PROJECT REPORT
ON
"TCP-BBR Congestion Control Algorithm"

Submitted in partial fulfillment of
Data Communication and Computer Networks Laboratory (CS3072)

Submitted by:

Sumit Kumar (122CS0299)
Anmol Agrawal (122CS0300)

NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA

Department of Computer Science and Engineering

Under the guidance of
Prof. Sanjeev Patel

Semester 6 | Academic Year 2024–2025

Problems with loss-based congestion control

2011: many reported excessive buffering and delays on the Internet (a.k.a. bufferbloat)

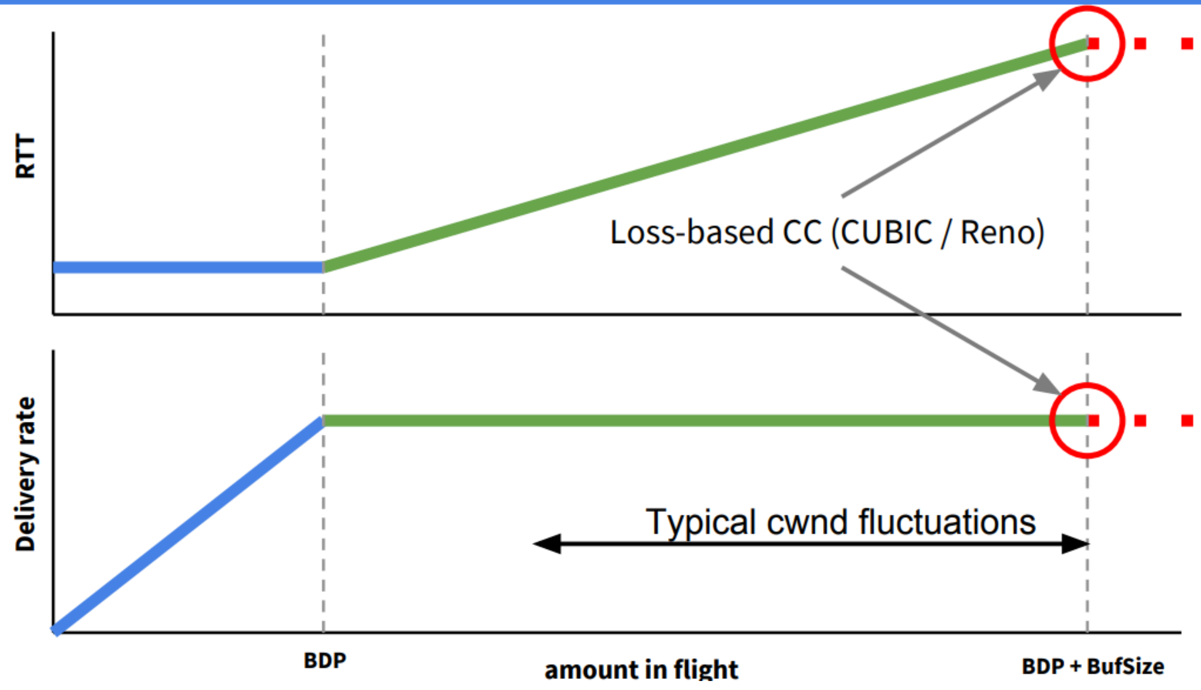
2012: single-connection HTTP/2 was much slower than multi-conn HTTP/1 on lossy links

2013: poor TCP throughput on WANs w/ commodity shallow-buffer switches

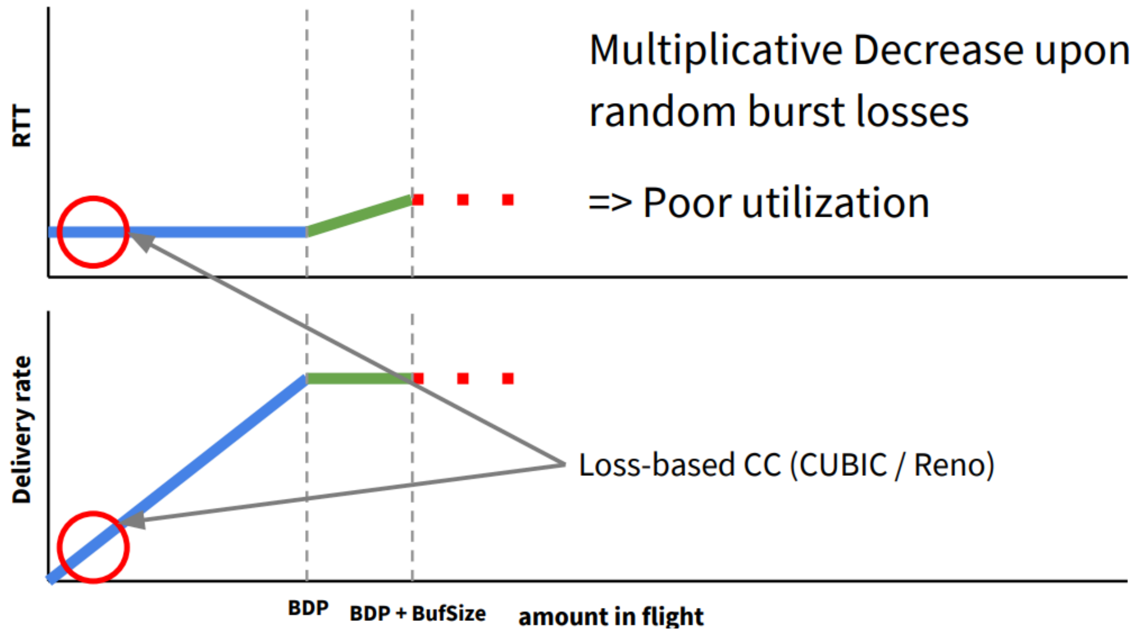
Culprit: loss-based congestion control (Reno, then CUBIC)

