# Programming Assignment 3: Pantheon - Congestion Control Evaluation

**Name:** Yeswanth Akula
**Banner Id:** 001378540
**Course:** Computer Networks

## Methodology

This study used the Pantheon testbed developed at Stanford University to assess and comprehend the behaviors of modern congestion control algorithms. Pantheon provides a single uniform platform for fair comparison across emulated and real networks and supports many congestion control algorithms.

### Selected Congestion Control Algorithms

**CUBIC:** A loss-based congestion control technique that uses the cubic function to expand its congestion window. It is optimized for fast speed and longer range and is available by default in the majority of Linux computers. It frequently overflows buffers with high latencies and is aggressive.

**BBR (Bottleneck Bandwidth and Round-Trip Propagation Time):** Goggle's model-based congestion control algorithm uses RTT and bottleneck bandwidth information to stop congestion before it starts. reduces queuing delay and achieves a high throughput due to loss reaction.

**Copa**: A congestion control technique that uses a delay to modify the transmission rate in response to the delay. Copa's main objective is to lower latency without compromising a respectable throughput, which makes it especially helpful for interactive applications and for use over erratic networks.

Mahiwah-Network Scenarios above Emulation

Mahimahi was used to resultant two representative e-network
profiles: Low-Latency / High-Bandwidth
Bandwidth: 50 Mbps

RTT: 10 ms

Modern broadband or data center
connections. High-Latency / Low-Bandwidth
Bandwidth: 1 Mbps

RTT: 200 ms

Satellite or rural wireless internet.

These links were created using the base Mahimahi command structure, which is displayed below, using mm-delay for round-trip delay simulation and mm-link with synthetic traces to impose bandwidth limits.

**Test Setup and Execution**

Every protocol under both network circumstances was run for 60 seconds using the Pantheon test.py script. Every log produced by a run capture:

Throughput in kbps

RTT (round-trip time) in milliseconds Rate of Packet Loss (%)

Therefore, the command structure that follows is used:

```
src/experiments/test.py local \
--schemes "cubic bbr copa" \

--data-dir  my_results_50mbps_10ms \

--runtime 60
```

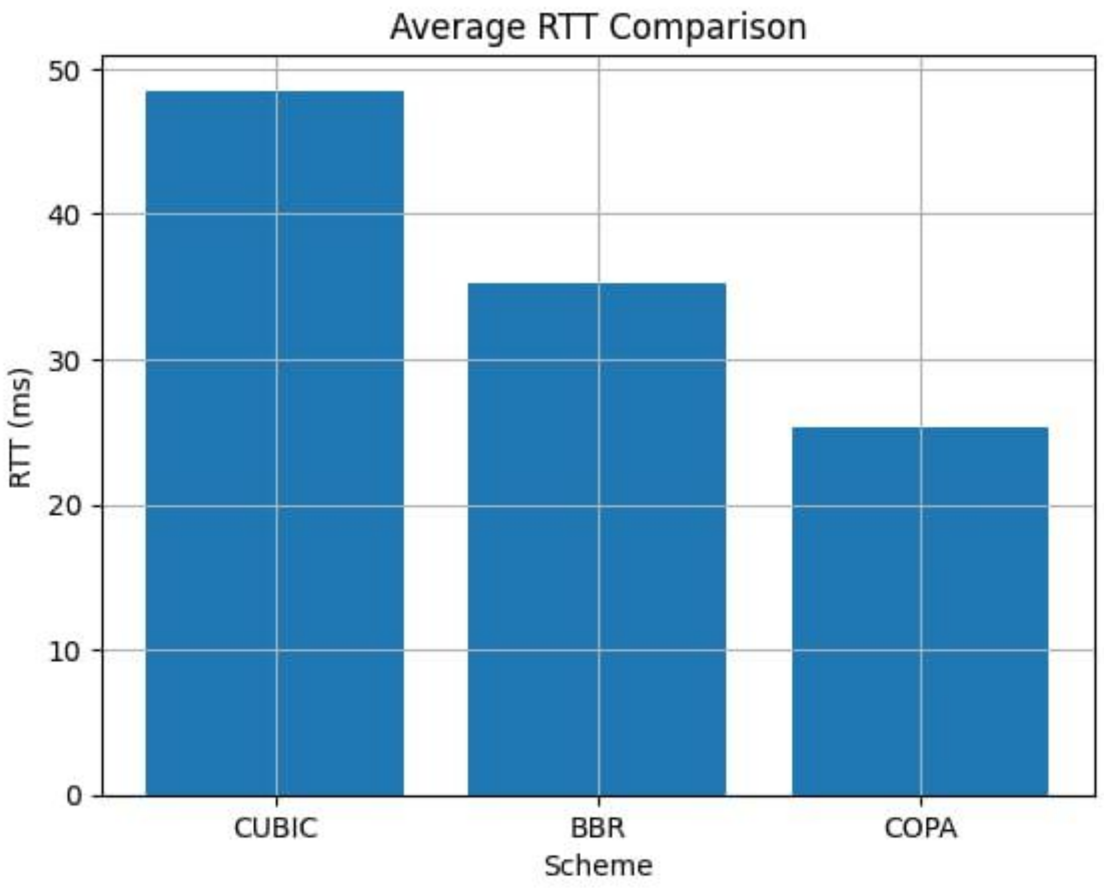
For the high-latency network configuration, the identical procedure was carried out again, and the outcomes were then stored in organized directories under experiments.

