*Performance Analysis of BBR Congestion Control*

*Protocol Based on NS3*

Suyash Musalgaonkar 2020H1030127P

Avinash Dasigi 2020H1030123P

Ashwin Shriyan 2020H1030125P

|  |  |
| --- | --- |
| Table of contents | Page No. |
| 1.Introduction | 03 |
| 2.BBR Algorithm | 03 |
| 3.BBR States | 05 |
| 4.BBR vs TCP Reno | 06 |
| 5.BBR vs TCP Vegas | 08 |
| 6.Project Execution | 09 |
| 7.Conclusion | 10 |
| 8.Referrences | 11 |

**Introduction**

Bottleneck Bandwidth and Round-trip propagation time (BBR) is a TCP congestion control algorithm developed at Google in 2016. Up until recently, the Internet has primarily used loss-based congestion control, relying only on indications of lost packets as the signal to slow down the sending rate. This worked decently well, but the networks have changed. We have much more bandwidth than ever before; The Internet is generally more reliable now, and we see new things such as buffer bloat that impact latency. BBR tackles this with a ground-up rewrite of congestion control, and it uses latency, instead of lost packets as a primary factor to determine the sending rate.

**BBR Algorithm:**

BBR attempts to run a TCP connection at the bottleneck bandwidth

rate with minimal delay. This happens only when the total

Data in flight is equal to the bandwidth-delay product (bandwidth×

Delay) or BDP.

* To compute the BDP, BBR determines the minimum round-trip

time (Rmin) and the maximum delivery rate on the path from the sender to the receiver.

1. To determine Rmin, BBR keeps a window of round-trip time estimates for the past 10 seconds. Rmin is then selected as the smallest value in this window.
2. To determine Bmax, BBR keeps a window of receiver delivery rate estimates for the past 10 round-trip times. Bmax is then selected as the largest value in this window.

* BBR uses Bmax and Rmin to determine the number of bytes to

have in flight BDP = Bmax × Rmin, allowing the TCP congestion window to grow to a small multiple of the BDP.

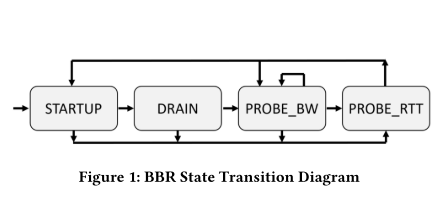
* BBR paces sending packets at a rate that matches the bottleneck bandwidth (Bmax ) – the pacing rate is BBR’s primary control parameter.
* Every time BBR receives a packet acknowledgment, it estimates

the round-trip-time (RTT) and bandwidth (Btlbw ) for that packet. It then adds RTT and Btlbw to the round-trip time and bandwidth windows, respectively.

BBR goes through 4 distinct phases to adjust the pacing rate and congestion window in order to reach steady state conditions quickly

network changes to bottleneck bandwidth or round-trip time.

BBR’s state transition diagram is shown in Figure 1.