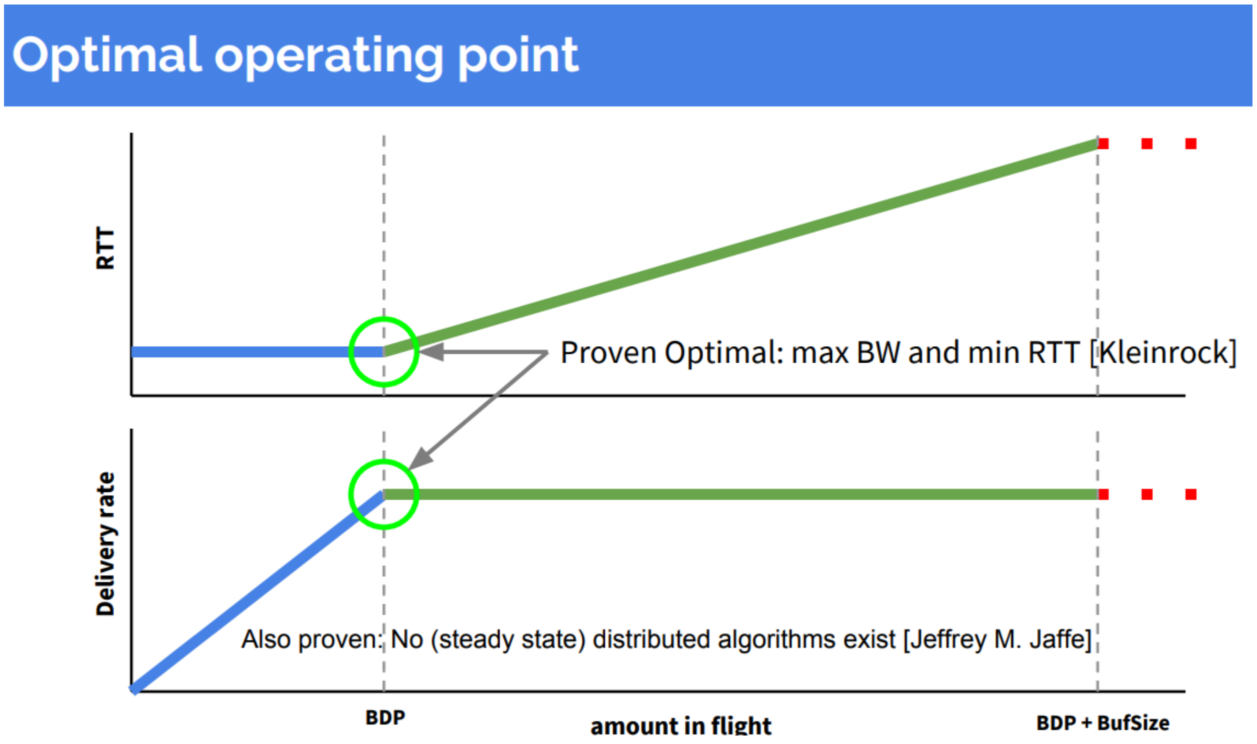
- Packet loss alone is not a good proxy to detect congestion
- Loss-based CC is overly sensitive to losses that come before congestion
    - 10Gbps over 100ms RTT needs <0.000003% packet loss
    - 1% loss over 100ms RTT gets 3Mbps
- Loss-based CC bloats buffers if loss comes after congestion

