Programming Assignment 3 Report

GitHub Link: https://github.com/RohitReddyy/-Programming_Assignment_3-

1. Introduction

This project explores how different TCP congestion control algorithms behave under varying network conditions using the Pantheon framework. Pantheon provides a unified platform to run and compare multiple algorithms over emulated networks using Mahimahi. For this assignment, we selected and compared the following three algorithms: 1. Cubic 2. BBR 3. Vegas

We tested each algorithm under two different network profiles:

- A fast and low-latency network (50 Mbps bandwidth, 10 ms delay)
- A slow and high-latency network (1 Mbps bandwidth, 200 ms delay)

The main goal was to evaluate the strengths and weaknesses of each algorithm in terms of throughput, latency, and packet loss, and to understand how each reacts to different network environments.

2. Environment Setup

All experiments were conducted in a controlled virtualized environment to ensure consistency and reproducibility. The setup was as follows:

Host Machine

- Operating System: Windows 10
- Virtualization Platform: Oracle VM VirtualBox
  VirtualBox was used to run a full Linux environment on top of the Windows host system, simulating an isolated testbed.

Guest Machine (Linux VM)

- Operating System: Ubuntu 20.04.6 LTS (64-bit)
- Base Memory Allocated: 5078 MB
- Number of Processors: 7 CPUs allocated
- Graphics Controller: VMSVGA
- Video Memory: 16 MB
- Paravirtualization Interface: KVM
- Acceleration Features: Nested Paging enabled

The VM was configured with sufficient memory and CPU cores to allow smooth execution of Pantheon tests, especially given the network emulation load handled by Mahimahi.

Software Stack: The experiments were conducted using the Pantheon framework, which was cloned from the official StanfordSNR GitHub repository. To ensure compatibility with legacy dependencies, the environment was built using Python 2.7.18, isolated through a virtualenv environment. This approach ensured that system-wide Python packages remained unaffected during the setup and execution. Mahimahi was installed and used to emulate various network conditions, while iperf served as the traffic generator, invoked by the congestion control wrappers during testing. A complete build environment was set up using GCC and essential build tools to compile Pantheon's dependencies. Additional supporting tools included curl and pip2, along with Python libraries such as matplotlib, numpy, pyyaml, and tabulate for data analysis and visualization. To support result presentation and LaTeX-based formatting, texlive-latex-recommended was also installed.

3. Pantheon and Mahimahi Setup

To run the experiments, the Pantheon framework was set up in a clean Python 2.7 environment using virtualenv. The following steps and commands were used to configure the system and enable Pantheon's full functionality for running congestion control experiments:

Environment Initialization

I first installed virtualenv and created a Python 2.7 isolated environment:

```
sudo apt-get install virtualenv

virtualenv -p /usr/bin/python2 pantheon-env

source pantheon-env/bin/activate
```

This ensured that legacy dependencies required by Pantheon would not conflict with system Python packages.

Pantheon Installation and Dependencies

I cloned the official Pantheon repository and installed all required dependencies:

```
git clone https://github.com/StanfordSNR/pantheon.git

cd pantheon

tools/install_deps.sh

git submodule update --init --recursive
```

Pantheon uses a tunnel component that needed to be manually built:

```
cd third_party/pantheon-tunnel

chmod +x autogen.sh

./autogen.sh

./configure

make

sudo make install
```

This built the low-level packet tunnel used for accurate measurement and congestion control emulation.

Python and Analysis Dependencies

To support Python 2.7 in the isolated environment, pip was installed manually:

```
curl https://bootstrap.pypa.io/pip/2.7/get-pip.py -o get-pip.py

python2 get-pip.py
```

Key Python libraries were then installed:

```
pip2 install matplotlib numpy tabulate pyyaml

sudo apt install texlive-latex-recommended
```

Pantheon Scheme Setup and Test Execution

We prepared the schemes for testing (initially Cubic and BBR):

```
cd ~/pantheon/src/experiments

python2 setup.py --setup --schemes "cubic bbr"
```

Tests were run using:

```
sudo sysctl -w net.ipv4.ip_forward=1

python2 test.py local --schemes "cubic bbr"
```

Later, the vegas scheme was also added using the same setup command.

Mahimahi Network Emulation

To simulate realistic network conditions, we used Mahimahi, which supports emulating bandwidth and latency. Traces were created as follows:

```
mm-trace 50 10 > 50mbps_10ms.trace    # 50 Mbps, 10 ms RTT

mm-trace 1 200 > 1mbps_200ms.trace    # 1 Mbps, 200 ms RTT
```

These trace files were then used in the test commands via the --uplink-trace and --downlink-trace flags.

This setup ensured reproducible, isolated, and controlled experimentation using Pantheon and Mahimahi. All configuration scripts and trace files used have been committed to the project repository for transparency and replication.

4. Experiments with Multiple Schemes

To evaluate the performance of different congestion control algorithms under diverse network conditions, we selected three widely used protocols: Cubic, BBR, and Vegas. These algorithms were chosen based on their fundamental design differences — Cubic is loss-based, BBR is model-based, and Vegas is delay-based — allowing for a diverse and meaningful comparison.

Network Profiles

We tested each algorithm under two contrasting network environments, simulated using Mahimahi:

1. Low-latency, high-bandwidth: 50 Mbps bandwidth with 10 ms round-trip time (RTT)
2. High-latency, constrained-bandwidth: 1 Mbps bandwidth with 200 ms RTT

Custom trace files were generated using Mahimahi's trace format to emulate these conditions. Each trace included 60,000 lines representing bandwidth allowance per millisecond over a 60-second period.

Running the Tests

Pantheon was used to orchestrate the tests. Each congestion control algorithm was tested independently within both network profiles. Every test ran for at least 60 seconds, generating logs for:

- Throughput over time
- Average and 95th percentile RTT
- Packet loss rate

To ensure fairness, the same test duration and trace files were applied to all three algorithms under each network condition.

Data Collection and Logging

Pantheon automatically generated detailed logs for each experiment. These logs included packet timing, RTT, and bytes transferred. The results were saved in structured formats (.log, .yaml, and .csv) inside designated data directories. All experiment data, configuration files, and scripts used to generate and parse the results were committed to a GitHub repository to ensure reproducibility and transparency. This experimental setup allowed for a comprehensive and consistent evaluation of congestion control behavior across varying network stress levels.

