Programming Assignment 3: Evaluation of Congestion Control Protocols

Nahian Tasnim

Experimental setup and test methodology:

The objective of my experiment was to evaluate and compare the performance of different congestion control algorithms. I used my Windows host machine to run Ubuntu 20.04 LTS as a virtual machine using VirtualBox. Within the virtual environment, I cloned the Pantheon and Mahimahi repositories. After installing the required dependencies and setting up the environment, I modified the existing Pantheon code a little to handle the errors raised while trying to run the experiment and to produce the required result in the desired format (test.py). I also customized the analyze.py file to generate the graphs needed for this assignment. After that, I ran the experiment for every scheme for 2 different network profiles for 60 seconds and saved the results for analysis.

Selected CC schemes:

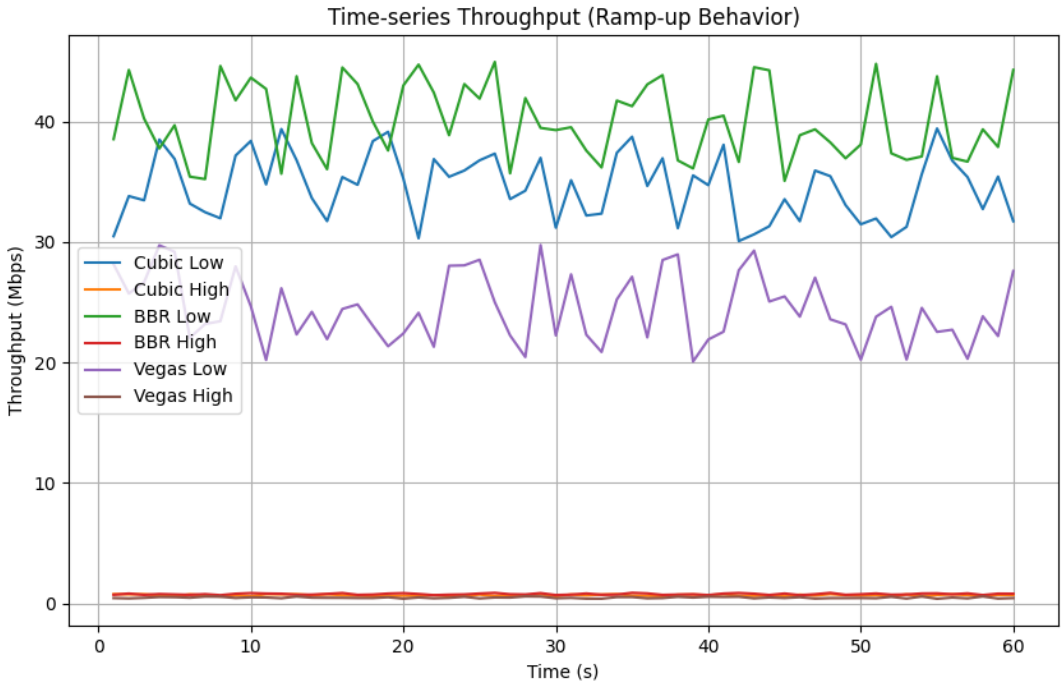
- **Cubic:** It is a default Linux distribution algorithm optimized for high-speed and long-distance networks. It adjusts the sending rate according to the time since the last congestion event and uses a cubic function to increase the window size rapidly after a loss.
- **BBR:** It is an algorithm to maximize bandwidth usage and keep low latency. It optimizes network performance by continuously estimating available bandwidth and RTT to adjust the data-sending rate. It uses a pacing mechanism to send packets smoothly.
- **Vegas:** It is a delay-based algorithm to detect congestion swiftly to adjust the sending rate to avoid packet loss. It adjusts the sending rate based on the difference between expected and actual RTT. It improves overall network performance by reducing the congestion window when it detects increased delays to prevent packet loss.