Loss-based congestion control in deep buffers



Loss-based congestion control in shallow buffers


Multiplicative Decrease upon random burst losses
=> Poor utilization
RTT
Delivery rate
Loss-based CC (CUBIC / Reno)
BDP
BDP + BufSize
amount in flight


Optimal operating point

BBR Congestion Control: Overview:

- BBR = Bottleneck Bandwidth and Round-trip propagation time
- BBR seeks high throughput with a small queue by probing BW and RTT
- Ground-up redesign of congestion control - Not loss-based, Not delay-based, Not ECN-based, Not AIMD-based
- Models the network path: probes and estimates max BW and min RTT
- Key insight: measure max BW and min RTT at different times
- Assume path properties are mostly stationary

Design : Overview

BBR is a model-based congestion control algorithm: its behavior is based on an explicit model of the network path over which a transport flow travels. BBR's model includes explicit estimates of **two parameters**:

1. **BBR.BtlBw**: the estimated bottleneck bandwidth available to the transport flow, estimated from the maximum delivery rate sample from a moving window.

2. **BBR.RTprop**: the estimated two-way round-trip propagation delay of the path, estimated from the the minimum round-trip delay sample from a moving window.

BBR uses its model to seek an **operating point with high throughput and low delay**. To operate near the optimal operating point, the point with maximum throughput and minimum delay, the system needs to maintain two conditions:

1. **Rate balance**: the bottleneck packet arrival rate equals the bottleneck bandwidth available to the transport flow.

2. **Full pipe**: the total data in in fight along the path is equal to the BDP(bandwidth-delay product or BBR.BtlBw * BBR.RTprop).

Control Parameters:

BBR uses its model to control the connection's sending behavior, to keep it near the target operating point. Rather than using a single control parameter, like the cwnd parameter that limits the volume of in-flight data in the Reno and CUBIC congestion control algorithms, BBR uses three distinct control parameters:

1**. pacing rate**: the rate at which BBR sends data.

2. **send quantum**: the maximum size of any aggregate that the transport sender implementation may need to transmit in order to amortize per-packet transmission overheads.

3. **cwnd**: the maximum volume of data BBR allows in-flight in the network at any time.

State Transition Diagram:

```
         |
         V
+---> Startup  ----+
|        |         |
|        V         |
|      Drain   ----+
|        |         |
|        V         |
+---> ProbeBW -----+
|      ^    |      |
|      |    |      |
|      +----+      |
|                  |
+---- ProbeRTT <---+
```

Github Link for code:

Tcp-bbr_github

Complete Procedure to run the tcp-bbr.cc:

1. **Save the File**: Save the above code as tcp-bbr.cc in the ns-2.35/tcp/ directory.

2. **Modify Makefile.in**: Add tcp-bbr.o to the list of object files in ns-2.35/Makefile.in. Find the line starting with OBJ_CC = and append:
tcp/tcp-bbr.o \

Then click ctrl+x to save and ctrl+y to exit

3.**Recompile NS-2**

```
    ->  cd ns-2.35
    ->  ./configure
    ->  make clean
```

NOTE: if ./configure doesn't run, run below command:

```
 ./configure
--with-tcl=/home/desktop/ns-allinone-2.35/tcl8.5.10
--with-tcl-ver=8.5.10
```

--with-tk=/home/desktop/ns-allinone-2.35/tk8.5.10
--with-tk-ver=8.5.10

4. **Run command**: make

5. Place the **bbr_test.tcl script**(in github repo(test.tcl)) in your ns-2.35 directory and run it:

```
cd ~/ns-allinone-2.35/ns-2.35
./ns bbr_test.tcl
```

This will execute the simulation and generate out.tr and out.nam.

6.**Launch the simulation:**

```
nam bbr_nam.nam
```

The nam tool will launch automatically to visualize the simulation.Use the out.tr file to analyze TCP-BBR's behavior (e.g., throughput, congestion window, RTT). You can process it with tools like awk, perl, or Python scripts

---

Conclusion:

In this project, we successfully implemented the TCP-BBR (Bottleneck Bandwidth and Round-trip propagation time) congestion control algorithm within the NS2 simulator. Our implementation was thoroughly tested and benchmarked against traditional TCP variants like TCP Reno. The comparative analysis using trace files and visualization tools like Xgraph demonstrated BBR’s efficiency in maintaining high throughput and low latency, especially under varying network conditions.

This project deepened our understanding of congestion control mechanisms, kernel-level protocol design, and simulation-based performance evaluation. It also provided hands-on experience with the NS2 architecture and the process of integrating and testing new transport protocols.

References:

- [1] “BBR Congestion Control,” IETF Draft: draft-cardwell-iccrg-bbr-congestion-control-00

- [2] Google Research Blog on BBR