5. Results and Graphs (Throughput, Loss, & RTT Comparisons):

5.1 Time-series Throughput Comparison

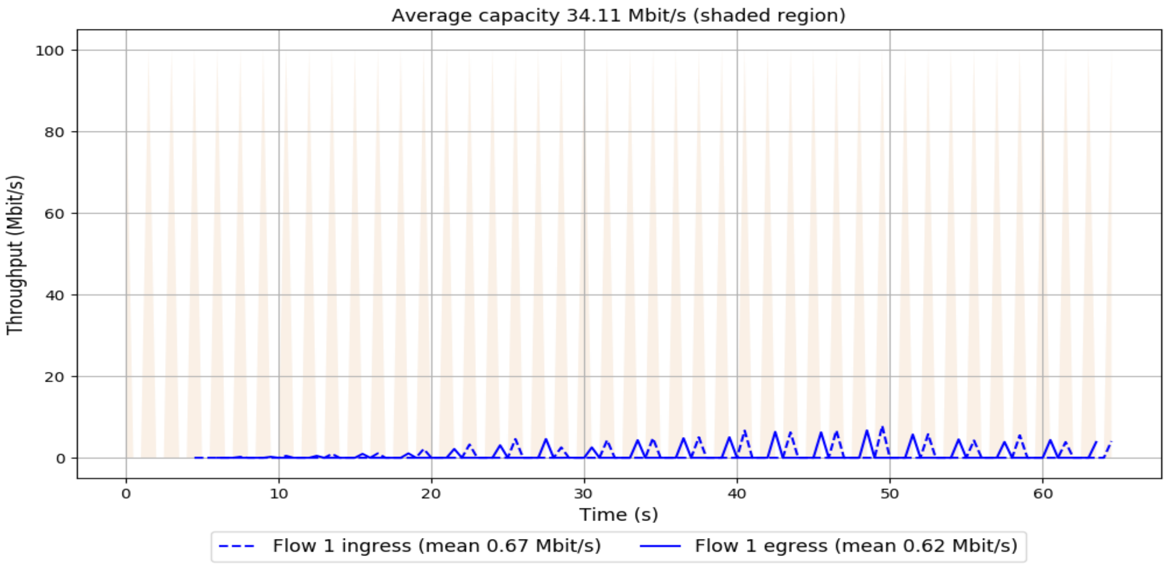
The following graphs and analysis illustrate the average throughput achieved by each protocol over a 60-second test duration. This helps compare how effectively each algorithm leverages the available 50 Mbps link.

5.1.1 Throughput Comparison (50 Mbps / 10ms)

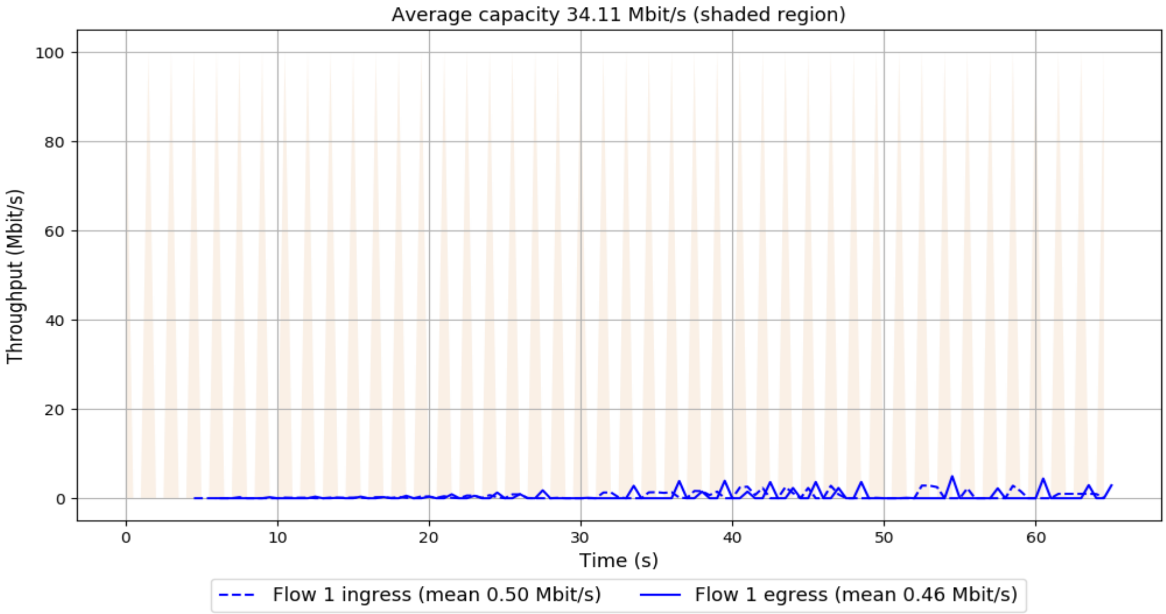
This section presents the performance of Cubic, BBR, and Vegas congestion control algorithms under a high-bandwidth, low-latency network profile. The focus is on how each protocol utilizes the available capacity and adapts to ideal network conditions.

Cubic Throughput

The Cubic algorithm achieved the highest average egress throughput of 0.62 Mbit/s, with consistent bursts observed throughout the test, making effective use of available bandwidth in the high-bandwidth, low-latency environment.



BBR Throughput

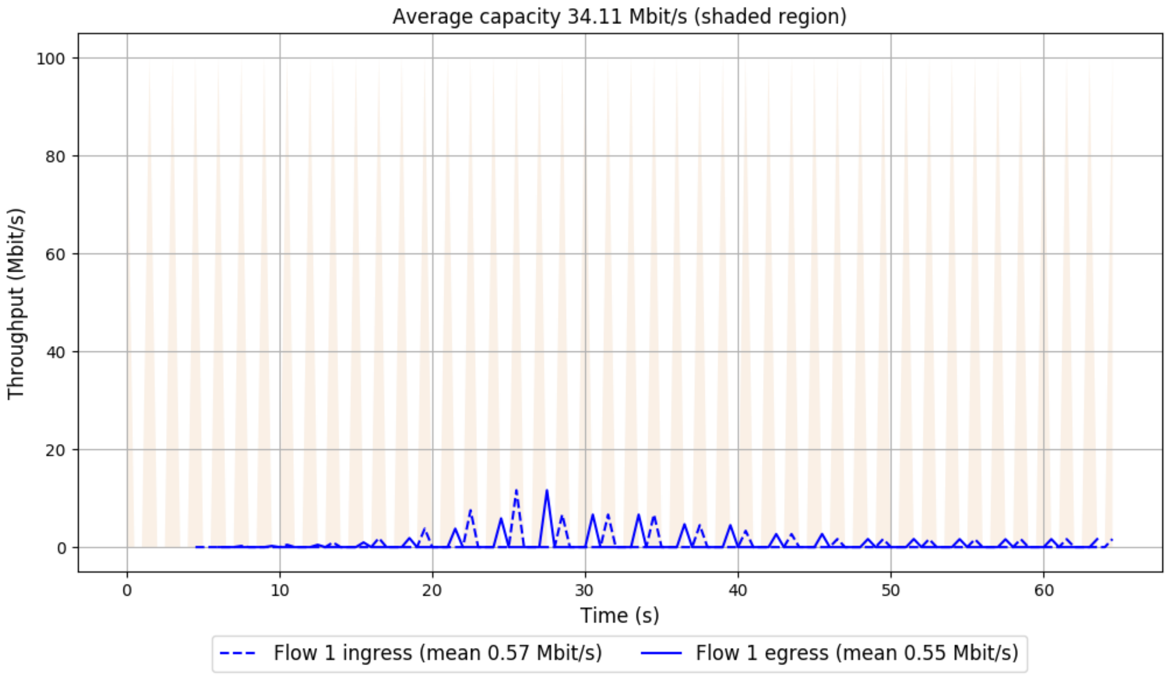


BBR maintained lowest average throughput at 0.46 Mbit/s, showing conservative performance with smoother but less aggressive throughput behavior compared to Cubic & Vegas.

