UGP: TCP Cubic Vs BBR

Agrim Saharia
Arvendra Singh Kushwah

December 12, 2024

1 Introduction

The purpose of this undergraduate project is to evaluate the performance of two popular congestion control algorithms used in the TCP layer, Cubic and BBR, over WiFi 6. The project will start by examining the factors and characteristics of WiFi channels that affect the performance of congestion control algorithms. To accomplish this, the project will simulate various network topology scenarios, typical of daily life, and develop a theoretical basis for the observed results and will also suggest ways to mitigate the factors that affect the performance of WiFi channels. By comparing the Cubic and BBR algorithms, the project aims to provide insight into the strengths and weaknesses of each algorithm. NS-3, an open-source discrete event network simulator, will be used for all simulations in this project.

2 Background on TCP Congestion Control Algorithms

TCP is a popular transport protocol for transmitting data over the internet. TCP congestion control algorithms help to prevent network congestion and ensure that data is transmitted efficiently and reliably. There are two categories of TCP congestion control algorithms: delay-based and loss-based.

Delay-based algorithms, such as TCP Vegas and TCP BBR, use network delay (also known as round-trip time or RTT) as a proxy for network congestion. These algorithms aim to maintain a steady state in which the network delay remains constant. They achieve this by continuously monitoring the RTT and adjusting the transmission rate accordingly. If the RTT increases, the algorithm reduces the transmission rate to avoid further congestion, whereas if the RTT decreases, the algorithm increases the transmission rate to utilize available bandwidth.

On the other hand, loss-based algorithms, such as TCP Reno and TCP New Reno, use packet loss as an indicator of network congestion. These algorithms rely on the assumption that packet loss is an inherent sign of network congestion, and they aim to reduce the transmission rate when packet loss is detected. They use a variety of techniques, such as slow start and congestion avoidance, to regulate the transmission rate and maintain network stability.

2.1 TCP BBR

TCP BBR is a delay based congestion control algorithm that dynamically adjusts the congestion window based on the current state of the network and the goals of the algorithm. BBR uses four different phases to control congestion: startup, drain, probe bandwidth, and steady state. [2]

- **Startup phase:**
  - Congestion window (cwnd) is increased exponentially: $cwnd = cwnd \times 2$
  - When a congestion event is detected (timeout or dupACK), cwnd is reduced to half its value: $cwnd = cwnd/2$
- **Drain phase:**
  - Congestion window (cwnd) is reduced linearly: $cwnd = cwnd - 1$

- **Probe bandwidth phase:**
  - BBR uses a gain formula to calculate the amount by which to increase the congestion window. The gain formula is:

$$gain = \frac{BtlBw \times RTT + \sqrt{Var}}{BtlBw \times RTT}$$

    where:
    - $BtlBw$ is the estimated bottleneck bandwidth
    - RTT is the round-trip time of the network
    - Var is the estimated variance in the bottleneck bandwidth
  - The congestion window is then increased by the gain times the current cwnd: $cwnd = cwnd + gain \times cwnd$
- **Steady state phase:**
  - BBR uses a congestion window gain (Cwg) to adjust the congestion window based on the current queue length and the target queue length. The congestion window gain is calculated as:

$$Cwg = max\left(\frac{BtlBw \times RTT}{2 \times BDP}, 1\right)$$

    where:
    - BDP is the estimated bandwidth-delay product of the network
  - The target queue length is set to 1.5 packets.
  - The congestion window is then adjusted based on the ratio of the current queue length to the target queue length:
    - if queue length $<$ target length: $cwnd = cwnd + Cwg$
    - else: $cwnd = cwnd \times 0.75$

BBR uses these formulas to dynamically adjust the congestion window based on the current state of the network and the goals of the algorithm. The exact behavior of the algorithm may vary depending on the specific implementation and network conditions, but these formulas provide a general overview of the mathematical principles behind BBR congestion control.