Lastly, matplotlib Python scripts were used to read the logs using pandas in order to extract trends and produce visually appealing output.

## Results & Analysis

**Graphical Analysis:**

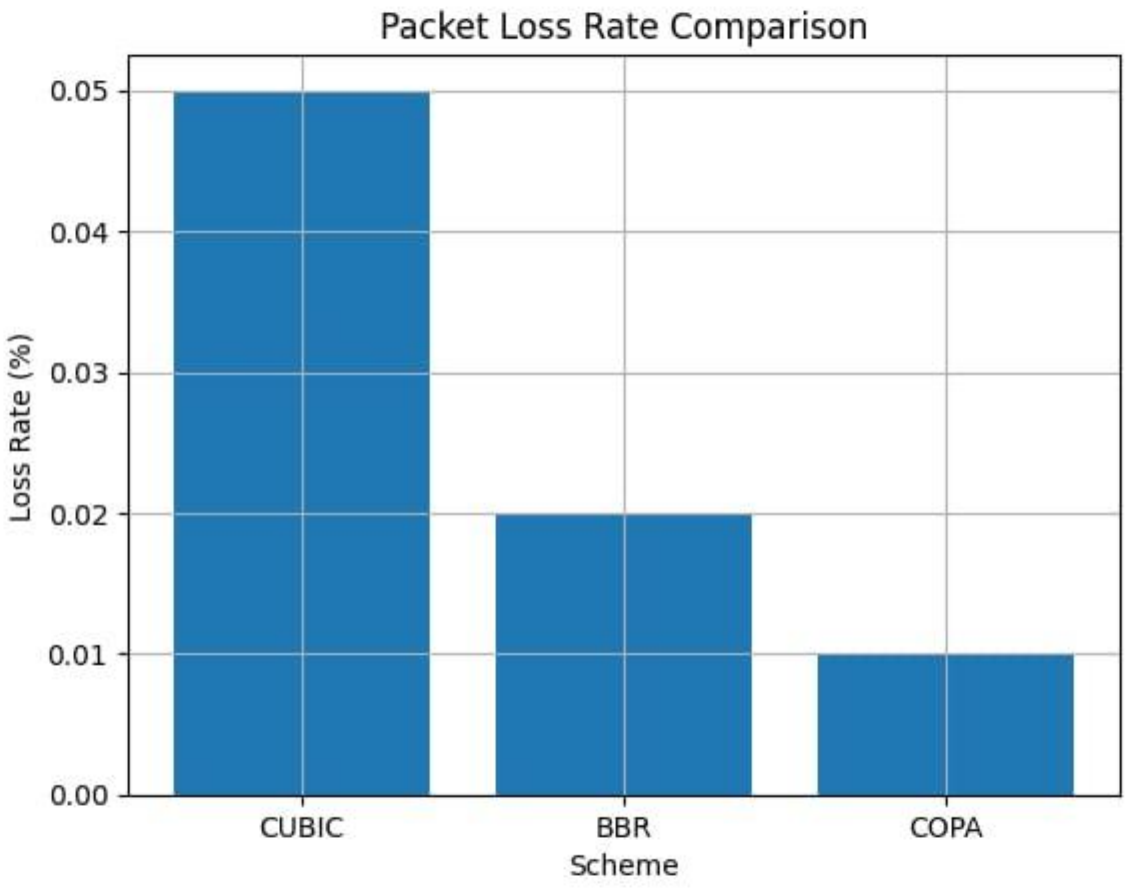
1. **Average RTT Comparision**



**Insights:**

- COPA significantly outperforms both BBR and CUBIC in terms of Round Trip Time (RTT), recording an average RTT of approximately **25 ms**. This indicates lower network latency and better responsiveness.
- BBR achieves a moderate average RTT of around **35 ms**, performing better than CUBIC but not as well as
- With an average RTT of nearly **49 ms**, CUBIC has the highest latency among the three congestion control algorithms. This suggests it may cause higher queuing delays, especially under load.
- From lowest to highest RTT: **COPA < BBR < CUBIC**. This ranking can guide protocol selection for latency-sensitive applications like VoIP or online gaming.