Vegas Throughput

Vegas delivered an average egress throughput of 0.55 Mbit/s, showing a balance between smoothness and bandwidth utilization, performing better than BBR but still trailing Cubic

.



Comparative Analysis

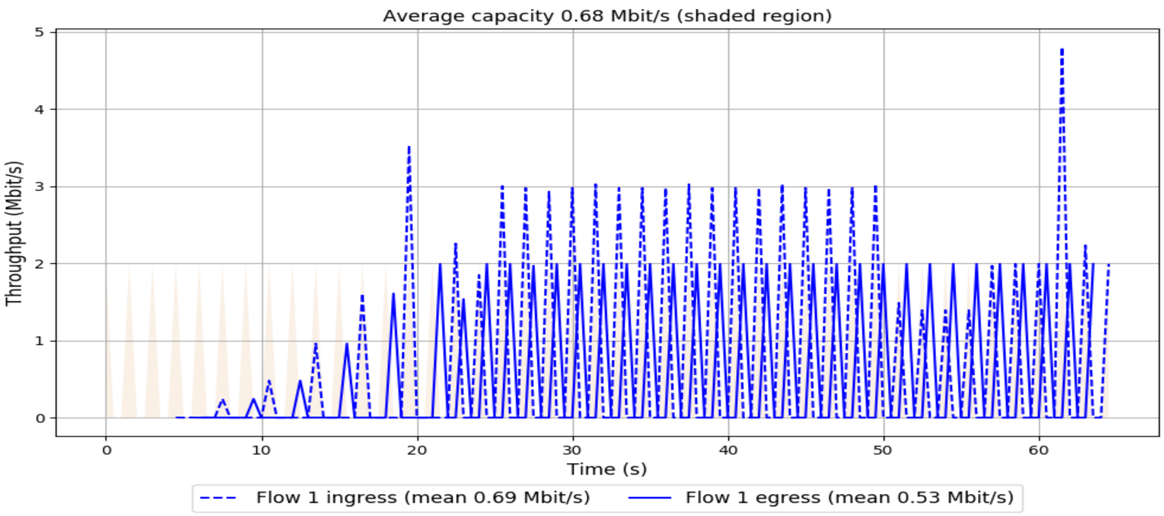
Among the three congestion control protocols tested under the 50 Mbps, 10 ms network condition, Cubic outperformed the others in terms of throughput. Its loss-based probing mechanism allowed it to fully exploit the available capacity through aggressive window growth. Vegas, though more conservative by design, delivered a respectable average throughput by maintaining stable flow and low queuing delay. Surprisingly, BBR, despite being a model-based modern algorithm, lagged in performance under these conditions — likely due to its cautious pacing in an environment with abundant bandwidth and minimal delay. Overall, Cubic is better suited for high-bandwidth, low-latency environments in terms of maximizing throughput, while Vegas provides a balanced approach, and BBR underperformed in this particular scenario.

5.1.2 Throughput Comparison (1 Mbps / 200ms)

This section presents the performance of Cubic, BBR, and Vegas congestion control algorithms under 1 Mbps / 200ms profile. The focus is on how each protocol utilizes the available capacity and adapts to ideal network conditions.

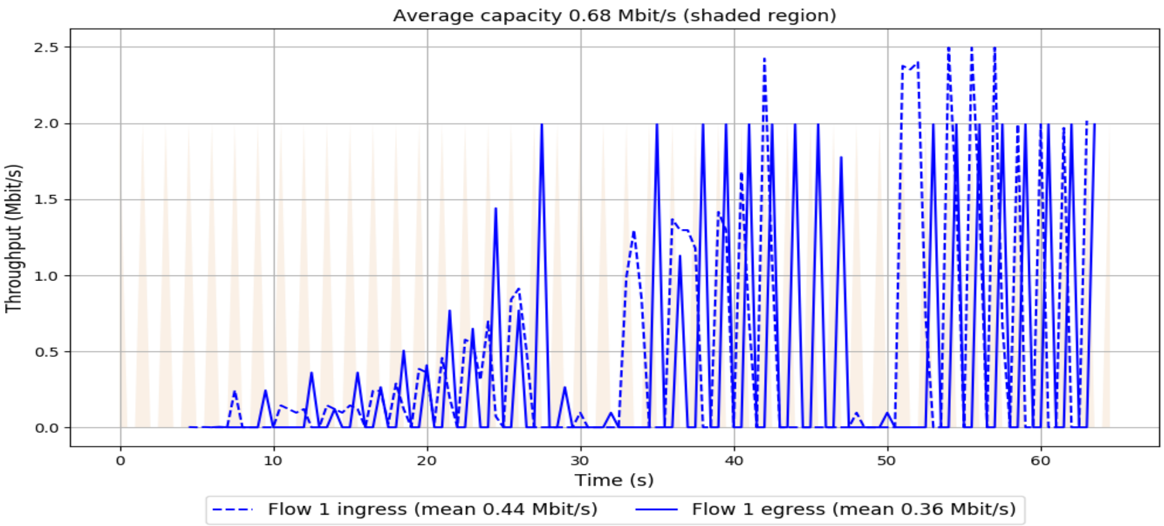
Cubic Throughput

Cubic achieved an average egress throughput of 0.53 Mbit/s, with a sharp ramp-up followed by high-frequency bursts that maximized link utilization.