2.2 TCP CUBIC

CUBIC is a loss based congestion control algorithm designed to provide high network utilization while maintaining low delay, typically in long fat networks. [3] CUBIC has mainly three regions:

- Initialization Phase: During this phase, the congestion window size (cwnd) is set to the initial value and the TCP connection enters the slow start phase.
- Slow Start Phase: Among the slow-start algorithms, CUBIC may choose the standard TCP slow start [RFC5681] in general networks, or the limited slow start [RFC3742] or hybrid slow start [HR08] for fast and long-distance networks. The slow start phase ends when the congestion window size exceeds the slow start threshold (ssthresh) value.
- Congestion Avoidance Phase: After the slow start phase, the TCP connection enters the congestion avoidance phase. In this phase, the congestion window size is increased using the cubic function, which is a concave curve that slows down as the congestion window size gets larger. The cubic function can be described by the following equation:

$$W_c(t) = C(t - K)^3 + W_{max} \tag{1}$$

where $W_c(t)$ is the congestion window size at time t, C is a constant that determines the rate of increase, W_{max} is the maximum congestion window size allowed, K is a constant that determines the inflection point of the curve and is defined as

$$K = \sqrt[3]{\frac{W_{max} \times (1 - beta_{cubic})}{C}}$$

where β_{cubic} is the CUBIC multiplication decrease factor, that is, when a congestion event is detected, CUBIC reduces its *cwnd* to $W_{cubic}(0) = W_{max} \times beta_{cubic}$

- Concave and Convex Regions: When receiving an ACK in congestion avoidance and if CUBIC is not in the TCP-friendly region then if $cwnd \geq W_{max}$, then CUBIC is in the convex region, else CUBIC is in the convex region. In these regions, *cwnd* MUST be incremented by $\frac{W_c(t+RTT)-cwnd}{cwnd}$ for each received ACK, where $W_c(t + RTT)$ is calculated using the cubic equation given above.

- TCP-Friendly Region: Standard TCP performs well in certain types of networks, for example, under short RTT and small bandwidth (or small BDP) networks. In these networks, we use the TCP-friendly region to ensure that CUBIC achieves at least the same throughput as Standard TCP. The average congestion window size of AIMD($alpha_{aimd}$, $beta_{aimd}$) is calculated using

$$AVG_{W_{aimd}} = \frac{2 \times alpha_{aimd} \times (1 + beta_{aimd})}{2 * (1 - beta_{aimd} \times P)}$$

CUBIC uses below equation to estimate the window size W_{est} of AIMD($alpha_{aimd}$, $beta_{aimd}$) with $alpha_{aimd} = 3 * (1 - beta_{cubic})/(1 + beta_{cubic})$ and $beta_{aimd} = beta_{cubic}$, which achieves the same average window size as Standard TCP.

$$W_{est}(t) = W_{max} \times beta_{cubic} + \frac{3 \times (1 - beta_{cubic})}{(1 + beta_{cubic})} \times \frac{t}{RTT}$$

When receiving an ACK in congestion avoidance, if $W_{cubic}(t) < W_{est}(t)$, then $cwnd = W_{est}(t)$ at each reception of an ACK.

- Fast Recovery Phase: If a packet loss is detected, the TCP connection enters the fast recovery phase. In this phase, the congestion window size is reduced to half of its current value (but not below ssthresh) and the TCP connection enters the fast retransmit phase to recover the lost packet. Once the lost packet is recovered, the TCP connection returns to the congestion avoidance phase.

3 Background on WiFi

Wireless Fidelity (WiFi) is a wireless networking technology that enables devices to communicate with each other and connect to the internet without the need for physical cables. WiFi is based on the IEEE 802.11 family of standards, which specify the protocols for wireless local area networks (WLANs).

WiFi operates in the unlicensed radio frequency (RF) bands, typically 2.4 GHz and 5 GHz. The available bandwidth in these bands is divided into channels, with each channel being a specific frequency range. The channels are then used by WiFi devices to transmit and receive data.

The project will focus on two versions of WiFi: WiFi 802.11n (WiFi 4) and the latest version, WiFi 802.11ax (WiFi 6)."

3.1 WiFi 802.11n

- Increased data rates up to 600 Mbps.

- Support for Multiple Input Multiple Output (MIMO) technology, which allows for multiple antennas to be used for transmitting and receiving data alloing an access point to communicate with a single device using multiple spatial streams.

- Use of wider channel bandwidths (up to 40 MHz) to increase throughput.
- Support for channel bonding, where multiple channels can be combined to create a wider channel and increase throughput.

