sort and thereby help improve overall network performance.

Acknowledgements

The project was done at Open Networking Laboratory, a startup 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.