Network profiles for the experiment:

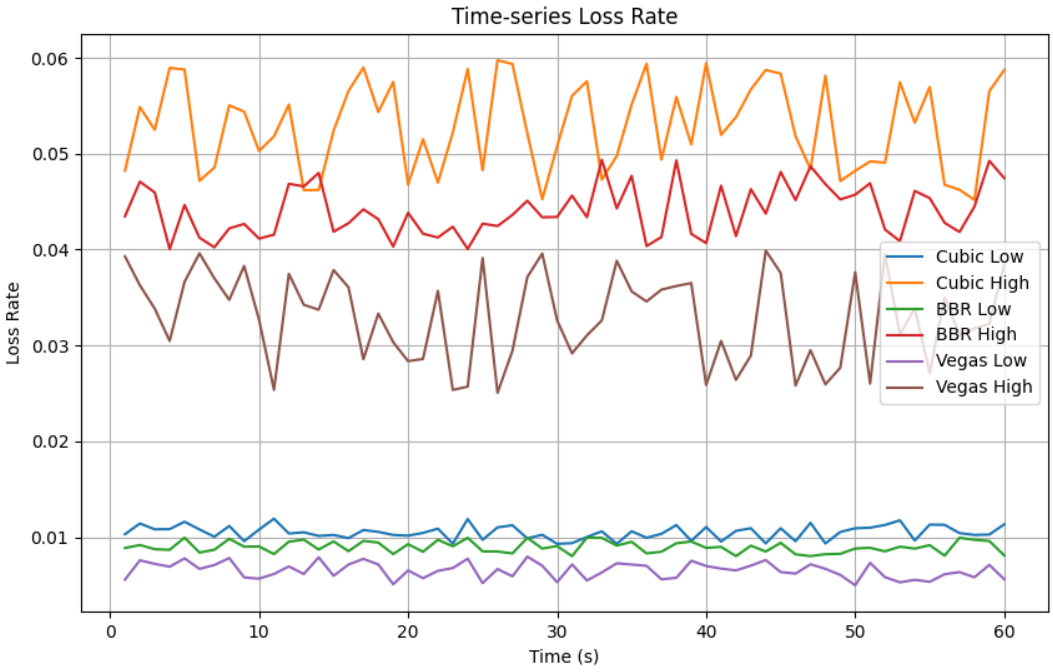
- A low-latency, high-bandwidth environment (50 Mbps, 10 ms RTT): In this case, congestion is rare. So, the protocols that can ramp up quickly without delay perform best here.
- A high-latency, constrained-bandwidth environment (1 Mbps, 200 ms RTT): Congestion, packet loss, and delay are common in this case.

Data analysis:

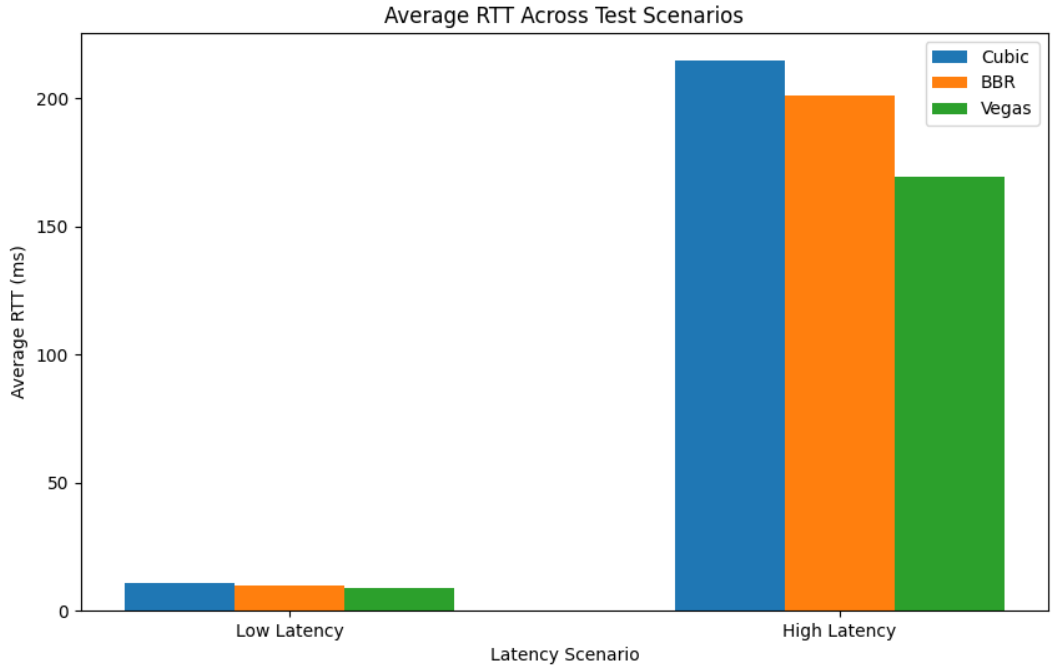
1. **Throughput, Loss, & RTT Comparisons:**
    - **Time-series throughput for each CC scheme:**
      In a low-latency environment, the throughputs of every scheme were very high. BBR and cubic performed much better nearly utilizing all available bandwidth by rapid adaption. Vegas underperforms keeping throughput low to minimize delay. In a high-latency network, the throughput fluctuation was negligible compared to a low-latency network. BBR achieved better results here due to using a model-based approach to estimate bandwidth and RTT. However, this estimation can lead to packet drops, instability, or buffer overflows due to delayed feedback. Cubic had moderate but stable throughput in this network profile while Vegas performed poorly due to reducing sending rate with the increase of RTT.