3.2 WiFi 802.11ax

- Increased data rates up to 9.6 Gbps.
- Support for Multi-User Multiple Input Multiple Output (MU-MIMO) technology, which allows an access point to communicate with multiple devices (upto 8) simultaneously. The access point divides its available bandwidth into multiple spatial streams and transmit data to different devices using different streams.
- Support for Orthogonal Frequency Division Multiple Access (OFDMA) technology, which divides each channel into sub-channels called Resource Units (RUs). This allows for more efficient use of the available bandwidth and better support for multiple devices.
- Improved performance in high-density environments using features such as Target Wake Time (TWT), which reduces power consumption and improves performance. TWT allows devices to schedule their transmissions to avoid collisions and reduce the amount of time they spend in active mode.
- Support for 160 MHz channel bandwidths to further increase throughput.

Both versions utilizes Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) as its medium access control protocol.

Overall, the combination of TWT, MU-MIMO, and OFDMA in Wi-Fi 6 improves the fairness and efficiency of the network, making it a better choice for environments with multiple nodes compared to previous Wi-Fi standards such as 802.11n.

3.3 Packet drops in Physical Layer of WiFi

The Packet drop in physical layer of WiFi can happen due to various reasons such as: [1]

- Signal strength: If the signal strength of a received packet is too low, the PHY layer may decide to drop the packet.
- Interference: If the PHY layer is receiving multiple signals at the same time, it may choose to drop packets that are difficult to distinguish from the interference.
- Bit errors: If the PHY layer detects too many errors in the received packet, it may choose to drop the packet.
- Collision: If two or more packets are transmitted at the same time and collide, the PHY layer may choose to drop both packets.
- Channel switching: If the PHY layer switches channels during the reception of a packet, the packet may be dropped.
- Physical layer limitations: Some PHY layers may have limitations in terms of the number of packets they can receive and process at the same time, and may drop packets if they exceed this limit.

In the simulation being conducted, channel switching will not be required as only one access point is being used. Additionally, the physical layer transmission capacity has been set to match that of the point-to-point link. As a fixed signal strength model has been employed, the receiver node will consistently receive a signal strength of -80db. Hence signal strength will not be responsible for any packet drops.

The majority of packet drops in the simulation are expected to occur due to interference, bit errors, and collisions. In the case of a single pair of sender-receiver nodes and access point (AP), there will be very little contention, except when the node and AP are trying to transmit simultaneously. Therefore, most packet drops are expected to result from bit errors.

4 Simulations

4.1 About NS3

NS-3 is an open-source discrete-event network simulator that is widely used for network simulation and research purposes. It allows researchers and developers to create simulations of various network protocols, applications, and scenarios to study their performance and behavior.

NS-3 simulation is typically written in C++ and it is event-driven. The simulation is composed of a set of objects that represent network devices, protocols, and applications. These objects interact with each other by sending events that are processed by the NS-3 event scheduler.

The event scheduler is responsible for scheduling events at specific times and executing them in a specific order. Events can be triggered by network activity, timers, or other events, and they can cause changes to the simulation state, such as packet transmission or reception, routing table updates, or application events.

NS-3 also provides a comprehensive set of network models, including models for link, network, transport, and application layers. These models can be customized or extended to support new protocols and scenarios.

NS-3 simulations can be visualized using the NetAnim animation tool, which provides a graphical representation of the simulated network topology and the traffic flowing through it. Additionally, NS-3 supports integration with external tools, such as Wireshark, to capture and analyze network traffic.

4.1.1 Models used in simulation

- PointToPointHelper: The ns-3 point-to-point model is of a very simple point to point data link connecting exactly two PointToPointNetDevice devices over an PointToPointChannel. This can be viewed as equivalent to a full duplex RS-232 or RS-422 link with null modem and no handshaking.
- BulkSendHelper: This traffic generator simply sends data as fast as possible up to MaxBytes or until the application is stopped (if MaxBytes is zero). Once the lower layer send buffer is filled, it waits until space is free to send more data, essentially keeping a constant flow of data.
- YansWifiChannelHelper: The intent of this class is to make it easy to create a channel object which implements the yans channel model. The yans channel model is described in "Yet Another Network Simulator"
- YansWifiPhyHelper: Make it easy to create and manage PHY objects for the yans model.
- ConstantRateWifiManager: Allows users to set constant rates for data and control wifi transmissions

4.2 P2P Simulations

The simulation is based on the dumbbell topology, comprising of two interconnected routers connected via a bottleneck link. Each end router is connected to an equal number of sender and receiver nodes via point-to-point (P2P) links. The programmed behavior of the sender nodes involves transferring bulk data to their corresponding receivers, while each receiver sends acknowledgments for every received data packet. The accompanying figure provides the link delay and data rate values for each link.

