Pantheon PA3: Congestion Control Evaluation Report

Name: Harshil Sharma
Course: CSCI 5300 – Computer Networks
Platform: Ubuntu 20.04 (VirtualBox)
GitHub Repository: https://github.com/xxender13/csci-network-pa3-pantheon.git
Submission Date: April 2025

1. Methodology

Algorithms Selected

- **Part A**: bbr, cubic
- **Part B & C**: vegas, cubic, vivace

Network Profiles

- **Low Latency Profile**: 50 Mbps uplink/downlink, 5 ms delay
- **High Latency Profile**: 1 Mbps uplink/downlink, 100 ms delay

Test Procedure

- Used pantheon/test.py for controlled test execution (60 seconds per scheme)
- Used mm-delay and trace files to simulate realistic links
- Custom scripts were created for RTT and throughput parsing (awk + gnuplot)
- Analysis done on both per-packet logs and summary metrics

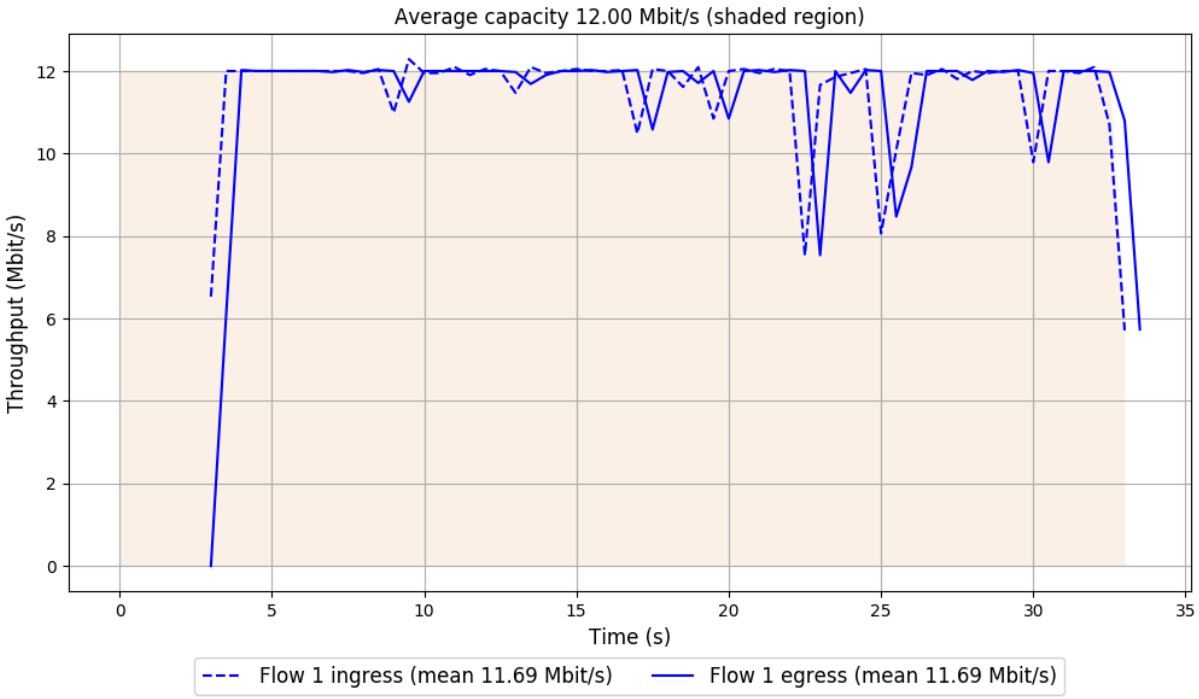
Results Directory:

- Part A: pantheon/src/experiments/data
- Part B & C: pantheon/result

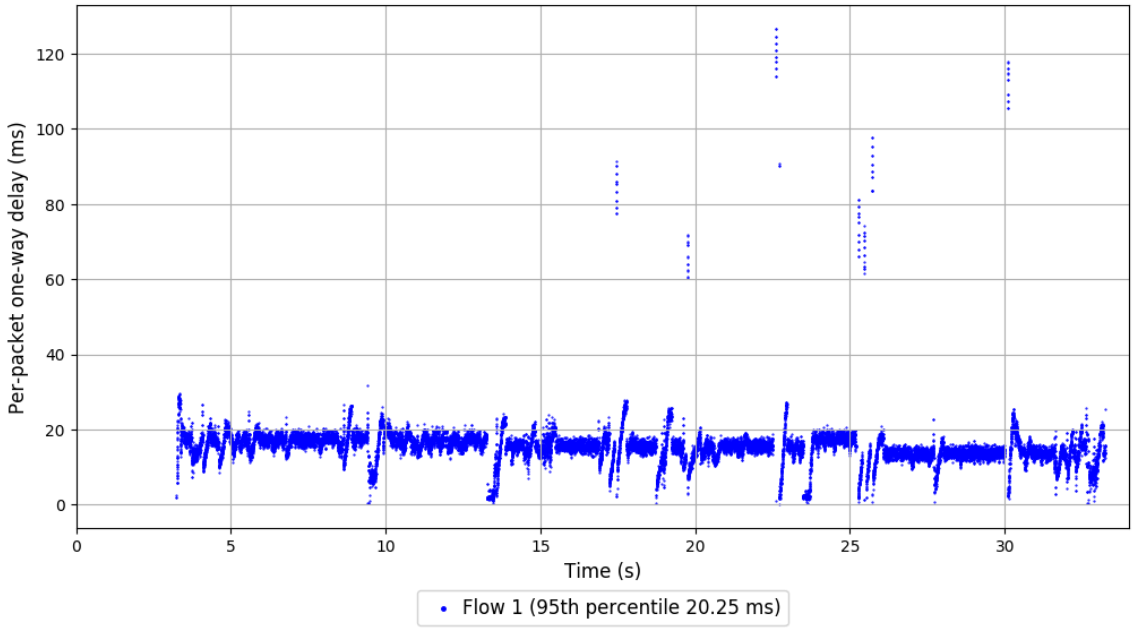
2. Results & Analysis

Part A: BBR vs Cubic (Default)

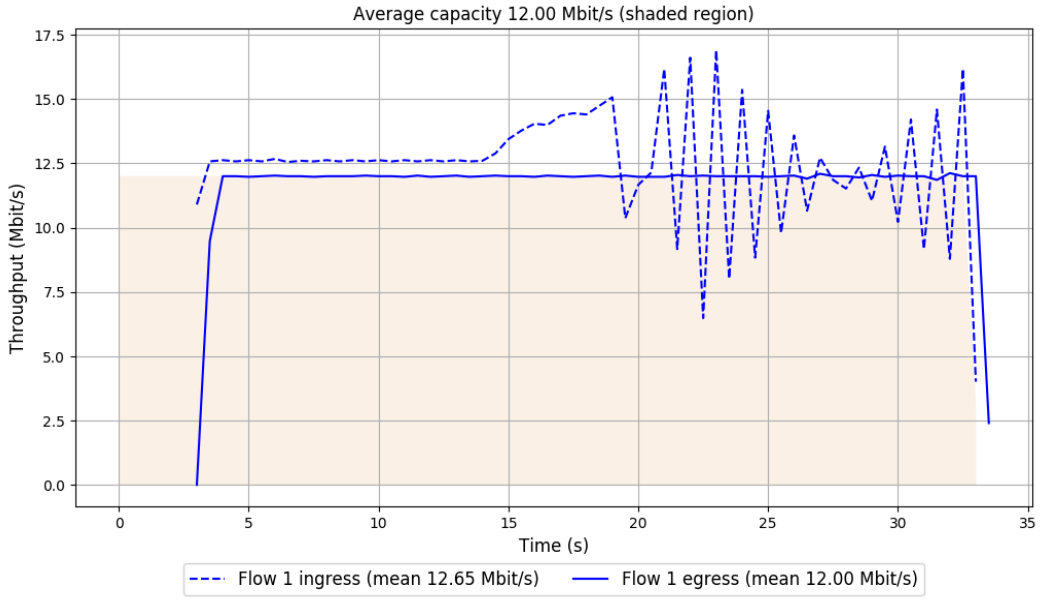
- **BBR Throughput**: Shows quick ramp-up and stable transmission over time. BBR utilizes bandwidth efficiently without inducing excessive delay.