* **BBR States:**

1. A BBR flow begins in the STARTUP state and quickly ramps up

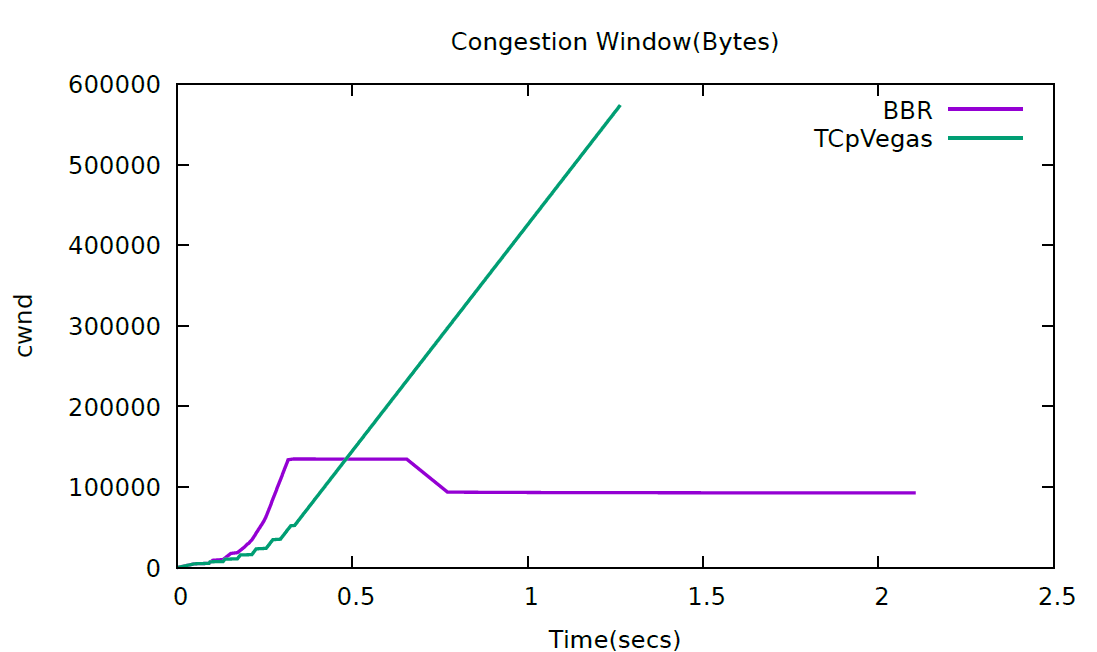
its sending rate. While in STARTUP, BBR sets the pacing rate and the congestion window to the BDP × 2/ln(2) , roughly doubling the bitrate each round-trip time.

1. When a BBR flow estimates the network pipe is full, it enters the DRAIN state to drain the queue built up during STARTUP. While in DRAIN, BBR reduces the pacing rate to Bmax \* ln(2)/2 , but keeps the congestion window high.BBR drains long enough to remove the built-up queue, then enters PROBE\_BW.
2. A long-lived BBR flow remains primarily in the PROBE\_BW

state, sending at the bottleneck rate, but repeatedly probing and attempting to fully utilize any additional network bandwidth, all while maintaining a small, bounded queue. PROBE\_BW does this by cycling, once per round-trip time, where the gain values are applied as multiples to the bottleneck rate. For example, when the gain is 1.25,BBR deliberately sends 25% more packets than the BDP for one round-trip time. If Bmax increases prior to this phase, the BDP, and thus overall sending rate, increases correspondingly. But if Bmax is unchanged, the gain of 0.75 in the subsequent phase drains any queue build-up caused by the previous higher gain.