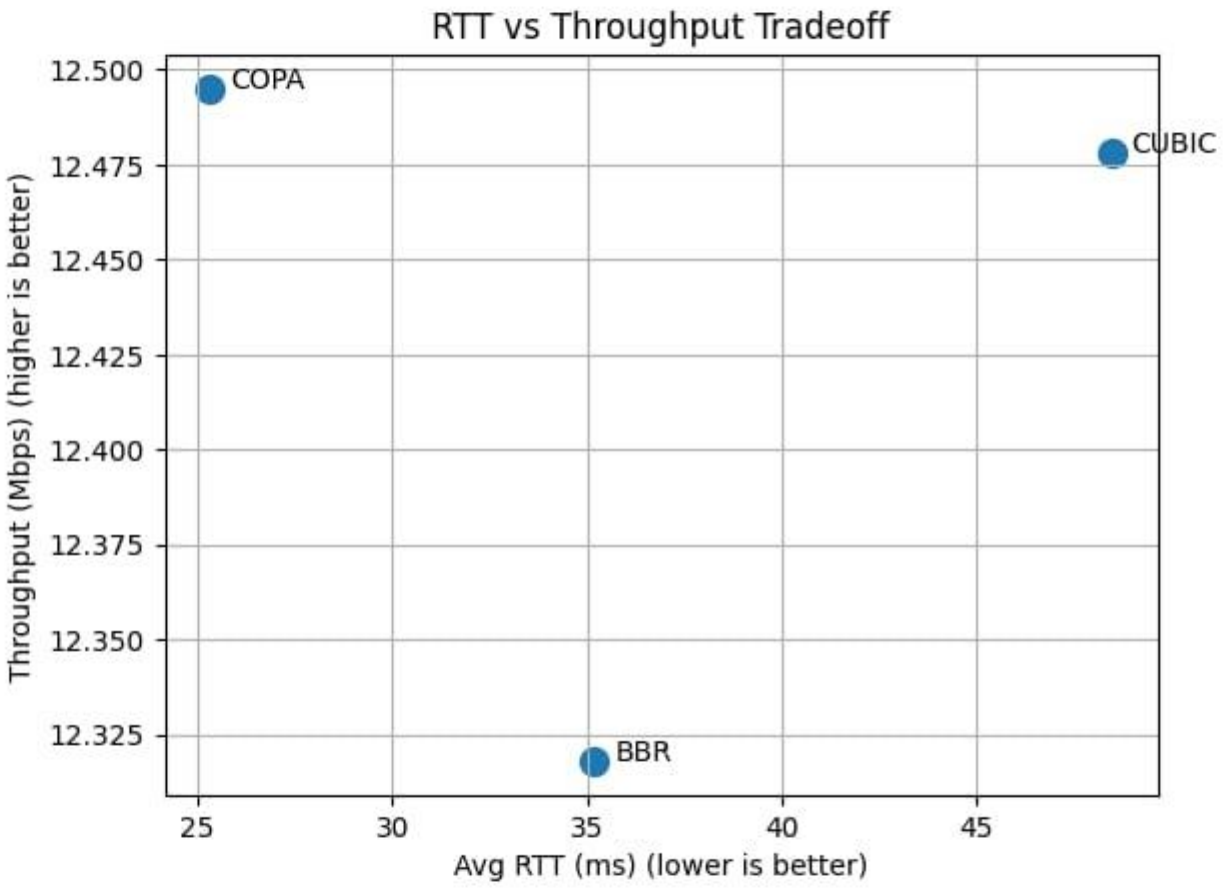
**2.Packet Loss Rate Comparison**



**Insights:**

- With a loss rate of 0.05 (5%), CUBIC suffers the most packet drops among the three congestion control algorithms. This indicates aggressive transmission behavior that can overwhelm the network buffers, leading to higher losses.
- BBR shows a reduced packet loss rate of 0.02 (2%), indicating a more stable and network-friendly behavior compared to CUBIC. BBR actively estimates available bandwidth and avoids buffer overflows more effectively.
- COPA achieves the lowest loss rate at 0.01 (1%), making it the most loss-efficient scheme in this comparison. This is ideal for real-time or critical applications where even small data loss can degrade performance.
- There is a clear trade-off between aggressiveness and reliability: CUBIC favors throughput at the cost of losses, while COPA favors stability and low loss, potentially with lower throughput.