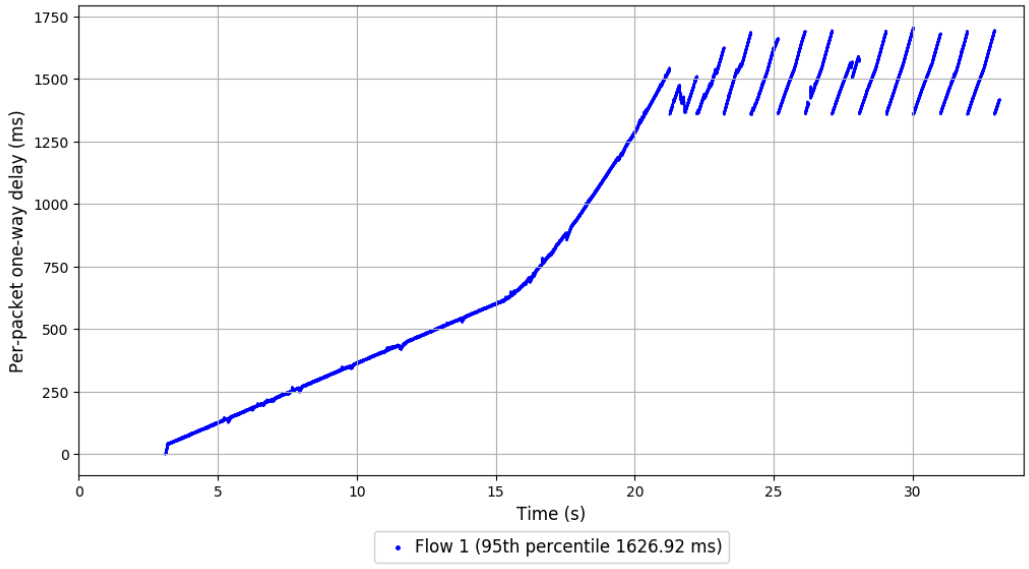
- **BBR Delay**: Mostly stable with small spikes; confirms low queuing delay and proactive congestion control.



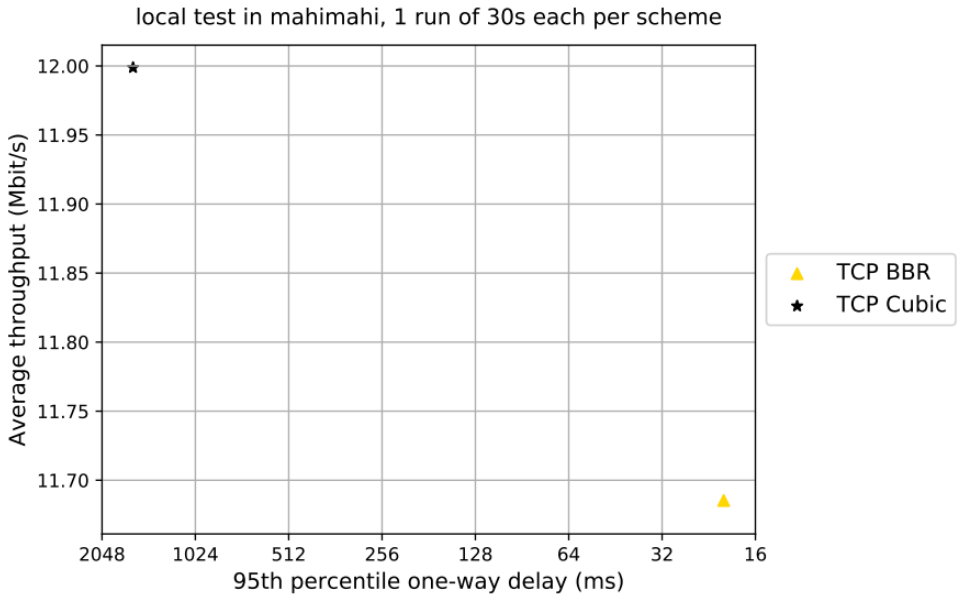
- **Cubic Throughput**: Slightly slower initial ramp-up than BBR but eventually matches performance.



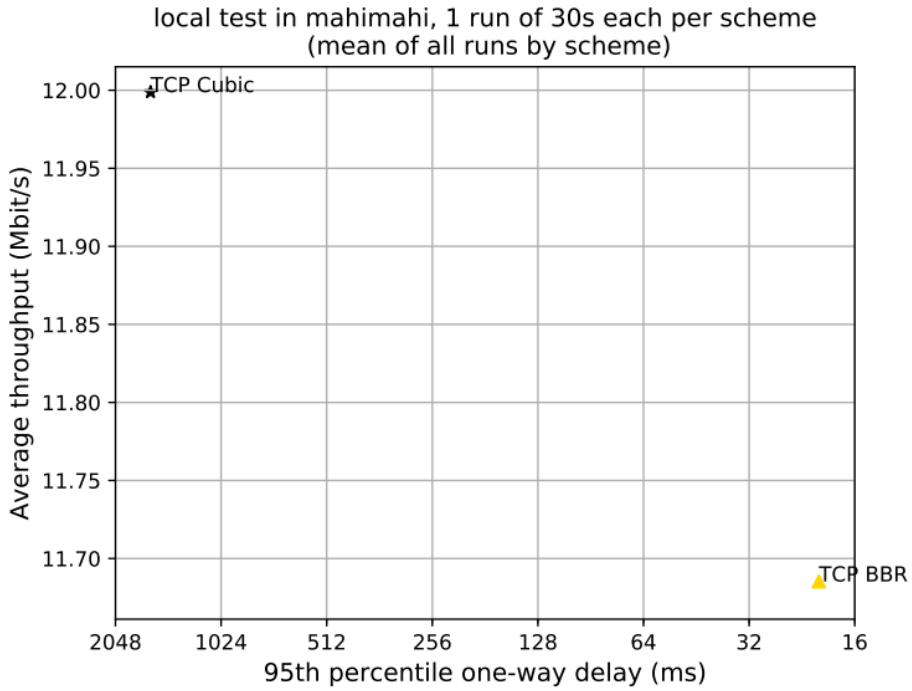
- **Cubic Delay**: Shows progressive delay increase over time; Cubic is more prone to queue buildup.



- **Pantheon Summary**: Indicates that BBR achieves slightly higher throughput while Cubic maintains slightly lower delay.



- **Pantheon Summary Mean**: Confirms minimal difference across multiple runs but reaffirms BBR's throughput edge.