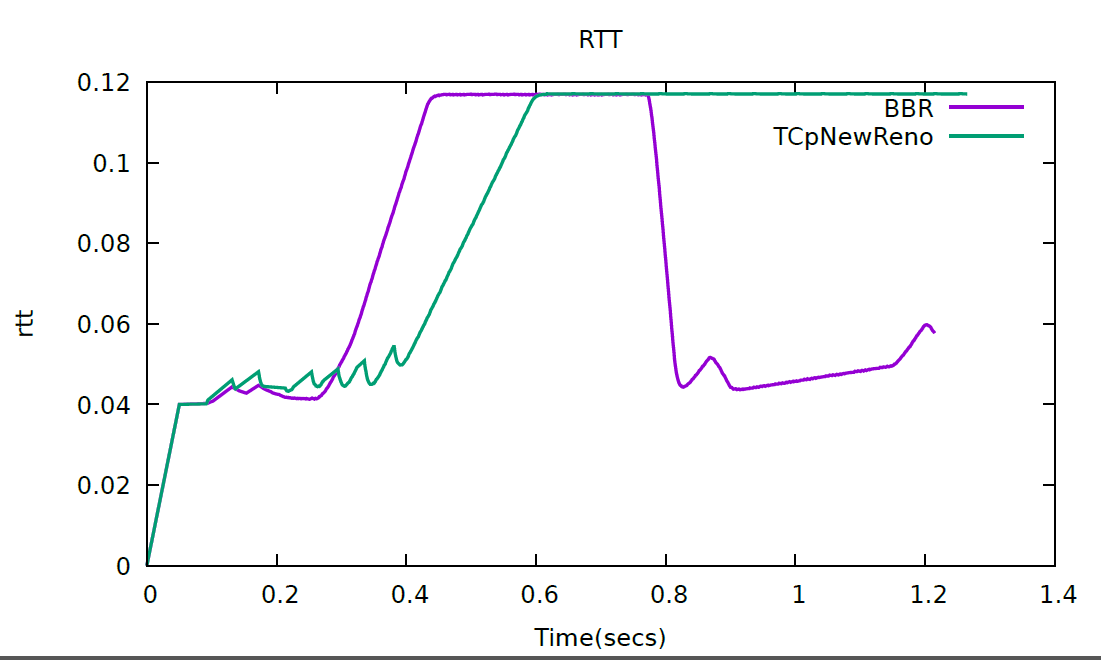
1. If BBR has not received an RTT sample that decreases the minimum round-trip time (Rmin) for 10 seconds, then it briefly enters the PROBE\_RTT state to quickly, greatly reduce (by 98%) the packets in flight in order to re-probe the path’s two-way propagation delay. The BBR flow stops probing after one round-trip time or 200 milliseconds, whichever is longer.
2. When a BBR flow exits the PROBE\_RTT state, if the full bandwidth estimate of the pipe has been reached, then it enters PROBE\_BW; otherwise, it enters STARTUP to try to refill the pipe.

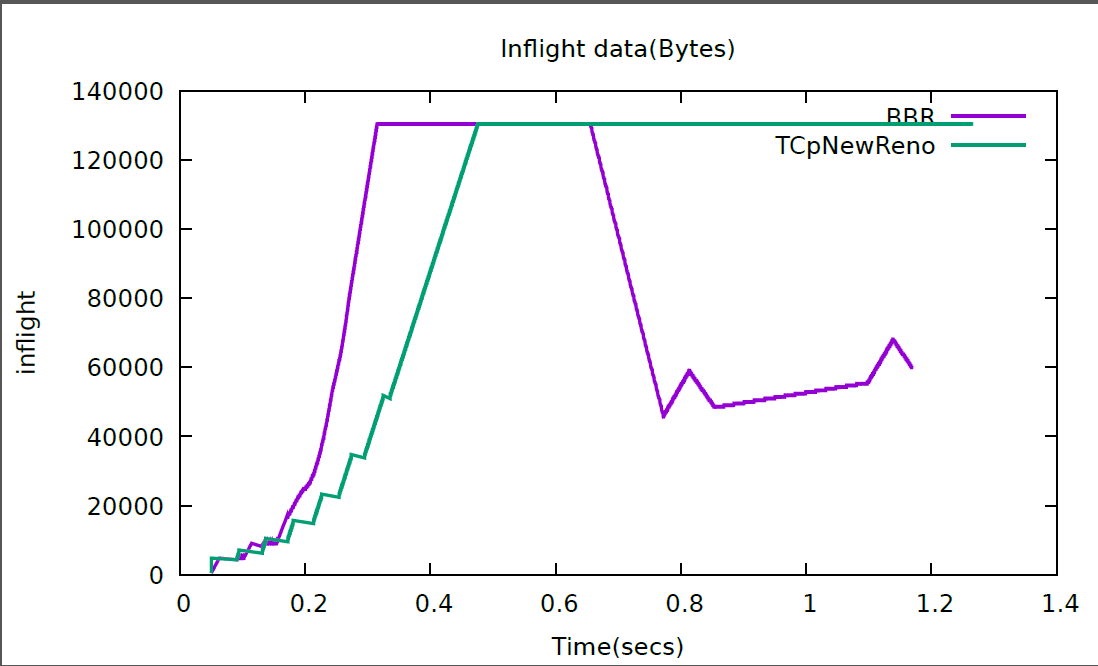
* **BBR vs TCP Reno:**
* *Congestion Window Comparison:*