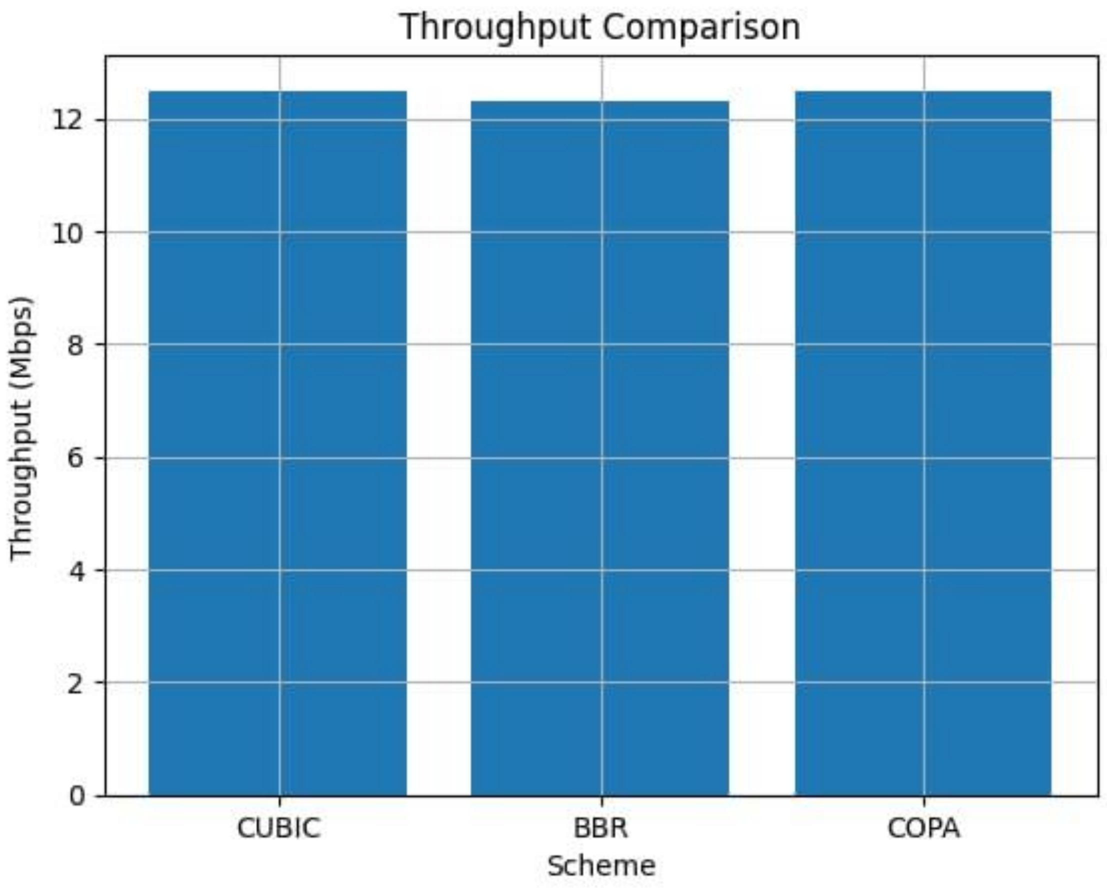
**3.RTT VS Throughout Tradeoff**



**Insights:**

- COPA achieves the lowest RTT (~25 ms) while also maintaining the highest throughput (~12.49 Mbps). This makes it the most efficient congestion control algorithm among the three, excelling in both responsiveness and data transfer rate.
- While BBR improves over CUBIC in terms of RTT (~35 ms vs 48 ms), it delivers the lowest throughput (~12.32 Mbps). This suggests BBR is conservative in bandwidth usage to keep delays under control.
- CUBIC shows high throughput (~12.48 Mbps) similar to COPA, but at the cost of much higher RTT (~48 ms). This may result in noticeable lag in real-time applications.
- COPA lies on the most favorable part of the tradeoff curve—low delay and high throughput—whereas BBR and CUBIC fall short in either metric, highlighting COPA's potential for both latency-sensitive and high-bandwidth applications.