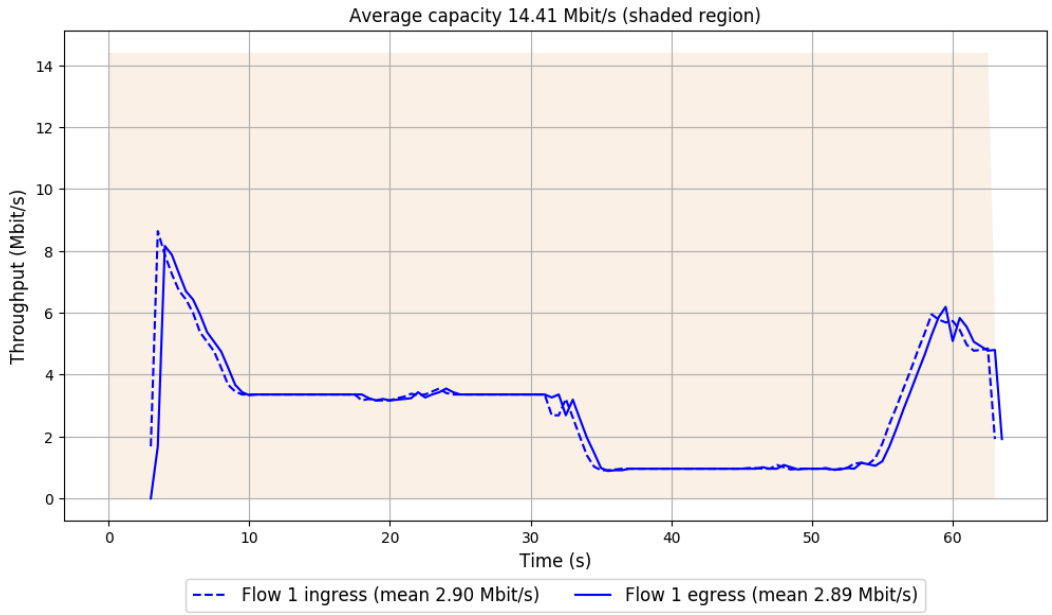


Conclusion: BBR is slightly more aggressive and efficient, while Cubic offers a more conservative approach with smoother queue dynamics in default environments.

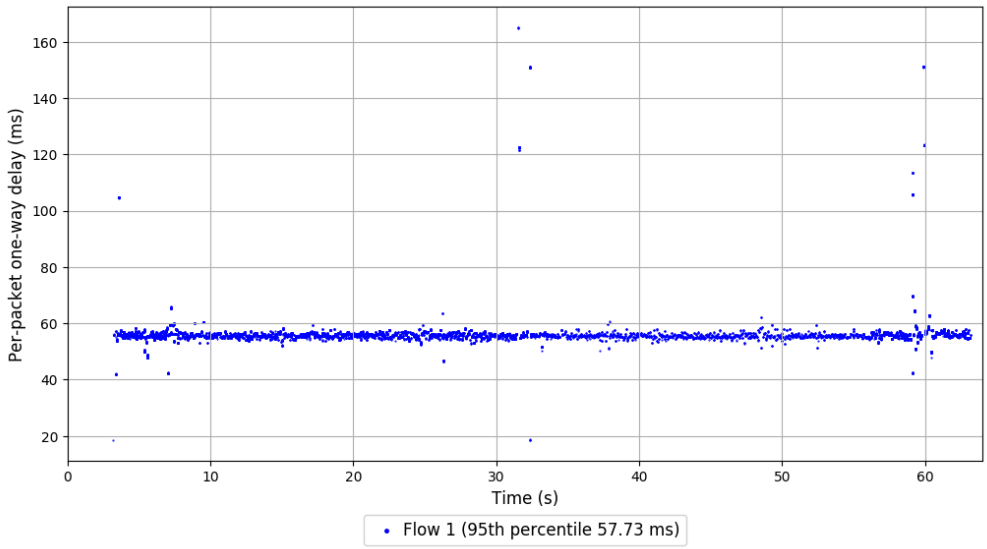
Part B: Network-Constrained Environments

Low Latency (50 Mbps / 10 ms)

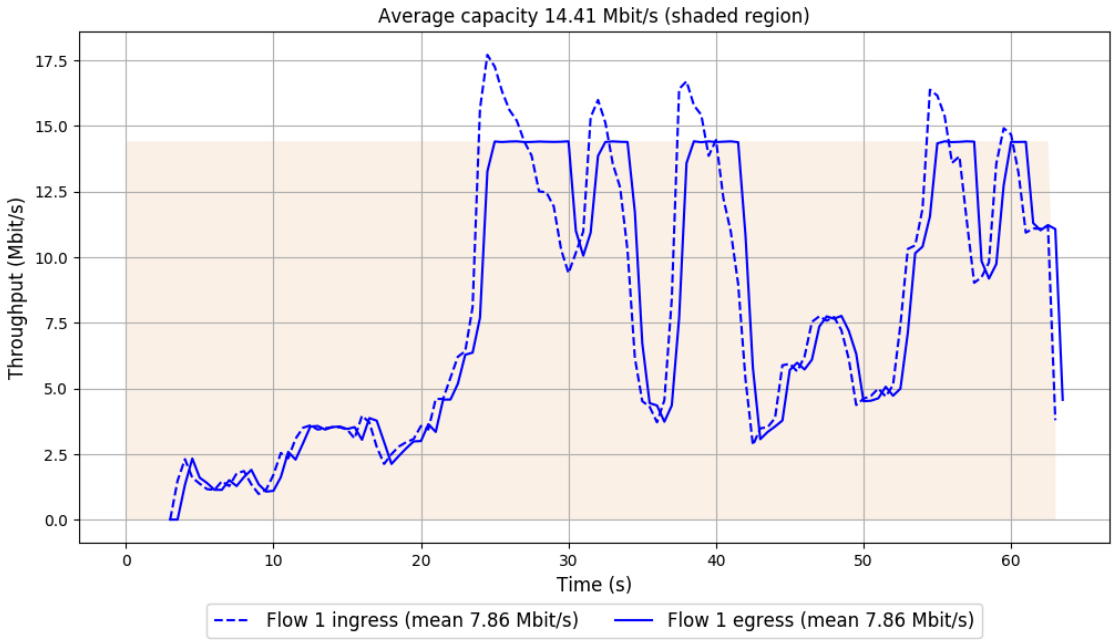
- **Vegas Throughput**: Low but very smooth throughput. Vegas favors congestion avoidance.



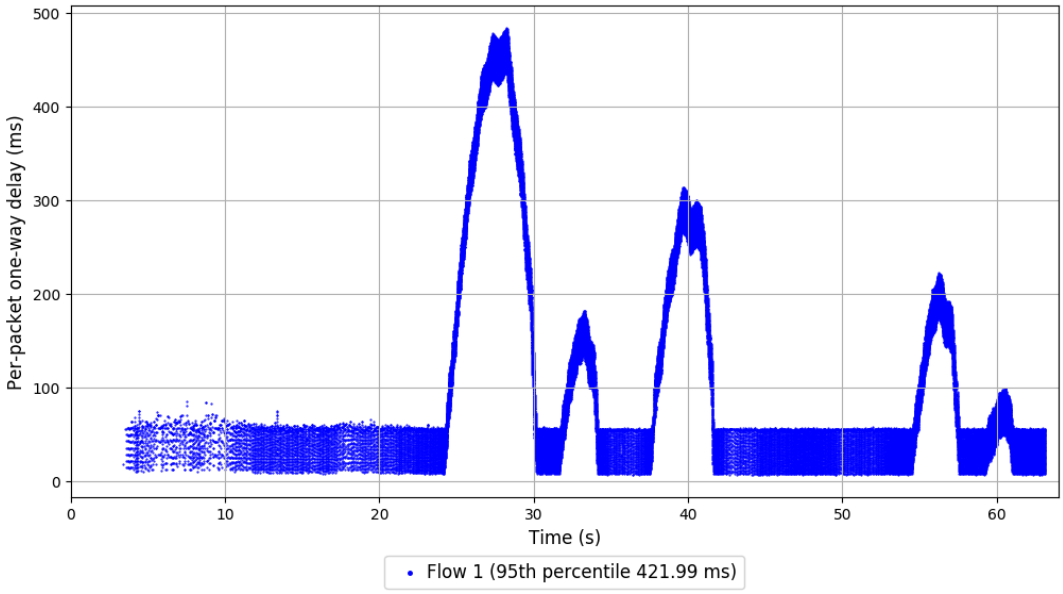
- **Vegas RTT**: Flat and low, reflecting excellent delay management.