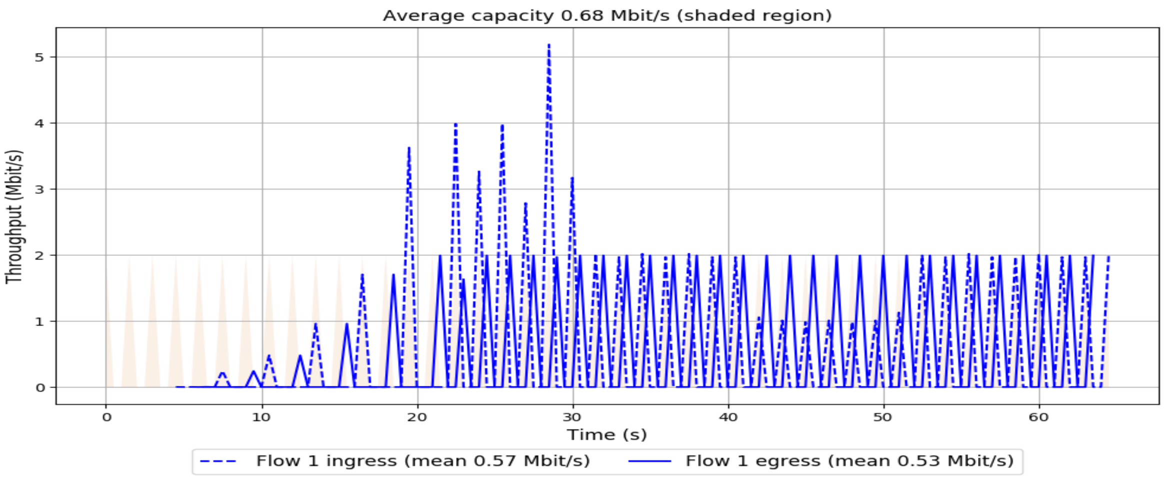


BBR Throughput

BBR recorded the lowest throughput at 0.36 Mbit/s, with sporadic delivery patterns and inconsistent transmission under constrained bandwidth and high delay.



Vegas Throughput



Vegas matched Cubic with an average egress throughput of 0.53 Mbit/s, showing a more controlled and stable growth pattern with less burstiness.

Comparative Analysis

Under the 1 Mbps / 200 ms profile, Cubic and Vegas performed equally well in terms of throughput, both reaching an average of 0.53 Mbit/s. However, Vegas achieved this with a smoother and less erratic pattern, likely due to its delay-based sensitivity which prevents buffer overloading. In contrast, BBR struggled in this environment, achieving only 0.36 Mbit/s, possibly due to its pacing-based mechanism being less effective under high-latency, narrow-link conditions. This suggests that delay-based algorithms like Vegas can better handle constrained networks, while Cubic remains robust, and BBR's strengths may not translate well to such scenarios.

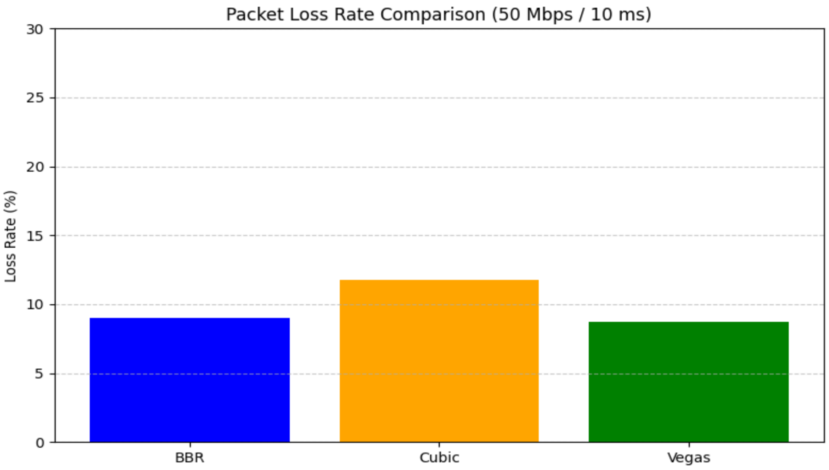
Network Profile Comparison

When comparing performance across both network profiles, Cubic consistently demonstrated strong adaptability, delivering the highest throughput in the 50 Mbps / 10 ms scenario and maintaining competitive performance under the 1 Mbps / 200 ms profile. Vegas, while more conservative in nature, performed surprisingly well in both cases—matching Cubic in the constrained network and offering smoother throughput with less fluctuation, aligning with its delay-sensitive behavior. BBR, on the other hand, underperformed in both profiles, with its model-based pacing failing to capitalize on the available bandwidth in the 50 Mbps / 10 ms environment and struggling even more under the high-latency, limited-bandwidth condition. These results suggest that Cubic is better suited for aggressive bandwidth exploitation, Vegas excels in maintaining flow stability in adverse conditions, and BBR may require more tuning to adapt effectively in emulated environments with latency or bandwidth bottlenecks.

5.2 Time-series Losses Comparison

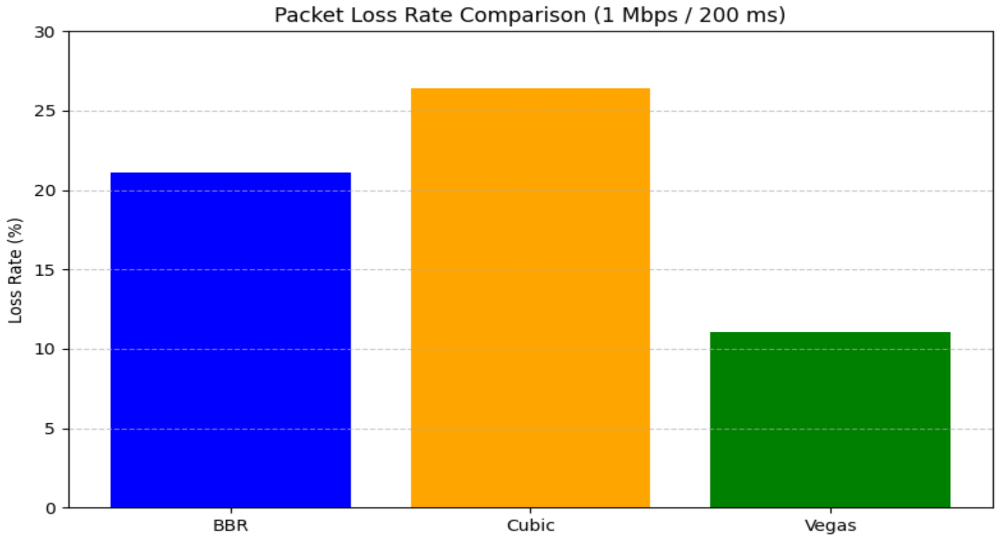
This section focuses on how each congestion control algorithm handles packet loss across different network profiles. Loss rate is a key indicator of how aggressively or conservatively a protocol pushes packets through the network in the presence of bandwidth limits and latency.