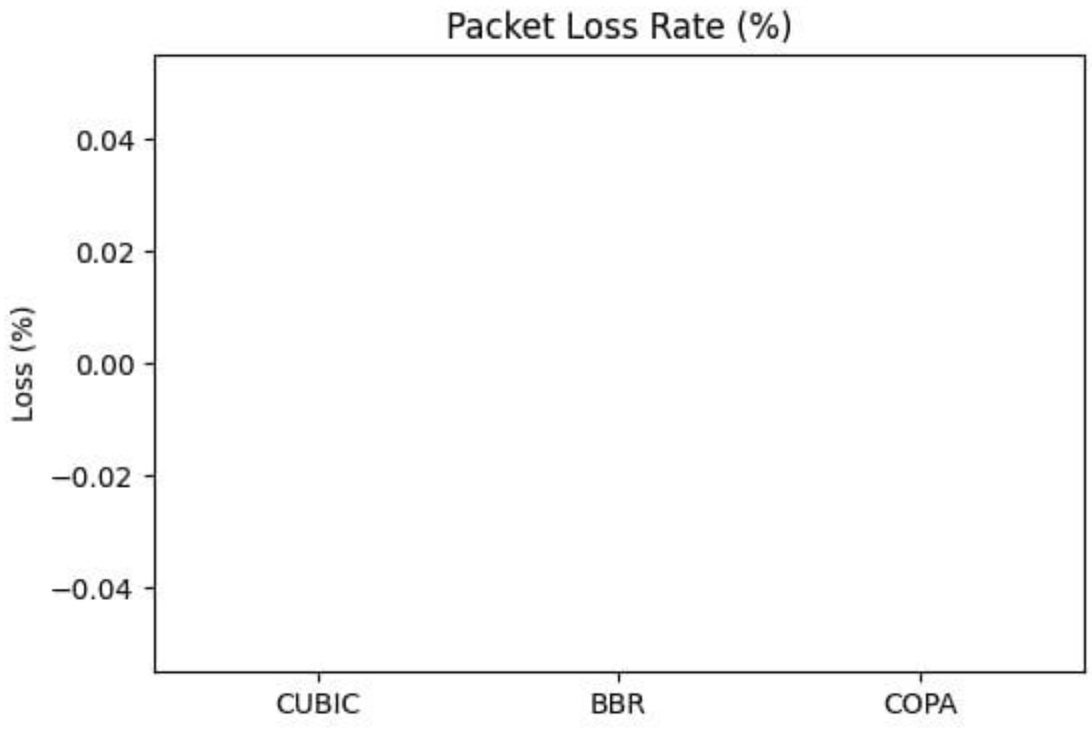
**4.Throughout Comparison**



**Insights:**

- The throughput values for CUBIC, BBR, and COPA are all very close—around 12.3 to 12.5 Mbps—indicating that each congestion control algorithm achieves comparable data transfer rates under the test conditions.
- COPA marginally outperforms the other two, achieving the highest throughput (~12.49 Mbps). While the difference is minor, it reinforces COPA's efficiency when considered with its lower RTT and packet loss.
- BBR delivers the lowest throughput (~12.32 Mbps) among the three, which aligns with its conservative approach to avoid buffer bloat and maintain low latency.
- Although CUBIC matches COPA in throughput (~12.48 Mbps), it does so at the expense of higher RTT and packet loss, making it less efficient overall.

