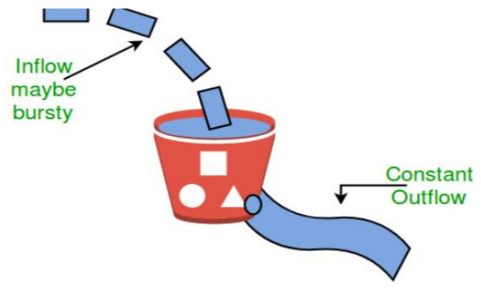
Project Title: BBR versus Cubic TCP: Experimentation and Analysis

Yelleswarapu Venkata Praveen Kumar - 111986420
Susritha Reddy YaramReddy - 111870572

Problem Statement:
The main goal of this project is to understand and compare the working of BBR and Cubic protocols under various circumstances by deploying servers at various locations, and to verify the fairness of BBR.

Introduction:
Congestion is a state occurring in network layer when message traffic is so heavy, as too many sources send too much data too fast for network to handle, that it slows down network response time. It can be shown as below:



Congestion Control algorithms are thus very important in any network as they determine the speed of the network. Two such congestion control algorithms are Cubic TCP and BBR. All the congestion control algorithms including Cubic before the invention of BBR interpreted loss of packets as Congestion. Cubic is an enhanced version of BIC with increased throughput and RTT fairness. Window growth function is a cubic function of time and thus the name. The major difference between Cubic and other TCP algorithms is that Cubic does not rely on receipt of ACKs to increase window size, it depends on last congestion event instead.

A loss based congestion control algorithm relies only on indications of lost packets as the signal to slow down. This worked well for many years, because internet switches' and routers' small buffers were well-matched to the low bandwidth of internet links.

But loss-based congestion control is problematic in today's diverse networks. In shallow buffers, packet loss happens before congestion. With today's high-speed, long haul links that use commodity switches with shallow buffers, even if the packet loss comes from transient traffic bursts, loss-based congestion control can result in abysmal throughput as it overreacts by halving the sending rate upon packet loss. In deep buffers, congestion happens before packet loss,

loss-based congestion control causes **bufferbloat** problem, by repeatedly filling the deep buffers in many links and by causing seconds of needless queuing delay.

BBR considers how fast the network is delivering data by using recent round trip time (RTT) and network's delivery rate to build a model that includes both the maximum recent bandwidth available, and its minimum recent round-trip delay. It then uses this model to control both how fast it sends data and maximum amount of data it's willing to allow in the network at any time.

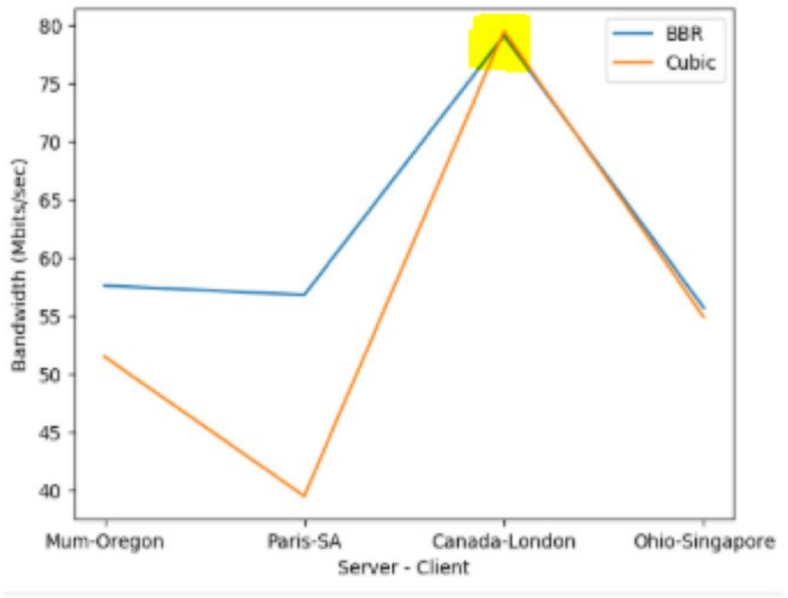
Experiments performed:
We have performed the following experiments to achieve our goal. The experiments include Real Environment using Amazon AWS and Simulated Environment using Traffic Control.

- Conducted a measurement study by measuring the performance of BBR and Cubic protocols based on bandwidth across different servers and clients deployed in Canada, London, USA, India etc using Amazon's cloud service, AWS.
- Conducted a measurement study after simulating a constant packet loss using TC (traffic control) to compare the performance of BBR and cubic protocols based on bandwidth in lossy networks.
- Conducted a measurement study by running Cubic and BBR protocols simultaneously on same server and client machines to check the fairness of BBR.