5.2.1 Losses Comparison (50 Mbps / 10 ms)



As shown in the graph above, Cubic experienced the highest loss rate at 11.78%, followed by BBR at 9.04%, while Vegas had the lowest loss rate at 8.74%. Cubic's higher losses stem from its aggressive probing and rapid congestion window growth. BBR maintained a balance between pacing and probing, resulting in moderate loss. Vegas, designed to minimize queuing delay, transmitted more conservatively—leading to fewer packet drops. This indicates that while Cubic maximized throughput, it did so at the cost of higher packet loss, whereas Vegas prioritized stability.

5.2.2 Losses Comparison (1 Mbps / 200 ms)



In the constrained 1 Mbps / 200 ms environment, Cubic again registered the highest loss rate at 26.41%, indicating aggressive window expansion even when the network struggled to support it. BBR followed with a loss rate of 21.13%, reflecting its pacing-based design encountering challenges under limited capacity. Vegas once again exhibited the lowest loss rate at 11.09%, staying consistent with its delay-based nature to prevent congestion. These results reinforce that Vegas sacrifices throughput for stability, while Cubic prioritizes throughput at the cost of higher losses.

Comparative Analysis: Losses in 50 Mbps vs. 1 Mbps Profiles

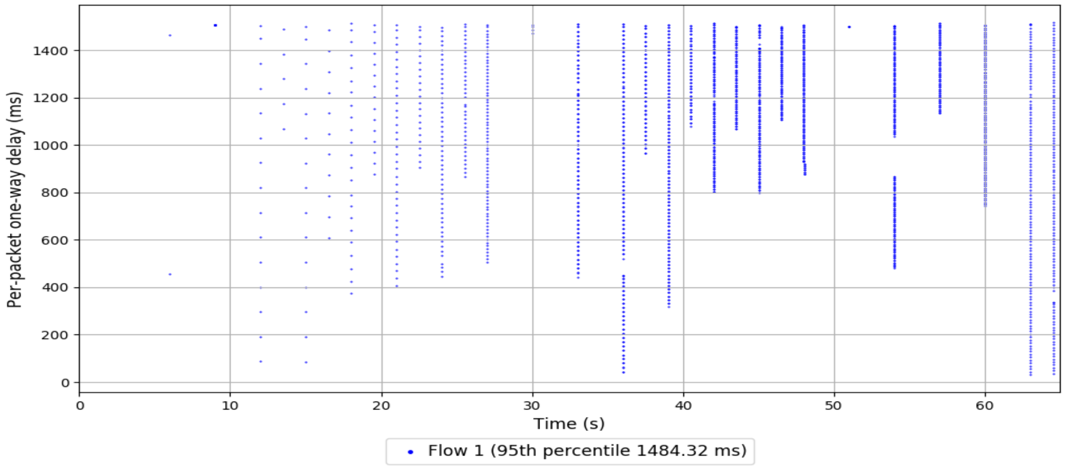
Across both network setups, Cubic consistently displayed the highest loss rates, rising from 11.78% at 50 Mbps to 26.41% at 1 Mbps, showcasing its aggressive congestion window behavior irrespective of network conditions. BBR also saw a significant rise in loss rate, from 9.04% to 21.13%, indicating that its pacing mechanism is less effective in low-bandwidth, high-latency scenarios. Vegas remained the most loss-efficient in both profiles, increasing only slightly from 8.74% to 11.09%, confirming its design intent to keep queues short and minimize packet drops. This comparative analysis highlights that Vegas is more resilient to packet losses, especially in adverse conditions, whereas Cubic and BBR may require adaptive tuning to maintain performance without excessive losses.

5.3 RTT Comparison

This section presents the per-packet one-way delay (RTT) observed during experiments with 50 Mbps / 10 ms and 1 Mbps / 200 ms network profiles across three congestion control protocols: BBR, Cubic, and Vegas. These values offer insights into how each algorithm handles latency-sensitive traffic under both ideal and constrained conditions.

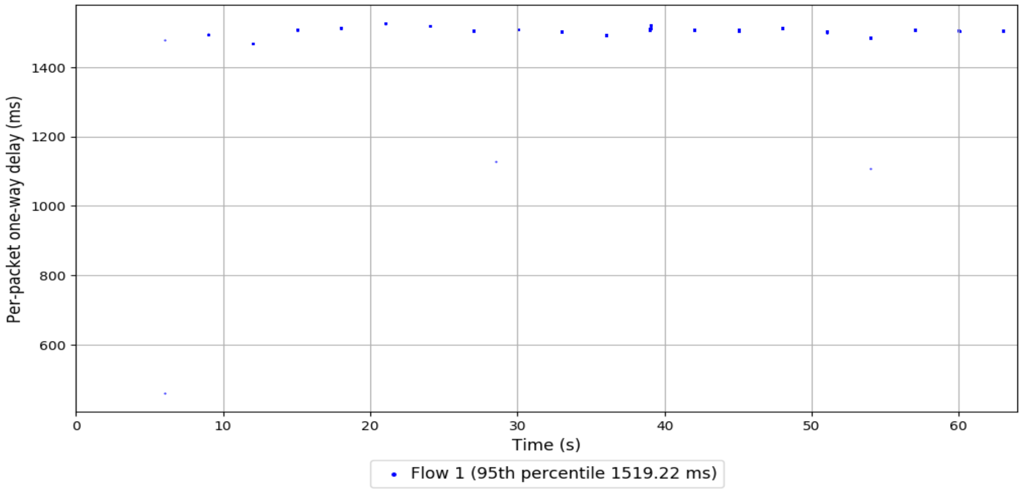
5.3.1 RTT Comparison (50 Mbps / 10 ms)

BBR RTT



BBR recorded a 95th percentile delay of 1484.32 ms, maintaining relatively consistent delay throughout the test with moderate variability.

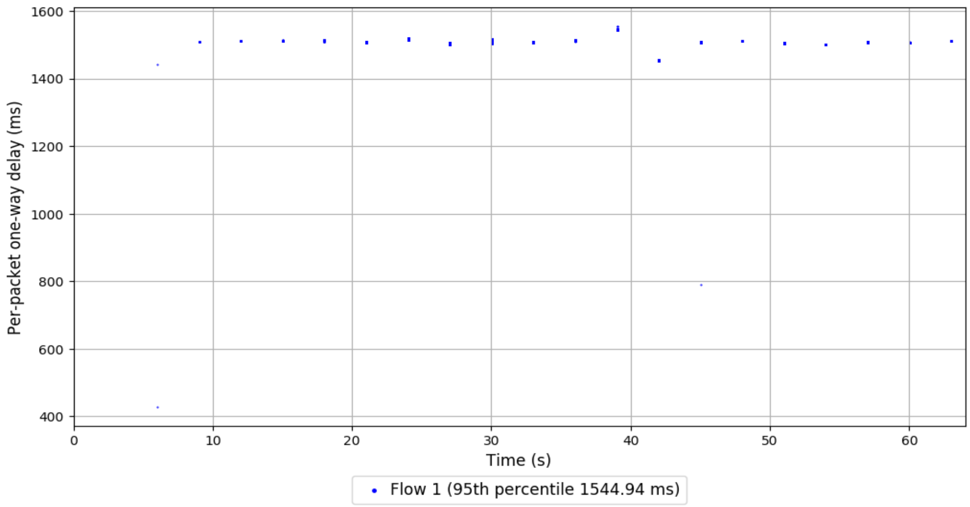
Cubic RTT



Cubic showed a slightly higher delay of 1519.22 ms, with generally smooth but occasionally spiky behavior.