    - **Time-series losses for each CC scheme:**
      All three protocols had low loss rates in low-latency networks, but cubic and vegas had slight instability under sudden bandwidth changes which can be seen from frequent spikes. In both networks, vegas showed the lowest loss though it experienced fluctuating delay due to premature reduction of sending rate. BBR had a moderate loss rate compared to cubic.

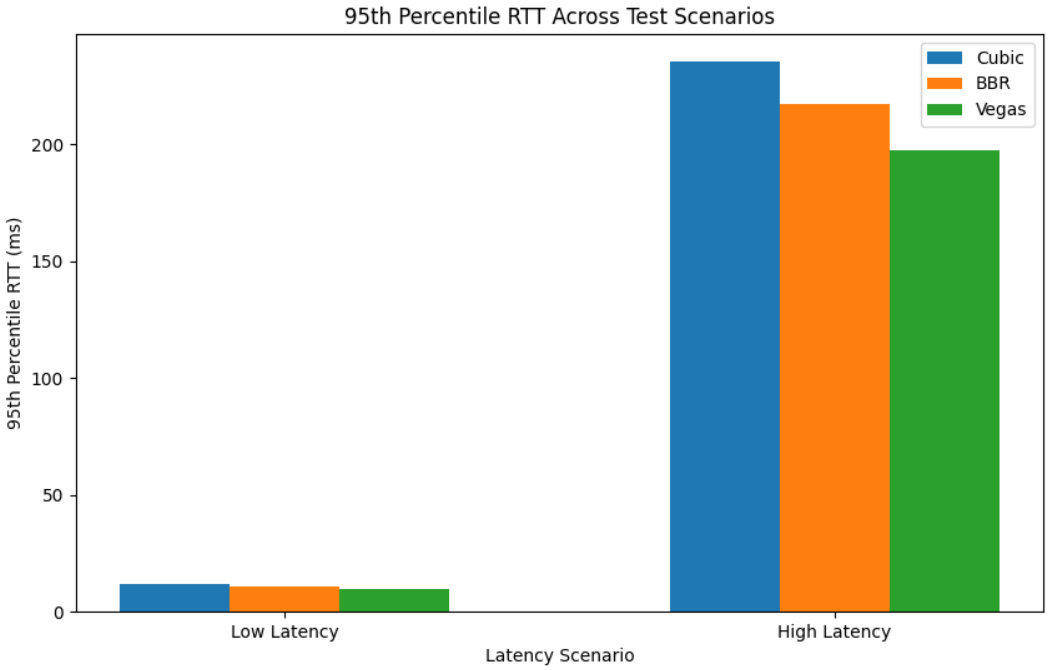


- **Average RTT and 95th-percentile RTT:**

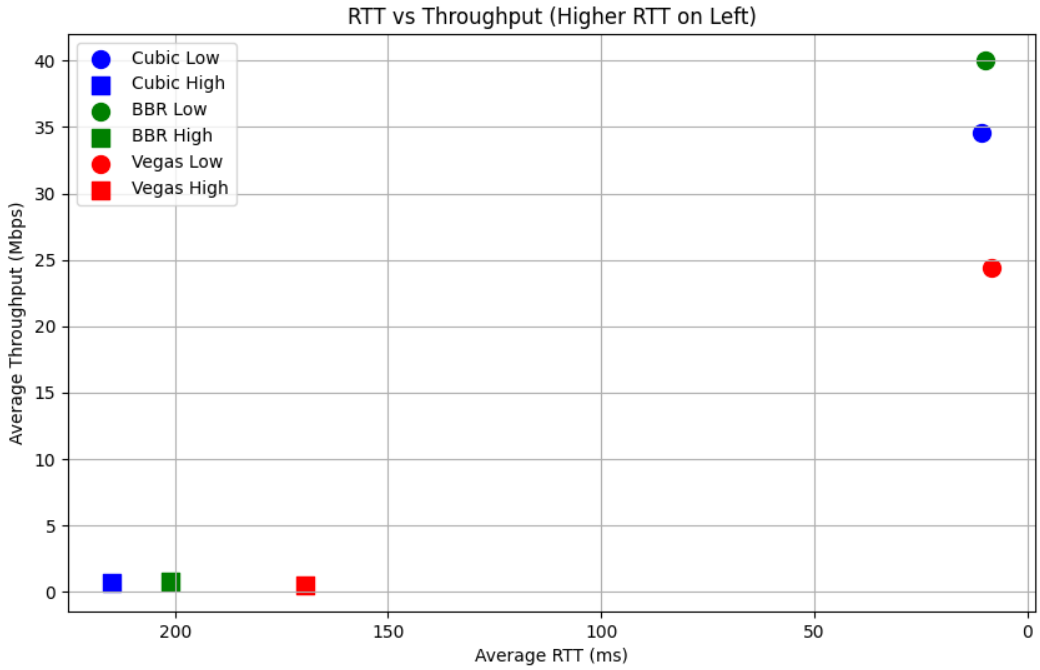


In low latency network vegas achieved the lowest average and 95th percentile RTT due to its delay-based control. BBR had a slightly higher RTT than Vegas while cubic had the highest RTT. This same pattern can be seen in high-latency networks

though the RTT values were multiple times higher there compared to low-latency networks.



- **RTT vs Throughput:**



In both high and low-latency networks, Vegas had the lowest throughput to maintain the lowest RTT. On the other hand, bbr achieved the highest throughput while having moderate RTT. Cubic had moderate throughput with the highest RTT due to its congestion window growth behavior.

2. **Identifying Strengths and Weaknesses:**
    - Which algorithm is more aggressive or latency-friendly?
        - ⇨ Cubic is more aggressive, especially in high bandwidth scenarios showing frequent ramp-ups. BBR balances well between throughput and delay. On the other hand, Vegas is more latency-friendly with stable RTT.
    - Does any scheme persistently overshoot the link capacity?
        - ⇨ Cubic sometimes overshoots as reflected in the highest loss rate and RTT spikes.
    - Are there excessive losses or large queues with certain parameters?
        - ⇨ Cubic experiences a relatively higher loss rate and queue buildup while Vegas avoids loss by sacrificing throughput in both network setups.