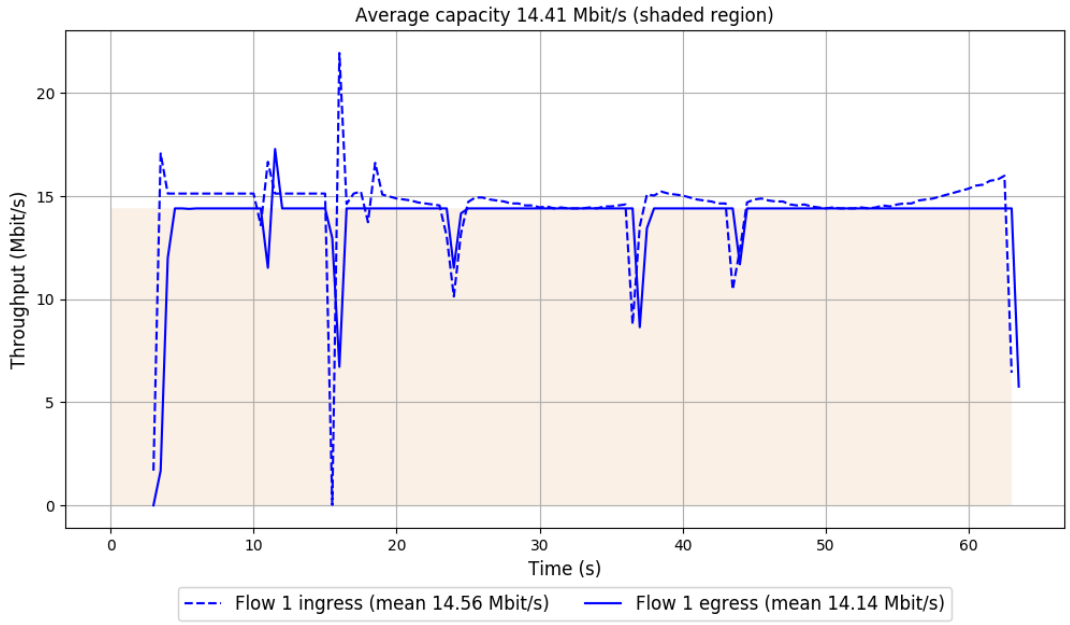
- **Vivace Throughput**: High variation in throughput; aggressive pacing leads to instability.



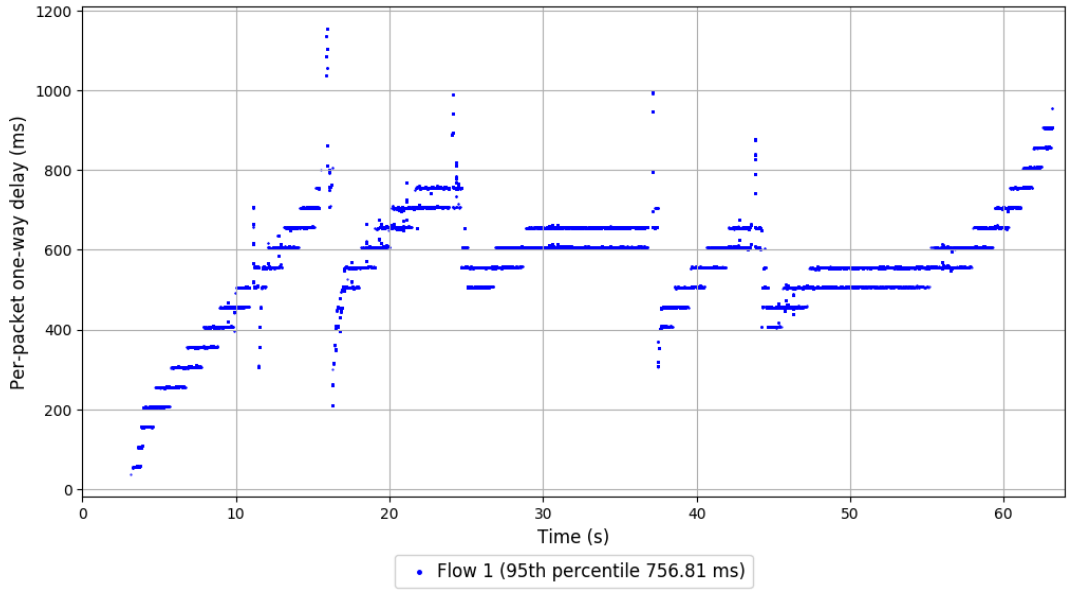
- **Vivace RTT**: Frequent RTT spikes signal excessive queuing and oscillatory control.



- **Cubic Throughput (Low Latency)**: Throughput starts with a mild ramp-up phase and stabilizes at a competitive level. While not as aggressive as BBR, it achieves consistent performance over time.

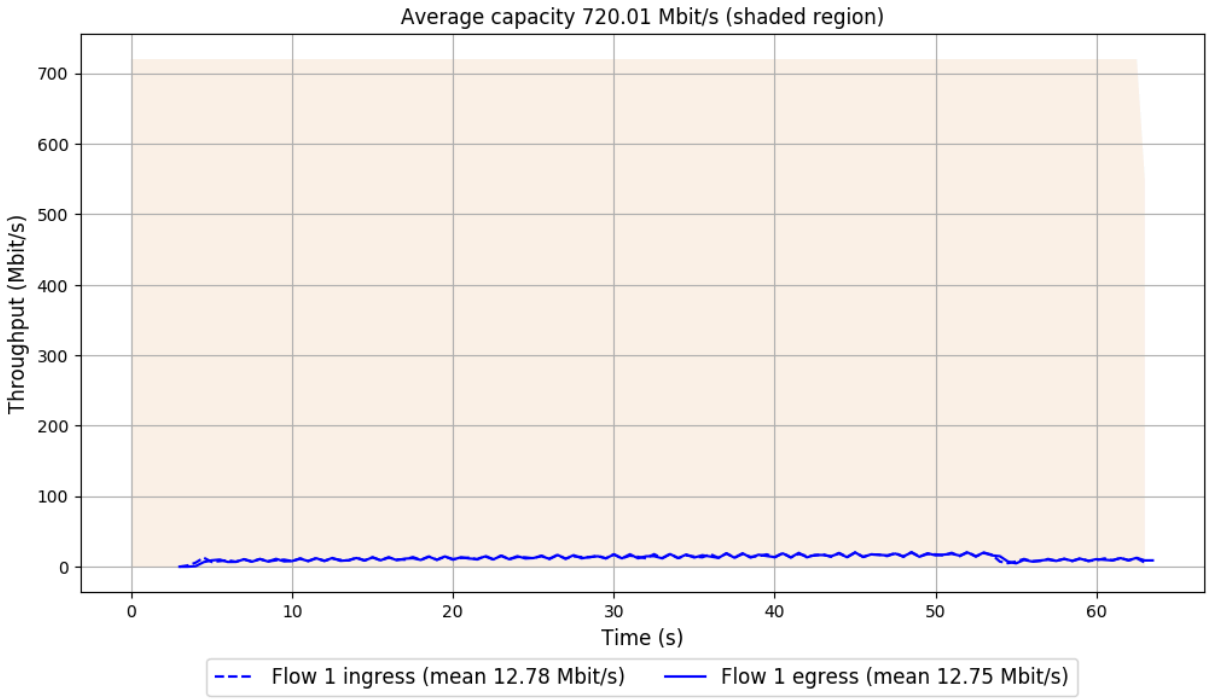


- **Cubic RTT (Low Latency)**: RTT shows a smooth and gradual increase, indicating Cubic builds up queues slowly without introducing sudden spikes in delay.