The traditional TCP protocols’ Cwnd are higher than BBR; The purpose of BBR convergence is to actively reduce the pacing rate to avoid the buffer bloat problem, but the other TCP protocols are to fill the queue buffer space, which directly causes BBR to starve in the deep queue. Based on the above analysis, a problem has also arisen. In the traditional queue management algorithms, how to set queue size threshold and the packet loss rate for different versions of the TCP protocol is a major challenge. How to associate the packet loss rate with the queue size is an improvement direction. The change in queue size causes a large discrepancy in BBR.

* *RTT and Inflight Data Comparision:*

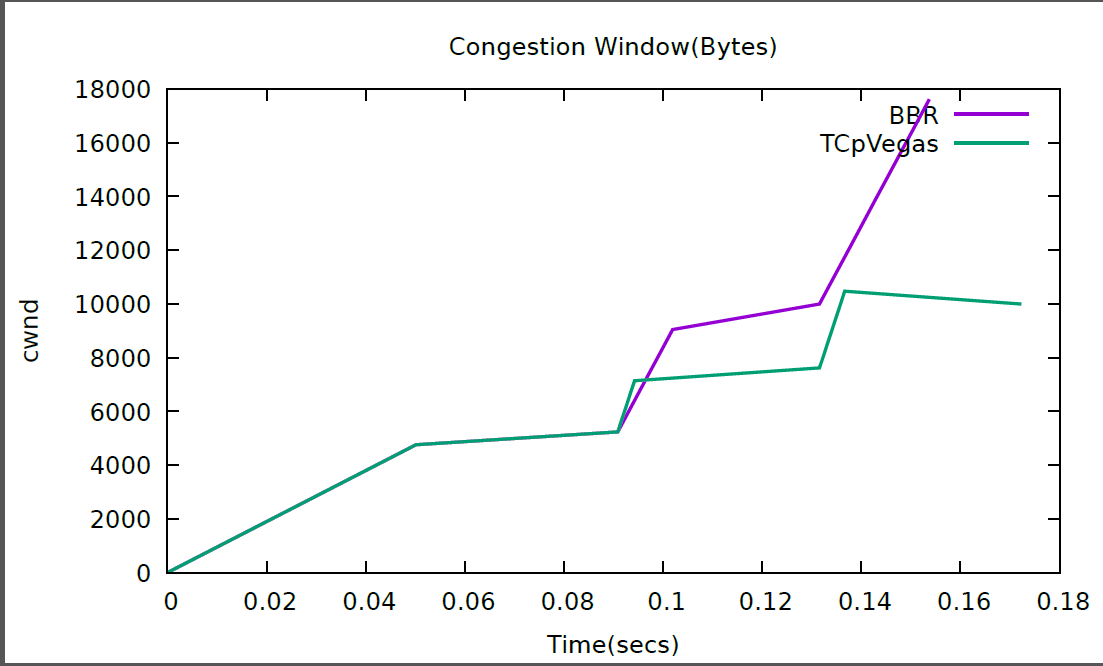


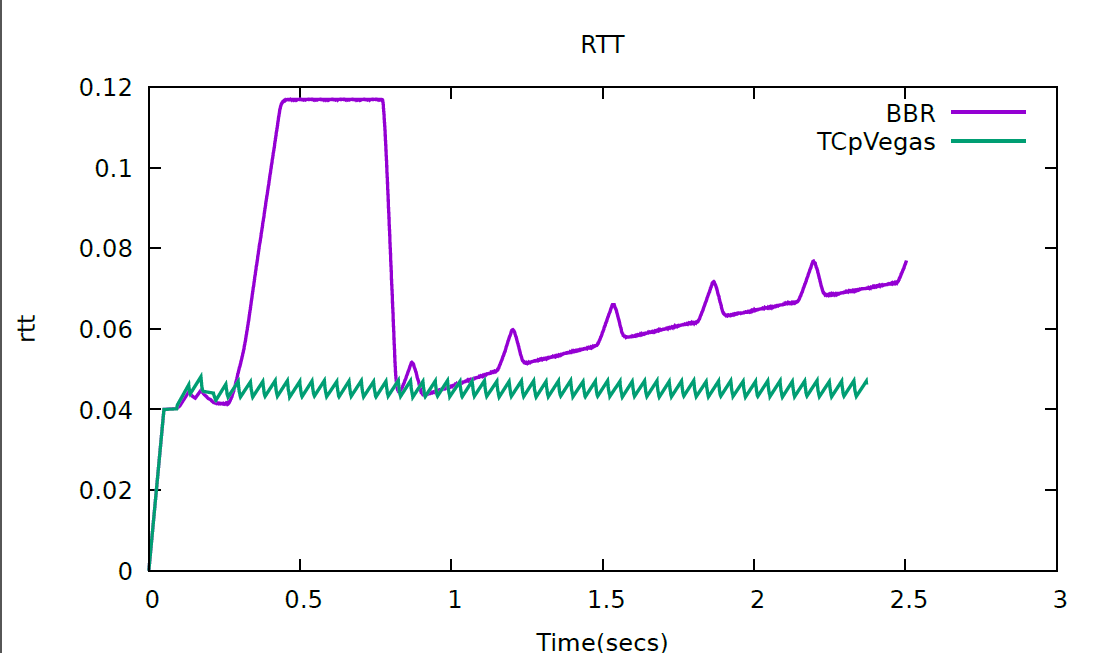


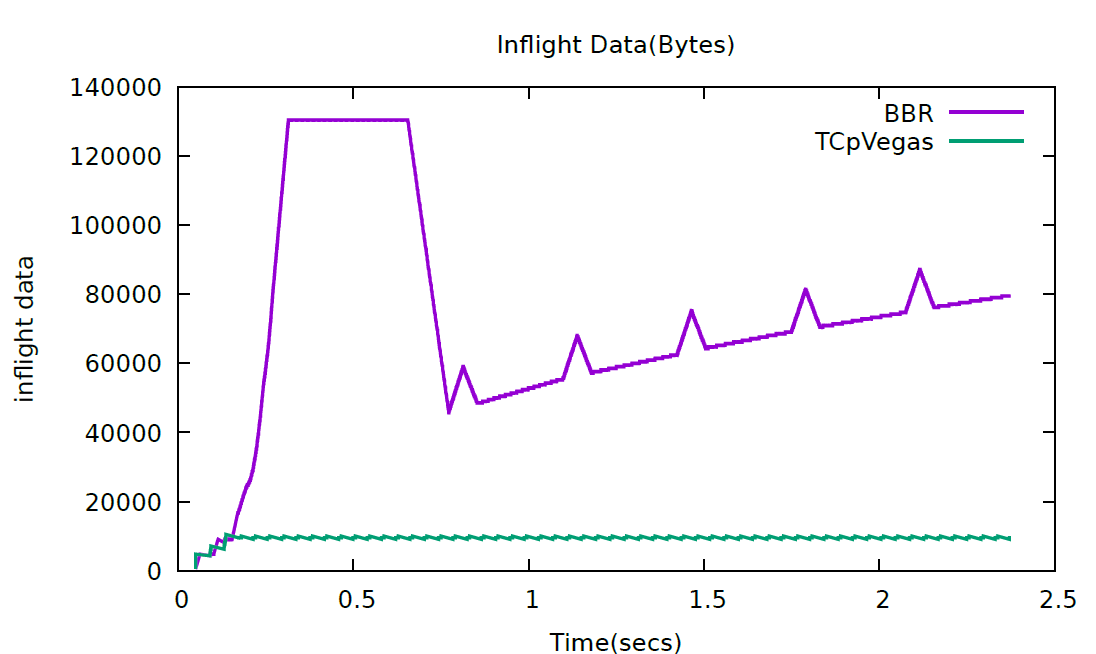
As seen in the figures the RTT value of BBR keeps on increasing until congestion happens once it recognises ,then it goes into the drain phase to free the buffer and then goes into the Probe\_bw stage on the other hand Reno unlike BBR gets it’s RTT increasing resulting in more congestion.