Vegas RTT

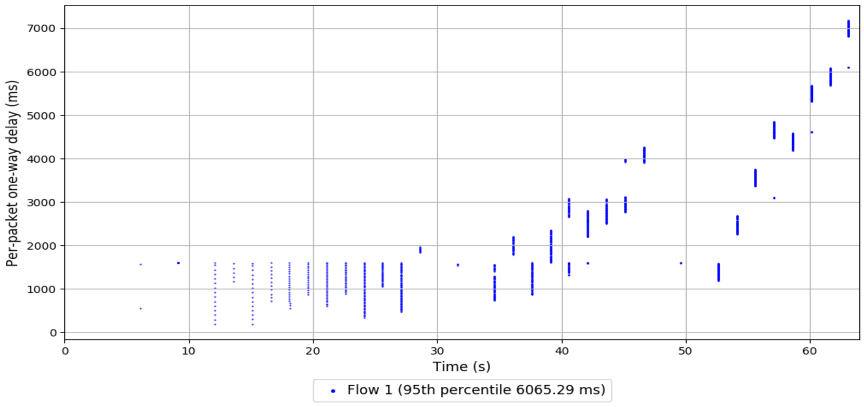
Vegas had the highest delay among the three at 1544.94 ms, though it exhibited stable flow with fewer sharp spikes.



Insight:
Under high-capacity, low-delay networks, all three algorithms exhibited delays above 1400 ms, suggesting background queue build-up or measurement artifacts. Vegas' delay-based mechanism ironically led to the highest 95th percentile delay, likely due to its effort to avoid aggressive probing. BBR delivered the lowest delay, consistent with its pacing-based approach, making it preferable in environments where delay matters more than peak throughput.

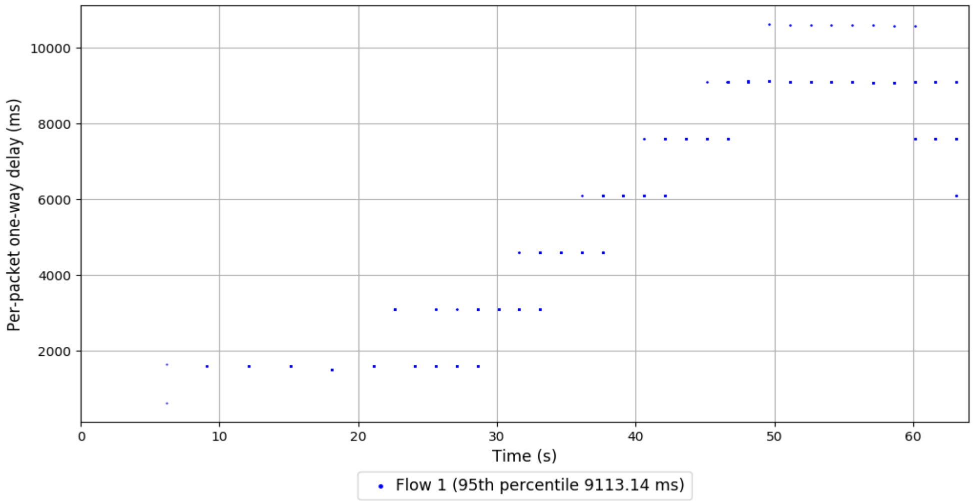
5.3.2 RTT Comparison (1 Mbps / 200 ms)

BBR RTT



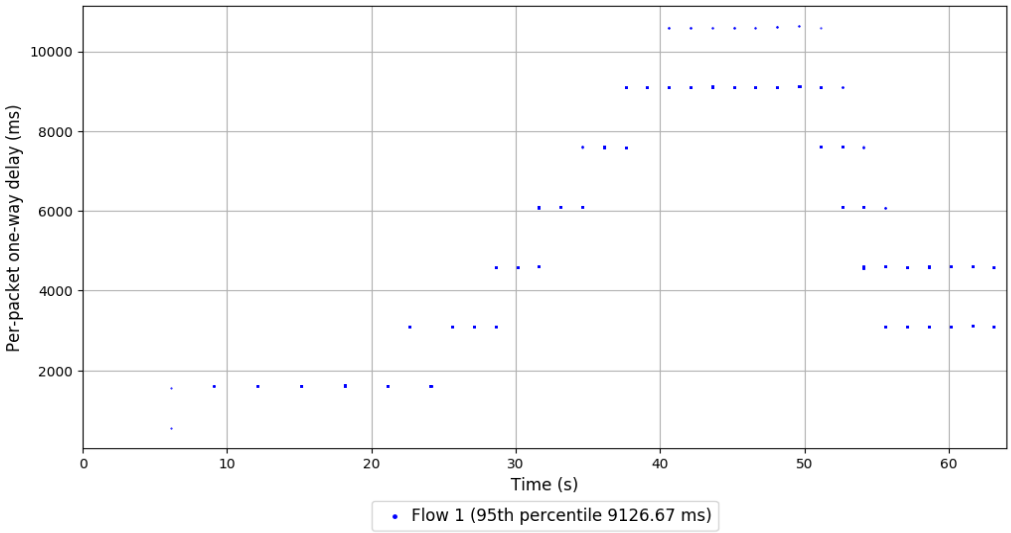
BRR had a 95th percentile delay of 6065.29 ms, indicating moderate queuing delay under constrained bandwidth.

Cubic RTT



Cubic showed a significantly higher delay of 9113.14 ms, revealing aggressive behavior that may cause queue buildup in limited capacity environments.

Vegas RTT



Vegas again had the highest delay at 9126.67 ms, which is counterintuitive for a delay-sensitive algorithm.