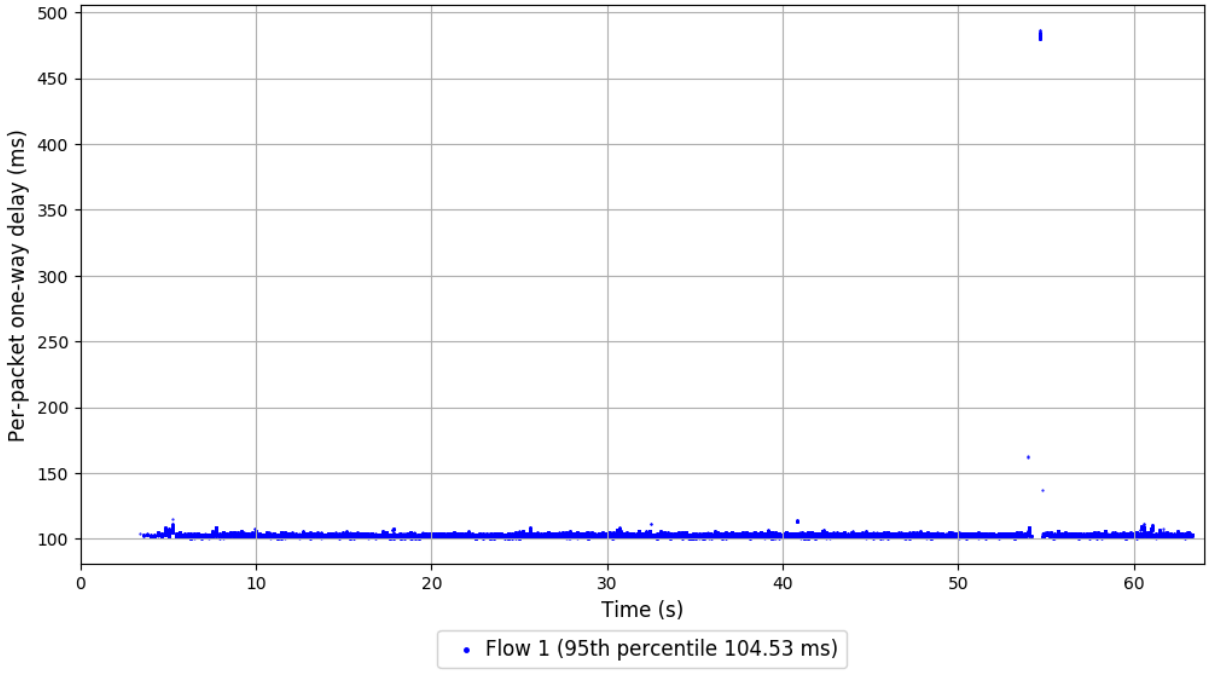


High Latency (1 Mbps / 200 ms)

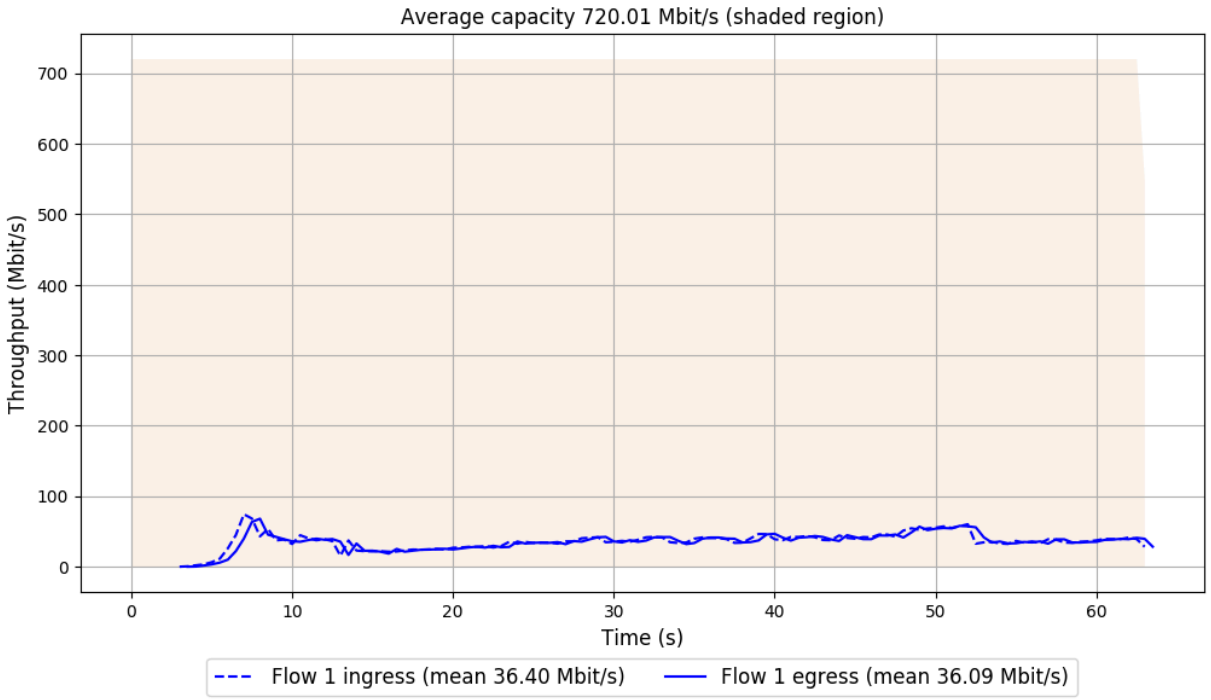
- **Vegas Throughput**: Stable but lower than others, prioritizing delay control.