The simulation will be divided into two parts, with one part using TCP BBR congestion control algorithm and the other using TCP CUBIC congestion control algorithm. A sender node and receiver node will be utilized in both parts to monitor the congestion window scaling and goodput of the sender node. In each part, both a single sender-receiver pair and four sender-receiver pairs will be tested.

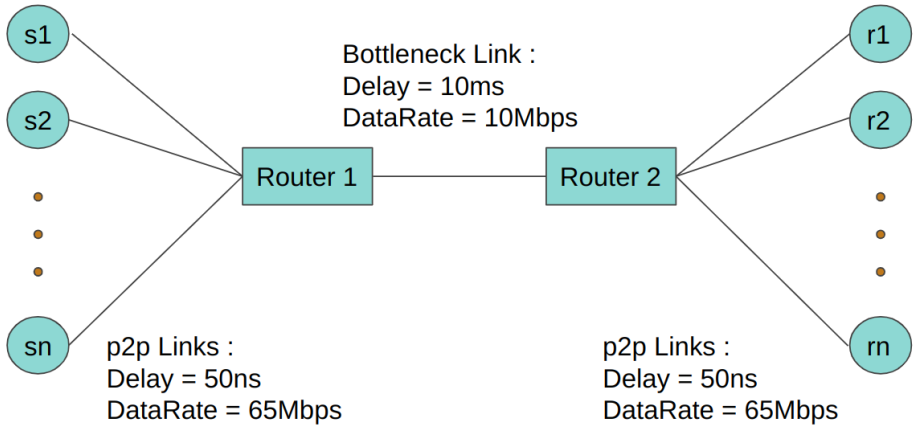
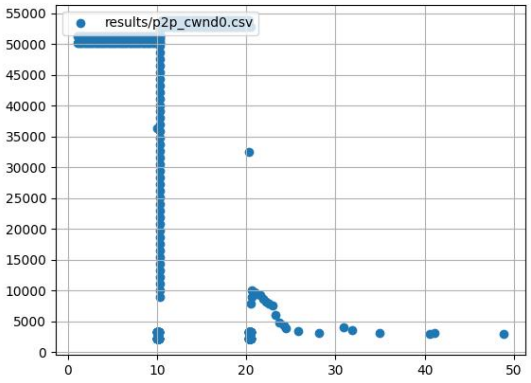


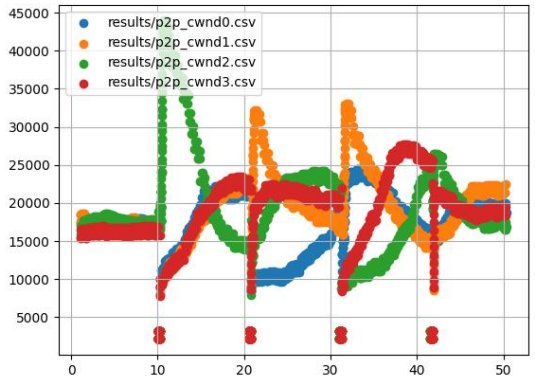
Figure 1: Simulation Topology

4.2.1 TCP BBR

For this segment of the simulation, TCP BBR congestion control algorithm will be employed.



(a) Using one sender node

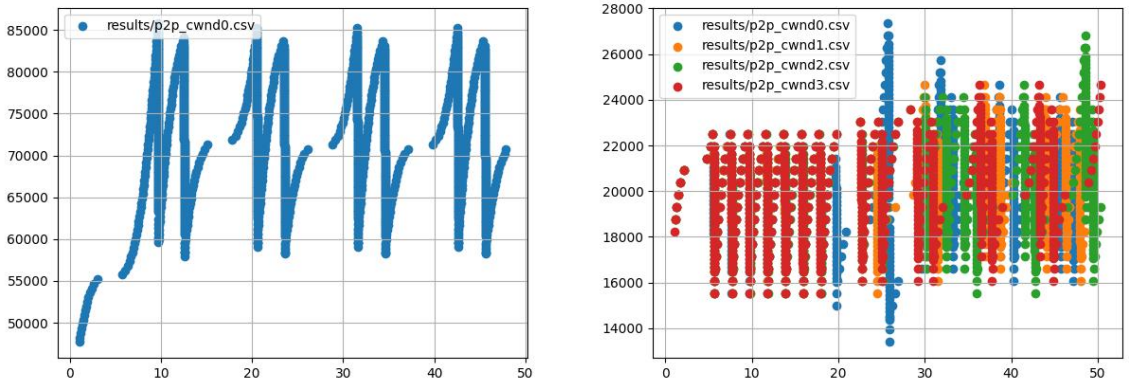


(b) Using four sender nodes

Figure 2: TCP BBR with P2P

4.2.2 TCP CUBIC

For this segment of the simulation, TCP CUBIC congestion control algorithm will be employed.