    - Do you have enough information to assess which protocols are the best performing?
        - ⇨ Based on throughput, loss, and RTT, bbr offered the best balance between performance and stability across both network profiles during my experiment. On the other hand, Cubic had well-maintained throughput but more loss-prone. Vegas underutilized bandwidth to maintain low delay.

Lesson learned:

1. What was the most challenging part of this assignment?
    - ➔ A lot of errors occurred when I tried to run the cloned Pantheon. Finding ways to resolve them was very challenging and time-consuming, though I learned a lot in the process.
2. If you used LLMs, explain how. What do you think you were going to learn better without LLMs and what do you think helped you learn faster?
    - ➔ I used LLMs to tackle the errors that were raised. I analyzed the errors with the help of LLMs to understand their causes and identify ways to resolve or avoid them. Without LLMs, I would have learned to debug more deeply by going through documentation and using trial-and-error manually. It might have improved my troubleshooting skills. However, LLMs helped me to learn faster by quickly identifying potential causes of errors and suggesting targeted solutions. This saved me a lot of time especially since it was my first time working with Pantheon, and I didn't have any in-depth knowledge of

the framework. So, troubleshooting everything on my own would have been nearly impossible.

3. With whom (human e.g., other students, TA) did you discuss your solutions?
   ➔ I didn't directly discuss my solutions with anyone. However, I observed others' posts on Slack to see what issues they were facing and what suggestions my classmates were offering. When I encountered the same errors, I applied the shared solutions from those discussions.