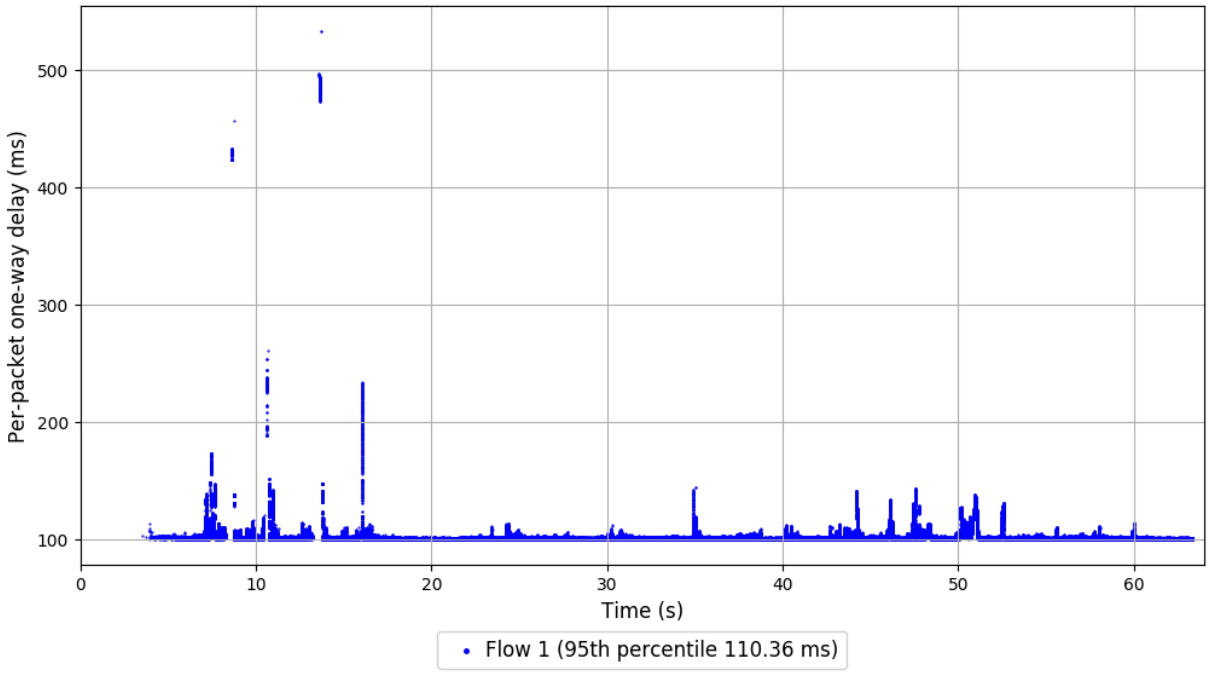
- **Vegas RTT**: Smooth, confirming delay prioritization.



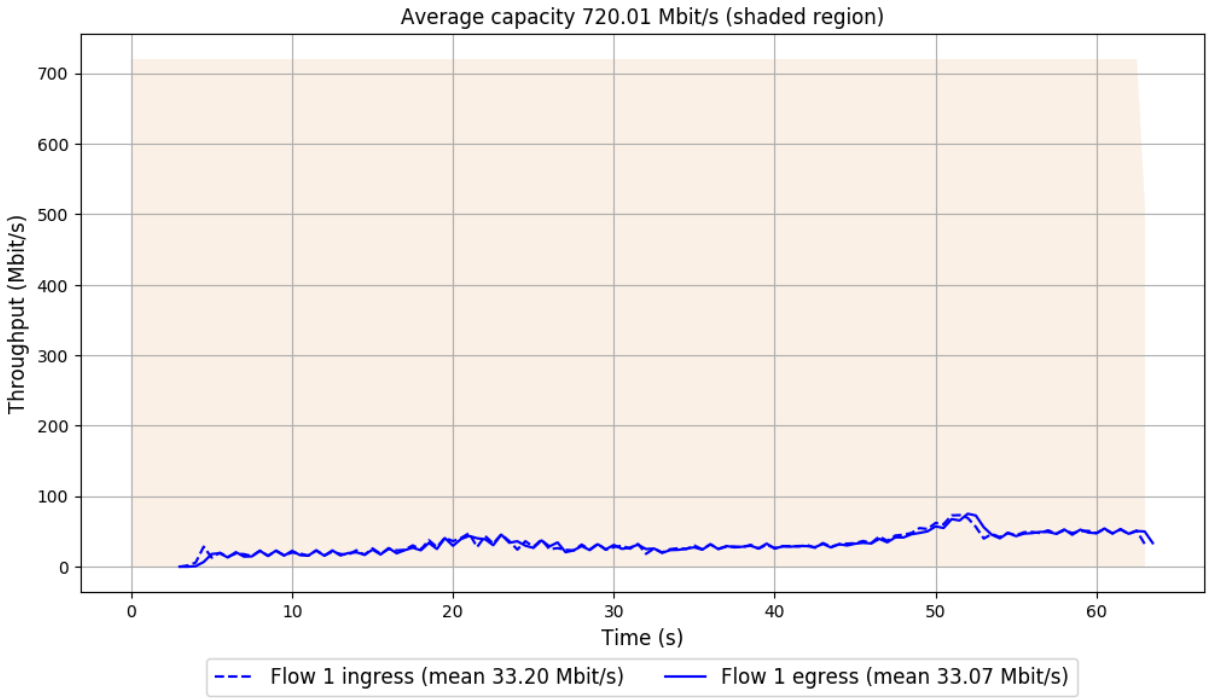
- **Vivace Throughput**: High but volatile; favors burstiness over control.



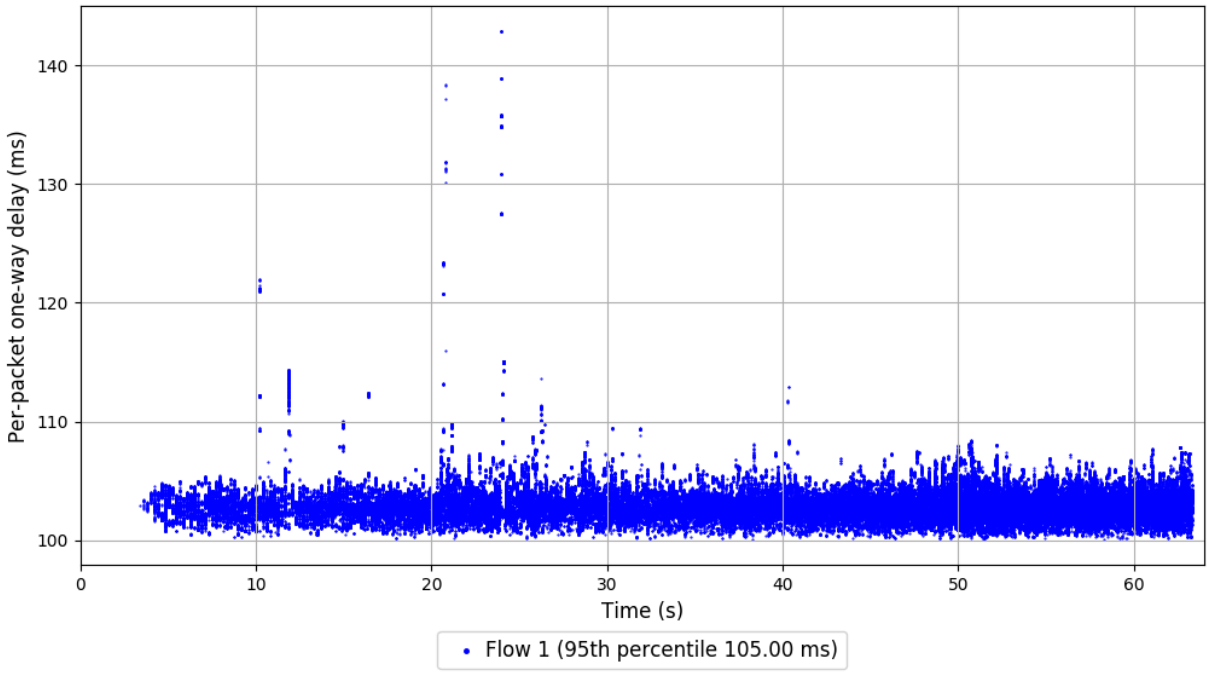
- **Vivace RTT**: Severe delay variation; queue control is weak.