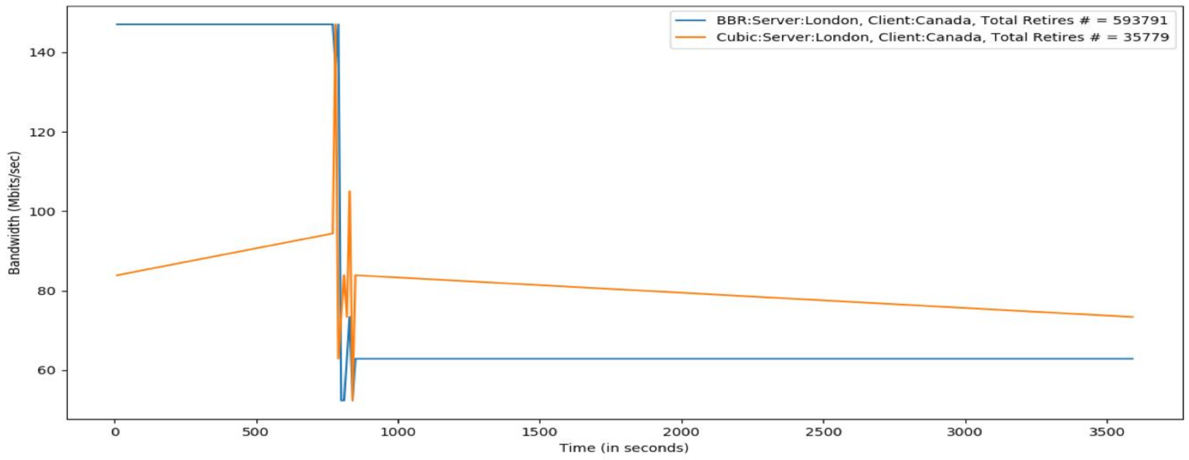
Experiments in Detail:

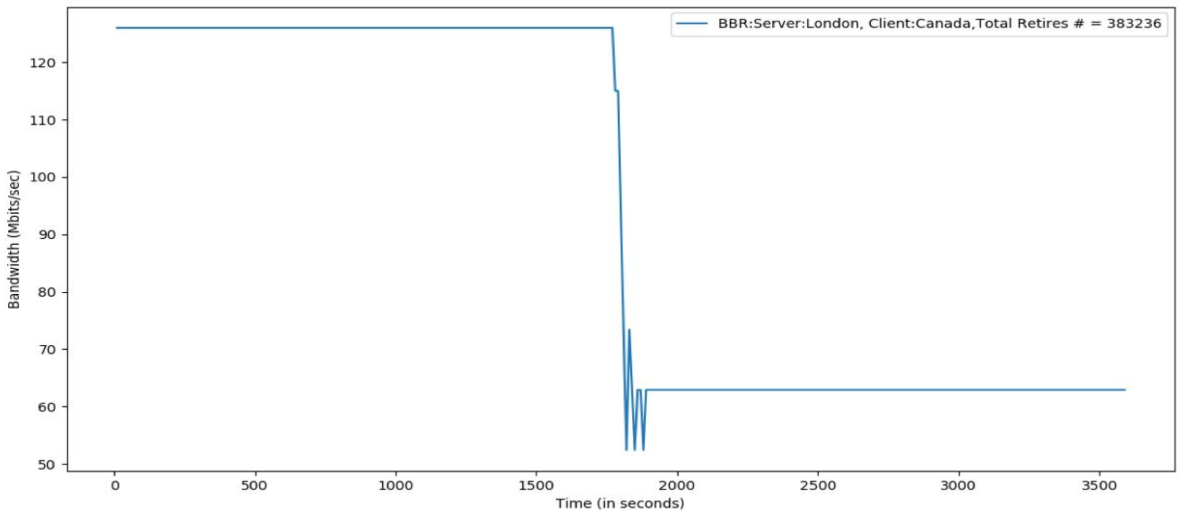
Experiment 1 :
Approach and Setup: Created servers and clients in 8 different regions over the world using Amazon EC2 instances. Used Iperf3 tool to measure the bandwidth by sending data from one instance to another in different location. In all these tests, servers sent the data and clients received the data. We observed that on an average, **BBR performed approximately 10% better than cubic.** Tests were run for an hour between the servers and clients. The following are the clients and servers used:

Client	**Server**
Mumbai	Oregon
Paris	South America
Canada	London
Ohio	Singapore



Observation - In BBR, as shown in the above graph, once when the retries are too many, the bandwidth was almost halved, too many retries occur sequentially and this low bandwidth gets stabilized till the end of test. This test was repeated to make sure that it is not a transient behavior. This was observed between server deployed in london and client deployed in Canada over AWS. (Highlighted this observed behavior in above graph in yellow)





In the above graphs, we took initial point at 10th second, final point at 3550th second and remaining points just before and just after too many retries so as to highlight the issue.