(a) Using one sender node

(b) Using four sender nodes

Figure 3: TCP CUBIC with P2P

4.3 WiFi Simulations

The simulation is based on a dumbbell topology, which consists of two routers connected by a bottleneck link. One of the routers serves as an access point, and all sender nodes transfer data wirelessly through it, while the other router connects to the receivers via a point-to-point (P2P) link. The sender nodes are programmed to transfer bulk data to their corresponding receivers, and the receivers send acknowledgments for each received packet. The accompanying figure provides the link delay and data rate values for each link.

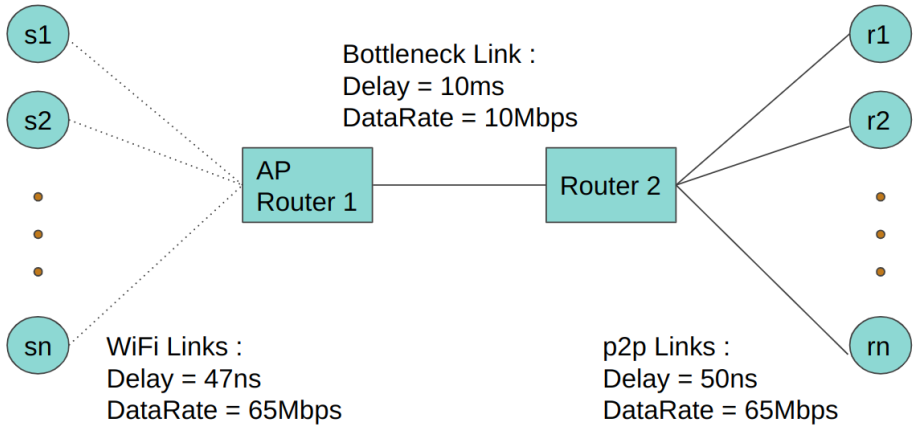
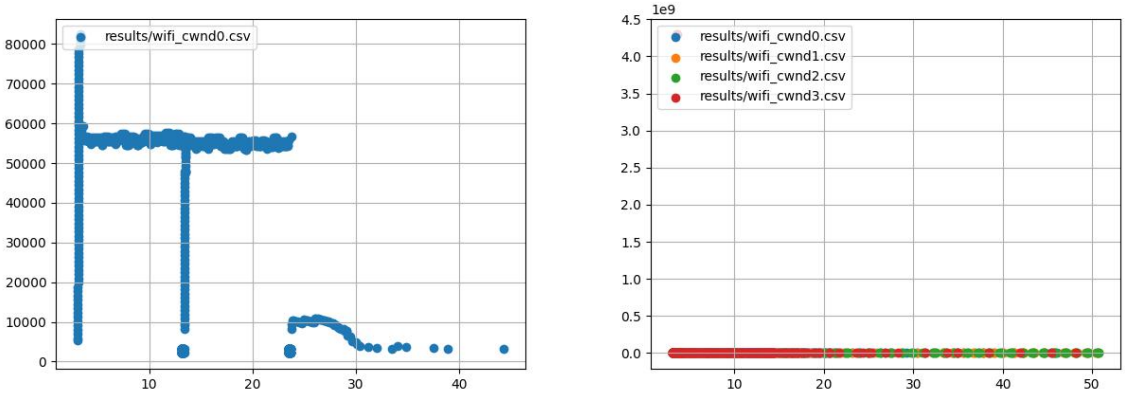


Figure 4: Simulation Topology

Similar to the P2P simulation, this simulation will also be divided into two parts. The first part will utilize TCP BBR congestion control algorithm, while the second part will use TCP CUBIC congestion control algorithm. For both parts, the congestion window scaling and goodput of the sender node will be tracked for both a single sender-receiver pair and four sender-receiver pairs.

4.3.1 TCP BBR

For this segment of the simulation, TCP BBR congestion control algorithm will be employed.