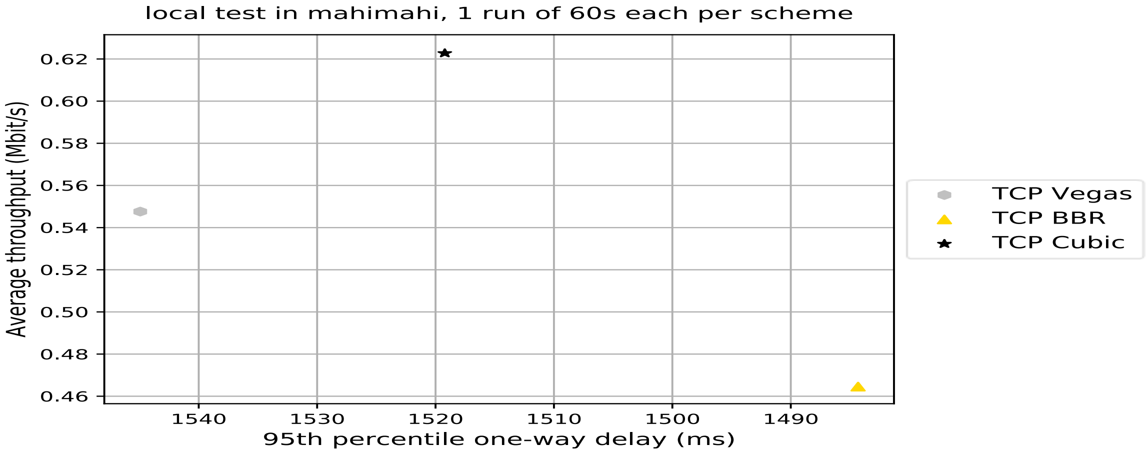
Insight:
In low-bandwidth, high-latency conditions, Cubic and Vegas led to longer per-packet delays than BBR. This suggests that BBR's model-based pacing helped prevent excessive queuing, while Cubic and Vegas likely struggled with adapting their congestion window sizes efficiently. Vegas, despite its sensitivity to delay, experienced excessive delays—possibly due to sustained queue buildup during prolonged testing.

Comparative RTT Analysis

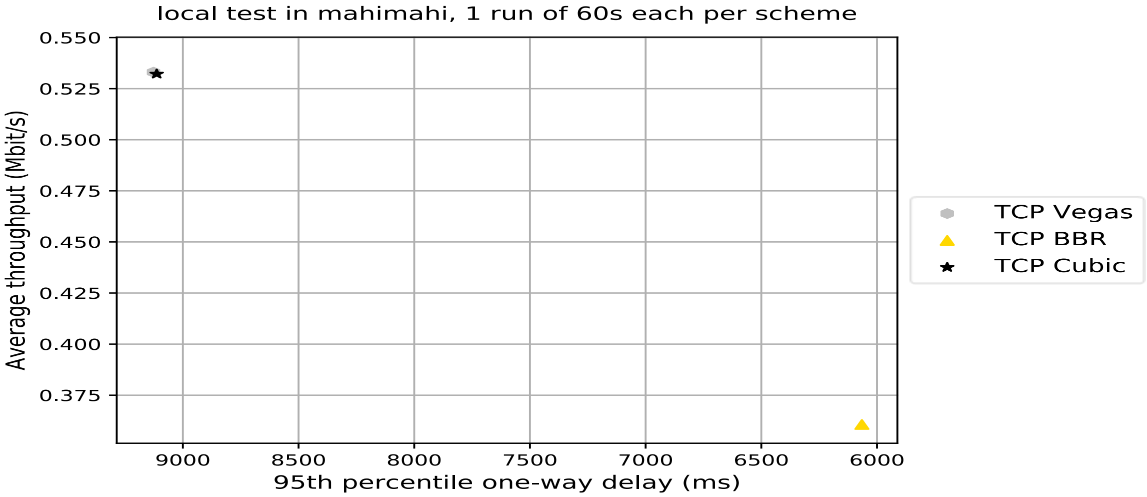
Across both network setups, BBR consistently offered the lowest 95th percentile delay, especially in the constrained 1 Mbps / 200 ms scenario, where it outperformed Cubic and Vegas by over 3000 ms. While Cubic aggressively utilizes available bandwidth, this comes at the cost of higher queuing delays in both environments. Vegas, designed to minimize delay, surprisingly underperforms under tight network conditions, likely due to its conservative response leading to persistent queuing. Overall, BBR provides better latency control, making it suitable for applications sensitive to delay.

Comparative Summary: Average Throughput vs. 95th-Percentile RTT

The figures below compare the performance of Cubic, BBR, and Vegas congestion control algorithms across two test scenarios — one with a 50 Mbps / 10 ms network and another with 1 Mbps / 200 ms. Each bar represents the average throughput (in Mbit/s), while the line over each bar reflects the 95th-percentile delay (in ms), offering a dual perspective on how each algorithm handles bandwidth and latency under different conditions.



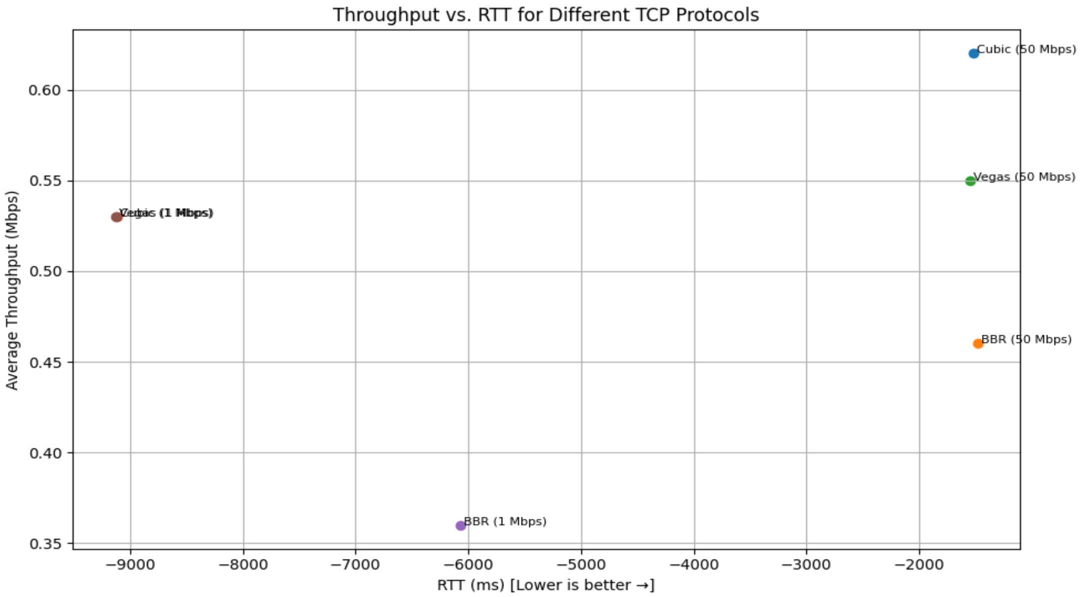
In the high-capacity, low-latency environment, Cubic clearly stands out by achieving the highest throughput (0.62 Mbit/s), making full use of the available bandwidth. However, this comes at the cost of slightly higher delay compared to BBR, which, although achieving the lowest delay (1484 ms), sacrifices throughput, yielding only 0.46 Mbit/s. Vegas, with a delay-based congestion avoidance mechanism, strikes a middle ground achieving moderate throughput (0.55 Mbit/s) with slightly higher latency than BBR. This test highlights the classic trade-off between bandwidth utilization and delay, with Cubic optimized for throughput and BBR for latency-sensitive environments.