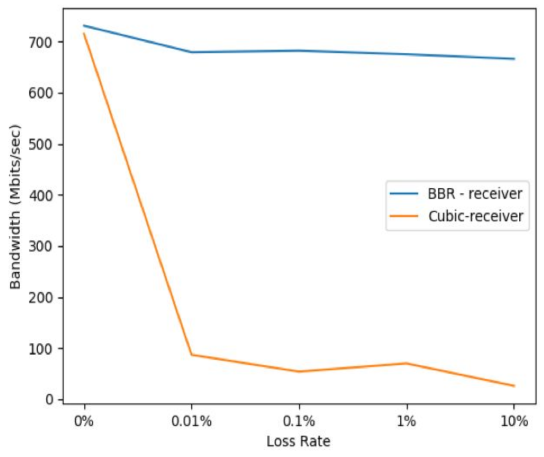
Hypothesis - Due to BBR's indifference to packet loss, when the router's buffers get full, BBR still sends data at a high rate, thus buffer's will always be full, there by BBR got stuck in this loop.

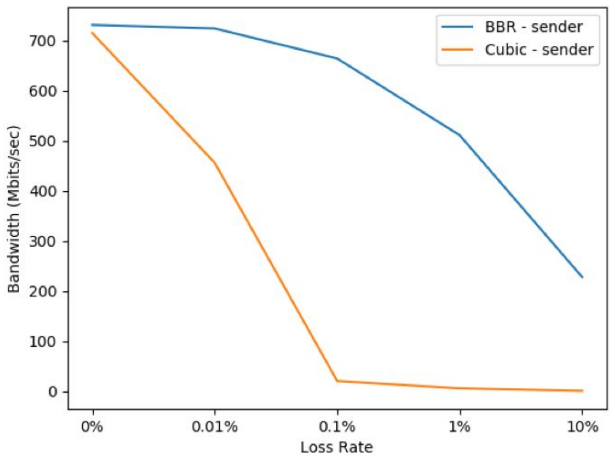
Experiment 2:

TC(Traffic Control) was used to simulate a steady constant loss rate which is equivalent to a lossy network.

Approach: A steady constant loss rate was set both at sender and receiver's side separately to compare BBR and Cubic.

Observation - As the loss rate increases, BBR outperforms cubic.





This might happen due to the following reasons:

1) Due to constant loss, Cubic gets stuck in congestion avoidance phase and never gets a chance to improve its window size.

2) Due to BBR's indifference to packet loss, even when packet loss happens, BBR doesn't decrease it's window size.

Experiment 3 : Fairness

The key challenge in having a setup for fairness is to see if the iperf instances are actually using BBR and Cubic or not.

Approach:

1) Firstly, performed a test to check if an iperf instance which runs using some congestion algorithm gets affected, when changes are made to the config file (where we modify the congestion algorithm)

Observation: It doesn't effect the ongoing iperf instance.

2) Tested if we can run two different congestion algorithms at the same time on iperf using different ports

Observation : We can have two seperate iperf instances simultaneously

3) Finally, set a high loss rate using TC and deployed two iperf instances simultaneously between the same server and client machines, so as to differentiate BBR and Cubic protocols using bandwidth.

Observation : Both iperf instances were running simultaneously between the same server and client machines, out of which one was using BBR and the other was using Cubic. This was clear as the instance which was using BBR had a very high bandwidth while the one using cubic had a very low bandwidth.

Experiment: Ran two iperf instances which use Cubic and BBR simultaneously on same server and client machines with **loss rate = 0** to see if BBR is fair to cubic or not.

Observation: When BBR and cubic were ran independently their bandwidth was 731 and 715 Mbits/sec, when they were run simultaneously, BBR's throughput came out to be 454 Mbits/sec whereas Cubic's was 462 Mbits/second. Running this second time gave throughput of 479 Mbits/sec to cubic and for BBR it was 438 Mbits/second.

The results from the above experiments show that BBR is fair to Cubic.

Fairness experiment according to us might account to the additional 30%.

Github Link:
https://github.com/Susritha/BBR-versus-Cubic-TCP-Experimentation-and-Analysis

References:
http://materias.fi.uba.ar/7543/dl/Paper_BBR.pdf
https://queue.acm.org/detail.cfm?id=3022184
http://doc.tm.uka.de/2017-kit-icnp-bbr-authors-copy.pdf
https://pdfs.semanticscholar.org/6e1d/f66dca7361178d56dbb21d7909c51ec42ec1.pdf
https://cloud.google.com/blog/products/gcp/tcp-bbr-congestion-control-comes-to-gcp-your-internet-just-got-faster