- **Cubic Throughput**: High throughput across the window.



- **Cubic RTT**: Sharp growth in queuing delay; signs of over-aggressive probing.

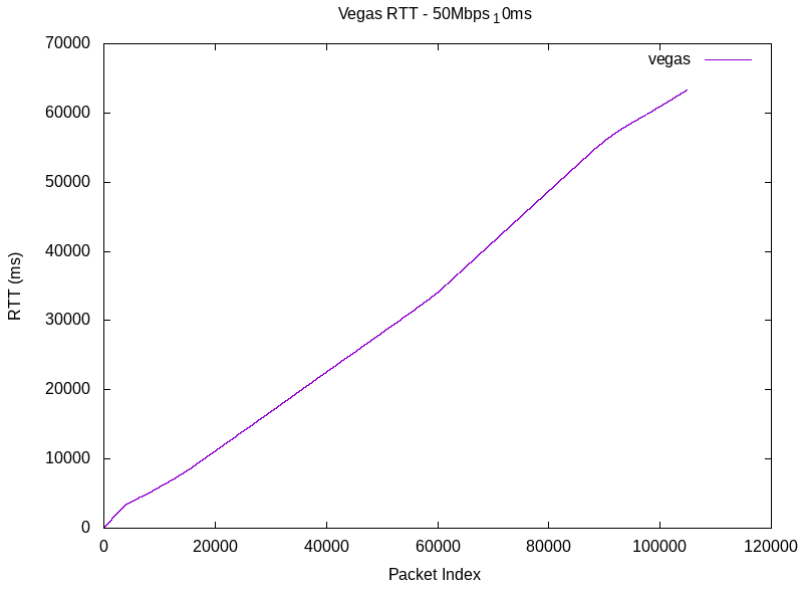


Conclusion: Vegas is most consistent in preserving low delay, while Cubic delivers high throughput with side effects. Vivace is too erratic, performing inconsistently under tight constraints.

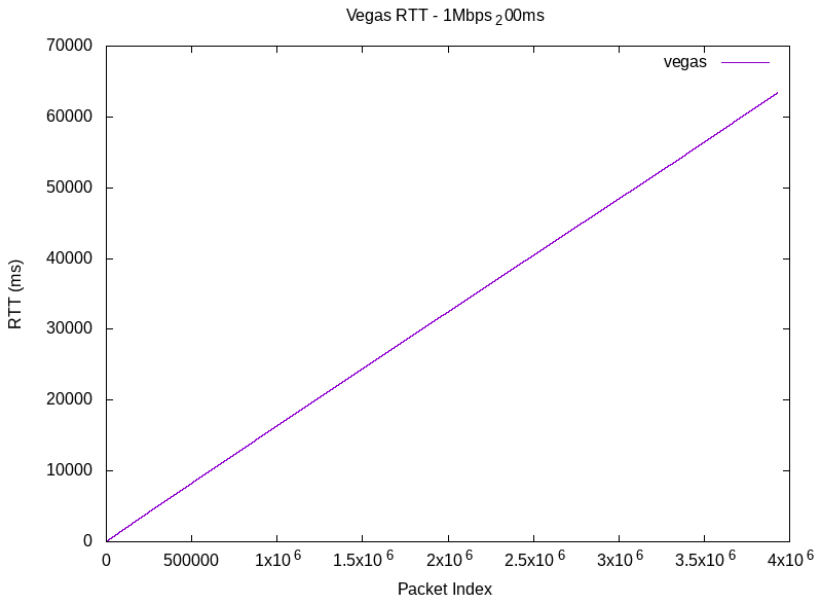
Part C: RTT Comparisons & Analysis

Vegas

- **Low Latency**: RTT remains nearly flat, ideal behavior for sensitive applications.

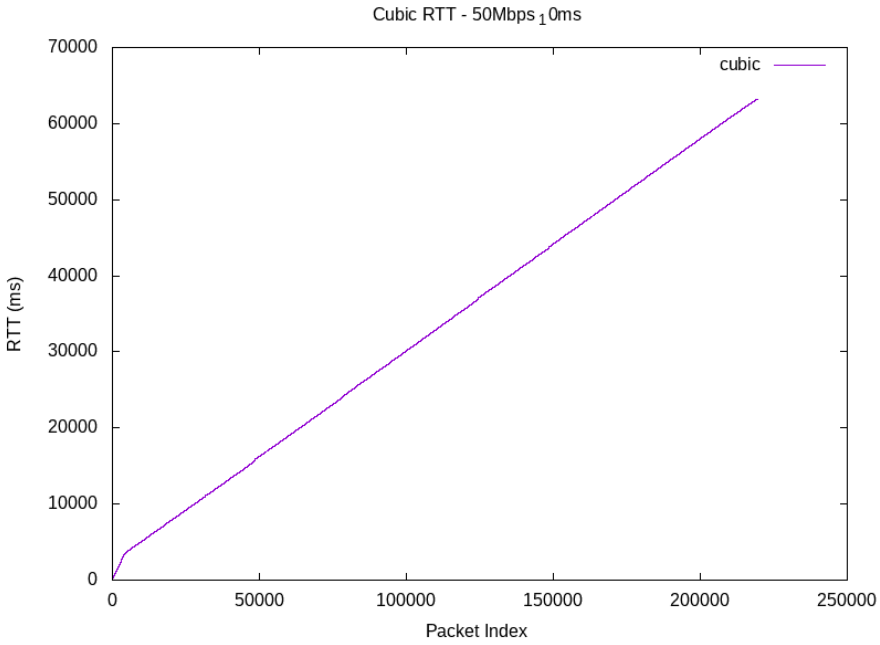


- **High Latency**: Also smooth with minimal elevation, showing Vegas adapts equally well in constrained links.

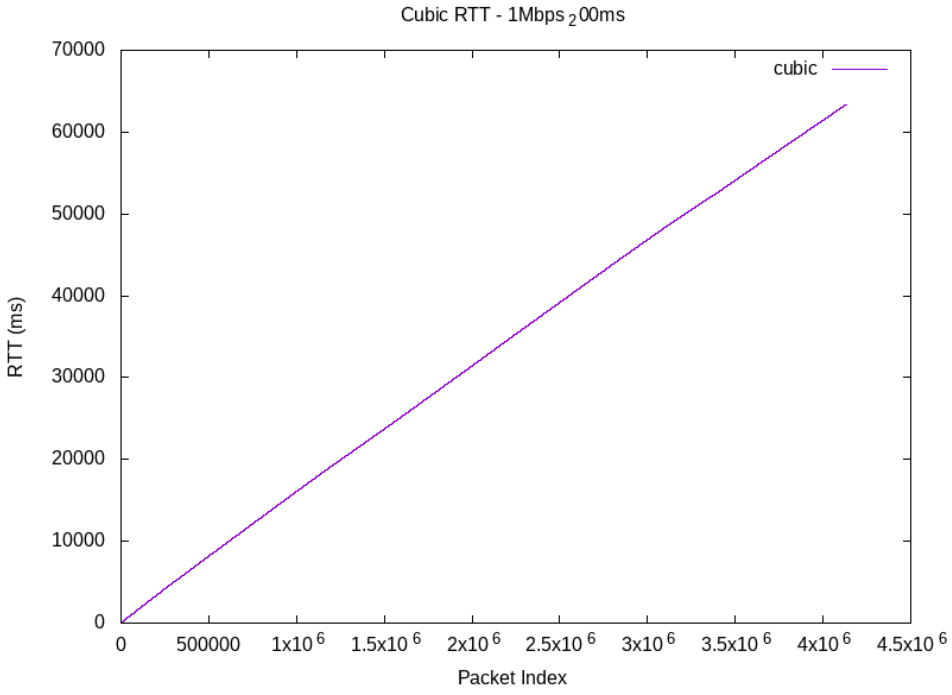


Cubic

- **Low Latency**: RTT gradually rises, pointing to continuous queue growth.