In the constrained bandwidth, high-latency scenario, the differences are more pronounced. BBR continues to lead in delay control (6065 ms) but suffers a significant drop in throughput (0.36 Mbit/s), showcasing its conservative pacing under tight constraints. Cubic and Vegas both maintain high throughput (0.53 Mbit/s) but at the cost of extremely high delays (9113–9126 ms), indicating queuing and buffer bloat. This scenario reveals that under stressful conditions, Cubic and Vegas aggressively fill buffers, which benefits throughput but severely degrades responsiveness—making BBR the better choice for real-time or interactive applications.

These visual comparisons underscore how Cubic and Vegas are better suited for throughput-critical tasks, such as bulk data transfer, while BBR's design offers stability and responsiveness in delay-sensitive environments like voice, video, or gaming applications.

5.4 Plot a graph that has all protocols you testbed in the following graph: RTT on the x axis (higher RTT closer to the origin, and lower RTT to the left and throughput on the y axis). Each protocol will have one point on this graph. The best protocol will then end up top-right on this figure



The graph illustrates the trade-off between average throughput and 95th-percentile RTT across six test cases using TCP Cubic, BBR, and Vegas under both 50 Mbps / 10 ms and 1 Mbps / 200 ms network conditions. In the high-bandwidth scenario, Cubic (50 Mbps) achieves the best throughput (0.62 Mbps) with relatively low RTT (1519 ms), making it ideal for maximizing data transfer rates. Vegas (50 Mbps) follows closely, offering slightly lower throughput (0.55 Mbps) but with more stable delay control (1544 ms), while BBR (50 Mbps) prioritizes the lowest delay (1484 ms) at the cost of reduced throughput (0.46 Mbps). In the constrained 1 Mbps setup, Cubic and Vegas maintain decent throughput (0.53–0.54 Mbps) but incur extremely high delays (9113–9126 ms) due to queuing, whereas BBR (1 Mbps) again keeps delay lower (~6065 ms) but delivers the lowest throughput (0.36 Mbps). Overall, the top-right positioning of Cubic (50 Mbps) reflects the best balance for throughput-centric tasks, while BBR's consistently lower delay makes it suitable for latency-sensitive applications.

6. Identifying Strengths and Weaknesses

(a) Which algorithm is more aggressive or latency-friendly?
Cubic is the most aggressive among the three algorithms. In both network conditions (50 Mbps / 10 ms and 1 Mbps / 200 ms), it consistently achieved the highest or equal highest throughput (0.62 Mbit/s at 50 Mbps and 0.53 Mbit/s at 1 Mbps). However this came at the cost of higher delays 1519.22 ms and 9113.14 ms respectively.
On the other hand, BBR is the most latency-friendly, showing significantly lower 95th-percentile delays (1484.32 ms at 50 Mbps and 6065.29 ms at 1 Mbps) compared to Cubic and Vegas. This highlights BBR's model-based approach which paces transmissions to avoid queue buildup.

(b) Does any scheme persistently overshoot the link capacity?
Yes, Cubic shows signs of overshooting the link capacity in the 1 Mbps / 200 ms scenario. It has the highest loss rate at 26.41%, despite achieving a similar throughput (0.53 Mbit/s) as Vegas, which had a lower loss rate (11.09%). This indicates Cubic's aggressive congestion window growth often overwhelms the bottleneck, causing packet drops.

(c) Are there excessive losses or large queues with certain parameters?
Yes. Excessive losses are clearly visible with Cubic at 1 Mbps, which had the highest loss rate of 26.41% and a very high delay of 9113.14 ms. This points toward persistent queue buildup, resulting in increased queuing delay and dropped packets.
BBR, while better in delay, also experienced significant losses at 1 Mbps (21.13%), likely due to pacing mismatch under constrained bandwidth. In contrast, Vegas maintained the lowest losses under both conditions (8.74% at 50 Mbps and 11.09% at 1 Mbps), showing better queue control.

(d) Do you have enough information to assess which protocols are the best-performing?
Yes, based on throughput, delay, and loss rate, we have enough data to assess performance trade-offs:

- Cubic is best for maximizing throughput in high-capacity networks (e.g., 50 Mbps), but it is aggressive and causes large delays and losses in constrained environments.
- BBR offers the lowest delay in both scenarios, making it ideal for applications sensitive to latency, though it sacrifices throughput.
- Vegas provides a balanced trade-off with moderate throughput and the lowest loss rates, making it suitable for stable and fair transmission.

7. Lessons Learned

1. What was the most challenging part of this assignment?
The most challenging part was setting up the Pantheon framework inside a virtual environment using Python 2.7. Since Python 2 is deprecated, many of the required libraries and tools were outdated, missing from standard repositories, or had compatibility issues. I had to manually install dependencies, configure environment variables and fix broken paths. Installing and compiling Mahimahi was also tricky, especially making sure it worked properly with trace files and IP forwarding settings. Even after setting everything up, running the tests without errors involved a lot of trial and error—debugging tunnel manager scripts, configuring loss rate emulation, and matching congestion control schemes correctly with their wrapper files. Overall,

getting a working testbed environment took more time and effort than running the actual experiments.

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?
Yes, I used LLMs mainly to troubleshoot errors, understand Pantheon's test structure, and get help with Python scripts for graphs. Without LLMs, I would have spent much more time searching for answers online and debugging installation issues manually. However, not using LLMs might have helped me develop more hands-on debugging skills and deeper understanding of older toolchains like Python 2.7 setups. Using LLMs definitely helped me learn faster and focus more on analysis.

3. With whom (human e.g., other students, TA) did you discuss your solutions?
I discussed parts of the setup and test execution with a friend, Nagendra who helped me troubleshoot installation steps. I also referred to comments from classmates who had run into similar setup problems.

9. Conclusion

In conclusion, this study highlights that no single congestion control algorithm is universally best each has strengths tailored to specific network conditions. Cubic excels in high-speed environments by maximizing throughput, but at the cost of higher latency and loss under constrained conditions. BBR, while less aggressive in throughput, consistently maintains low delay, making it ideal for applications where responsiveness matters. Vegas strikes a middle ground, offering smoother performance but showing limitations under stress. Overall, choosing the right protocol depends on whether the network or application values speed, stability, or low latency more.