* **BBR vs TCP Vegas:**

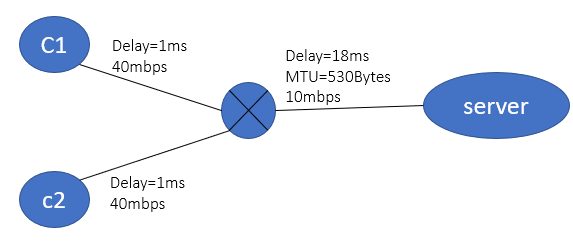
Similarly we have compared these performance analysis between TCP Vegas and BBR ,which have yielded the below results.







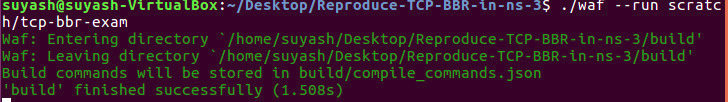
**Project Execution :**

* **Topology used:**

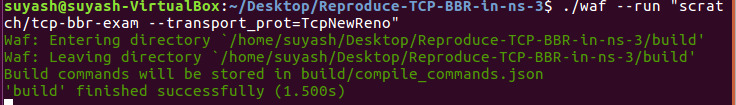
Here we have run different Congestion control algorithms separately, firstly we have executed TCP reno and then BBR and observed their performances (pcap files) and integrated their performance graph into one in order to visualise their outcomes in Wireshark as shown in previous pages (TCP Reno vs BBR graphs).

**Execution Steps:**

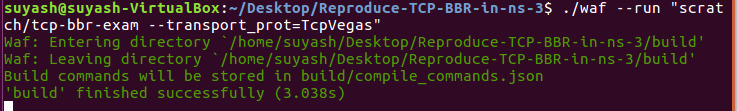
1. **Compile TCP BBR code using the below command:**

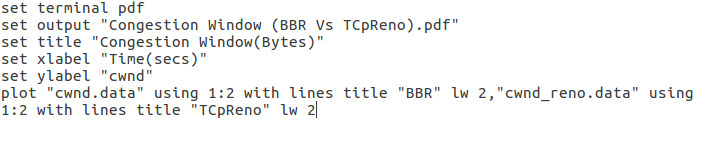


2. **Compile TCP New Reno code using below command:**



3. **Compile TCP Vegas code using below command:**

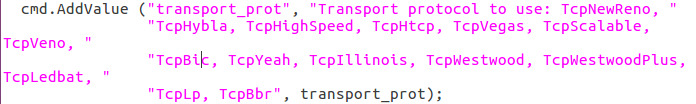
4. **Code for plotting the graph in gnu\_rtt.plt file:**



**5. Running the gnu plot command:**



**Various TCP variants available for comparisons**:



**Conclusion:**

We analysed and evaluated the performance of the TCP-BBR protocol based on the NS3. We use RTT, Congestion window, inflight data as the evaluation factors, and set up simulation environments. Firstly, we conducted a congestion window comparison experiment between TCP Reno and BBR, TCP Vegas and analysed the reasons for the high throughput of BBR. Secondly, we conducted the RTT and In-flight data analysis of BBR and other TCP protocols; we also understood the four stages of BBR mechanism.

**References:**

1. Hao Zhang, Halting Zhu, Yui Xia “Performance Analysis of BBR

Congestion Control Protocol Based on NS3” School of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing, China

1. Vivek Anand Jain “TCP-BBR-in-ns3”, GitHub source: ‘https://github.com/Vivek-anand-jain/Reproduce-TCP-BBR-in-ns-3’