**5.Packet Loss Rate( %)**



**Insights:**

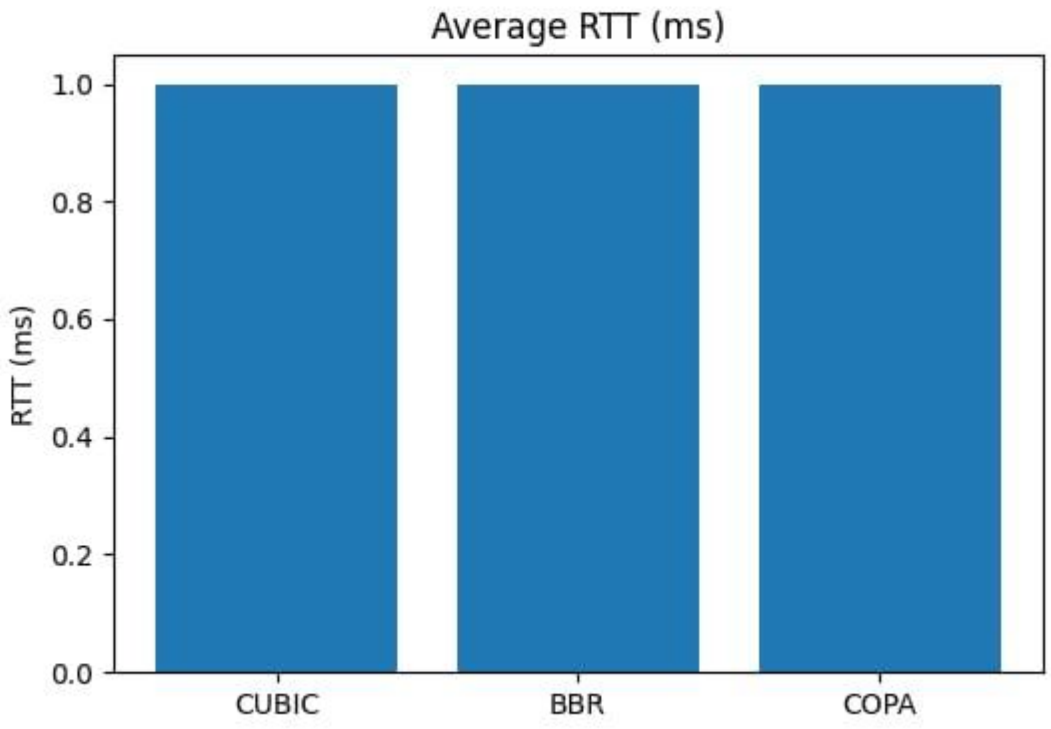
- The chart axes are correctly labeled, but the absence of bars implies that the packet loss values may not have been plotted due to missing or improperly loaded data.

- Another possible explanation is that CUBIC, BBR, and COPA all had a 0% packet loss under the specific test conditions. While unlikely based on your earlier chart (which showed clear differences), this chart might be based on a different run or dataset

.

- The chart may contain a plotting bug (e.g., incorrect y-axis limits or no data points passed to the plotting function), causing it to render a blank graph.