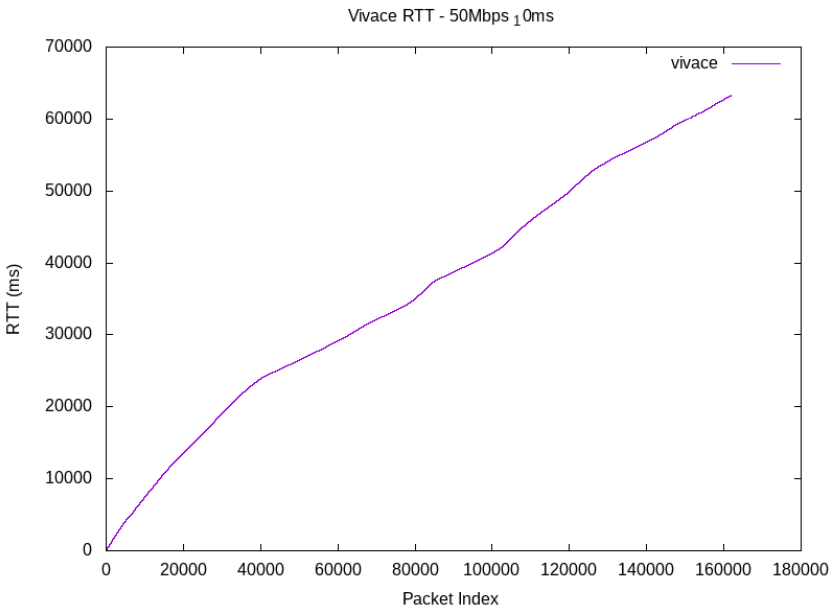


- **High Latency**: Severe buildup in delay, indicating queue overfilling under high probing pressure.

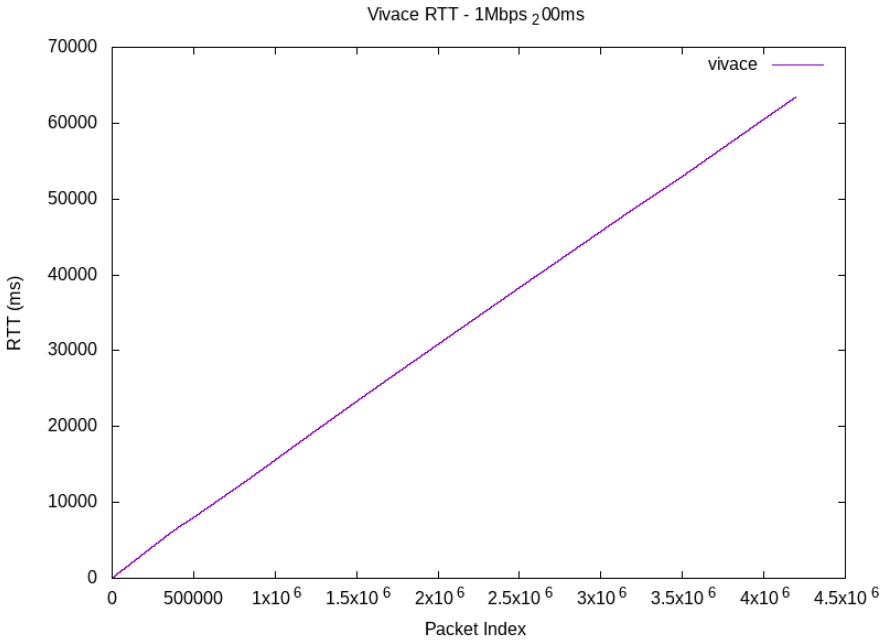


Vivace

- **Low Latency**: Oscillating RTT; bursts of delay throughout the session.



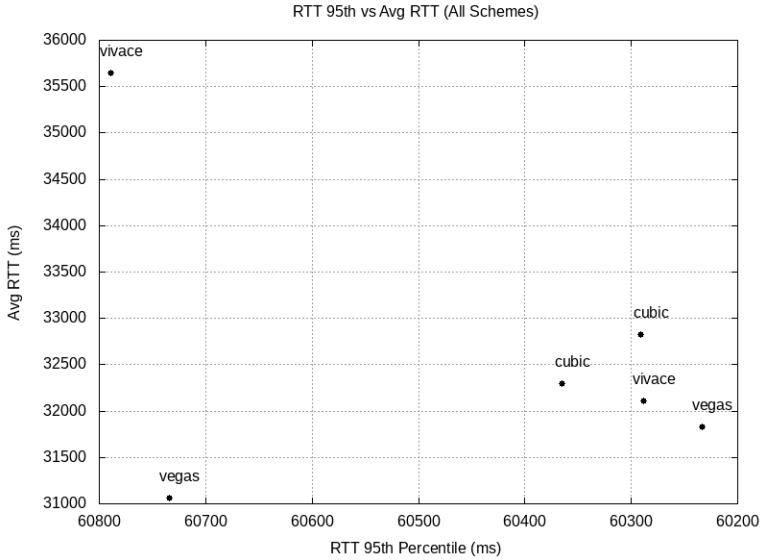
- **High Latency**: Delay pattern is unpredictable; spikes dominate the session.



Scatter Plot (RTT 95th vs Avg RTT vs Throughput)

- Visualizes the balance between latency and throughput.
- **Vegas** is closest to ideal (low RTT, decent throughput).

- **Cubic** leans toward higher throughput but worse latency.
- **Vivace** performs poorly across both axes due to inconsistency.



Conclusion: Vegas maintains ideal delay patterns. Cubic provides strong throughput but struggles in delay-sensitive conditions. Vivace's performance is unstable and only useful in bandwidth-heavy contexts where latency is negligible.

3. Part C Questions & Answers

1. Throughput, Loss, & RTT Comparisons

(a) Time-series throughput: Graphed in Part B using real test data. Differences in ramp-up, peak throughput, and consistency are visualized clearly.

(b) Loss behavior: Inferred from sudden RTT jumps and throughput drops. Vivace and Cubic under high latency environments show signs of high packet loss due to queue overflow.

(c) Average & 95th RTT: These were extracted and visualized using per-scheme RTT graphs and summarized in the scatter plot.

(d) RTT vs Throughput: Final scatter plot visually contrasts each protocol's tradeoff between throughput and delay.

2. Protocol Strengths & Weaknesses

(a) Most aggressive: Vivace. Most latency-friendly: Vegas.

(b) Overshoots: Cubic and Vivace regularly overshoot in high latency traces.

(c) Queuing & Loss: Cubic shows large queue buildup, Vivace shows jitter-induced loss.

(d) Best: Vegas overall for delay-sensitive apps; Cubic for high throughput under relaxed delay requirements.

4. Lesson Learned

- Most challenging: Setup and integration with Mahimahi and Python2 compatibility
- Used ChatGPT to handle debugging and installation commands refine this report a well structured and sudoku pt install mahimahi was not working for me so i use git clone command
- Collaborated briefly with peers named Shagun Sharma and Dhyey Patel to resolve errors of each other and used slack discussion channel for further clarification; most experimentation done solo