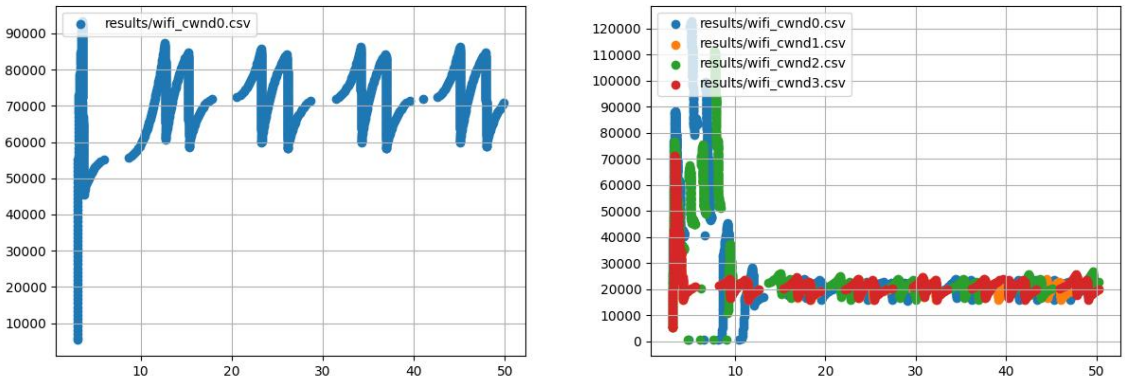
(a) Using one sender node

(b) Using four sender nodes

Figure 5: TCP BBR with WiFi

4.3.2 TCP CUBIC

For this segment of the simulation, TCP CUBIC congestion control algorithm will be employed.



(a) Using one sender node

(b) Using four sender nodes

Figure 6: TCP CUBIC with WiFi

5 Analysis

5.1 Issues with BBR implementation in ns3

As we observed in cases of Tcp-BBR, the congestion window faces issues while scaling in our simulation. The bottleneck bandwidth and delay are not properly being estimated by BBR, leading to reduced efficiency in terms of goodput. However, using other datarates and delay parameters for the links can lead to optimal scaling, the reason for which, is not clear.

5.2 Scaling of Tcp Cubic

Form our observations, we see that TcpCubic is fairly scaling the congestion window in all cases. The BDP of our link is $BDP_{bottleneck} = 30 * 1024 * 1024 * 0.01 = 314572.8$ bits.
In case of single flows, we observe similar scaling measures in P2P and WiFi links. The congestion window scaled in TcpCubic can be seen to be in the range of $[60000 - 85000]$ bits, which is decently adequate.
In case of four flows, we again observe similar scaling measures in P2P and WiFi links. On both links, TcpCubic performs really well in terms of intraprotocol fairness, as the scaled congestion window is around 20000 bits, which is slightly better than the case of single flow as $20000 * 4 = 80000$, which is near the upper-bound of its scaled congestion window.
The goodput achieved in case of P2P links with single flow is $\frac{8.65102}{10} = 86.51\%$ of the bottleneck bandwidth.

5.3 Differences in P2P vs WiFi Links

In case of Tcp Cubic, we observed higher goodput in case of using P2P links on the sender nodes as compared to WiFi links. In case of single flow, P2P links performed $\frac{8.65102-8.12672}{8.12672} = 6.452\%$ better than WiFi links. In case of four flows, P2P links performed $\frac{8.73617-8.16054}{8.16054} = 7.054\%$ better than WiFi links.

This improvement in performance in case of four flows can be attributed to the fact that WiFi links introduce more signal interference as the number of flows increase. We observed a $\frac{0.00105984-0.000146626}{0.000146626} = 622.819\%$ increase in ratio of packet drops on AP in case of four flows. More frequent packet drops on the physical layer reduces the overall efficiency of transmission.

References

[1] Ammar Mohammed Al-Jubari, Mohamed Othman, Borhanuddin Mohd Ali, and Nor Asilah Wati Abdul Hamid. Tcp performance in multi-hop wireless ad hoc networks: challenges and solution. *EURASIP*, 14(3):1–25, 2011.

[2] Neal Cardwell, Yuchung Cheng, Soheil Hassas Yeganeh, Ian Swett, and Van Jacobson. BBR Congestion Control. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-02, Internet Engineering Task Force, March 2022. Work in Progress.

[3] Injong Rhee, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. CUBIC for Fast Long-Distance Networks. RFC 8312, RFC Editor, February 2018.