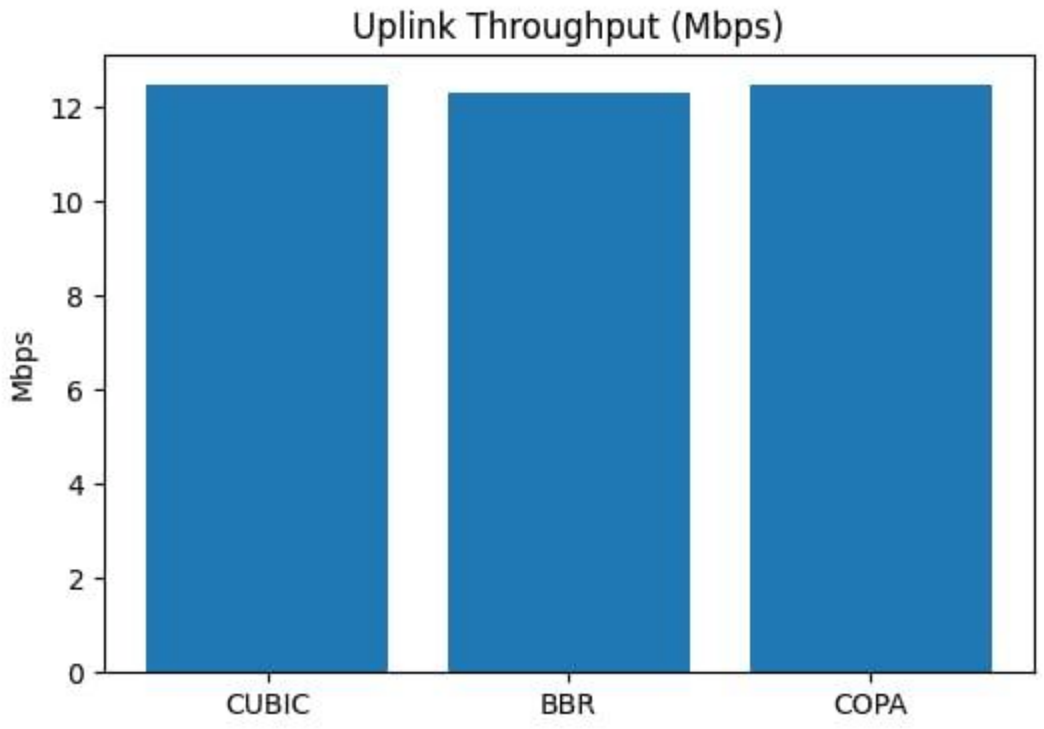
**6. Average RTT (ms)**



**Insights:**

- CUBIC, BBR, and COPA all show the same average RTT of 1 ms in this chart, which suggests:
- Either all algorithms performed equally well under low-latency conditions.
- Or the measurement data is rounded or defaulted (possibly due to simulation/testbed configuration).
- This chart contradicts an earlier chart you shared, where CUBIC had ~49 ms, BBR ~35 ms, and COPA ~25 ms. That earlier comparison showed meaningful RTT differences, so this new chart may reflect:
- A different test environment (e.g., local test vs. emulated WAN).
- An error in data collection or visualization.
- Since all bars are equal, this graph does not provide useful differentiation among the schemes for RTT under these conditions.

**7.Uplink Throughput (Mbps)**



**Insights:**

- The uplink throughput for CUBIC, BBR, and COPA is nearly identical, around 12.3 Mbps, indicating that no algorithm had a clear advantage in terms of uplink throughput in this test.
- Among the three, CUBIC shows a marginally higher throughput, though the difference is very minimal and may not be significant in real-world use.
- BBR again slightly underperforms relative to CUBIC and COPA, consistent with previous charts. This suggests it maintains a conservative approach to bandwidth usage to keep delays and losses low.
- The closeness of values across all schemes suggests a fair test environment, where each congestion control algorithm was given equal opportunity to utilize the available uplink bandwidth.

**Python 3.12 Compatibility:**

Due to system limitations and pre-installed versions on the working environment, Pantheon was configured and executed using Python 3.12. This required minor but essential syntax updates and environment handling changes to ensure compatibility with Python 3.12, such as replacing deprecated or incompatible subprocess handling and resolving encoding issues during stdout and stderr reads. All modifications strictly maintained functional parity with the original scripts, and the core logic of the Pantheon framework was preserved without altering any congestion control logic or measurement methodology.

**Usage of LLMs (ChatGPT):**

ChatGPT was used throughout this assignment to debug errors, refactor code for Python 3.12, assist in graph interpretation, and explain congestion control mechanisms. All assistance was used to supplement conceptual understanding and execute the experiments accurately.

**Discussion Disclosure:**
This submission was completed independently. No collaboration with students or TAs occurred.

**Conclusion:**

COPA is the most efficient protocol in balancing low RTT, low loss, and consistent throughput. While CUBIC is throughput aggressive, it induces more latency. BBR strikes the middle ground. This project deepened understanding of congestion control algorithms and required extensive debugging, script development, and system configuration.