MPTCP Performance Optimization

| Hengshou Zhang | hzhan402@ucr.edu | Zelai Fang | zfang052@ucr.edu |

Abstract

The purpose of this study is to investigate the impact of different configurations (e.g., bandwidth, congestion algorithms, and loss rates) on data transmission over different paths in the network. By creating a virtual network using the Mininet tool and Wireshark for packet observation, we conducted three experiments to analyze the performance in different scenarios. We selected two representative congestion control algorithms for the study, namely Cubic and Reno. The study covers a variety of network environments and data paths, including fixed versus dynamic bandwidth, different packet loss rates, and single versus multi-path data transmission. This research has important implications for deeper understanding and optimization of network design and data flow.

Introduction

The focus of this project is on the research and optimization of network design. Our main objective is to understand and improve the performance of data transmission in different network environments and paths. Through the design and execution of a series of experiments, we evaluate how various factors, such as bandwidth, congestion control algorithms, and packet loss rate, impact the efficiency and stability of data transmission in specific network environments.

The primary questions we aim to address include: How does the allocated bandwidth affect data transmission across different paths? How do the chosen congestion control algorithms impact data transmission efficiency? And how does packet loss rate influence network performance? These questions are of significant importance as they directly influence the performance, efficiency, and reliability of network systems, all of which are critical factors in our increasingly data-driven world. By providing detailed experiments and analysis in a network setup with three paths under different conditions, our project contributes to the existing body of work and enhances the understanding of network design and optimization.

Building upon the foundation of Course Project 1, our research specifically evaluates the performance differences of different congestion control algorithms, such as Cubic and Reno, in handling network congestion. This contributes to a deeper understanding of the advantages and limitations of these algorithms and how to select and apply them in practical network design and optimization. Additionally, our experimental results reveal how factors such as bandwidth and packet loss rate impact the efficiency and stability of data transmission, aiding developers in better understanding and optimizing network performance.

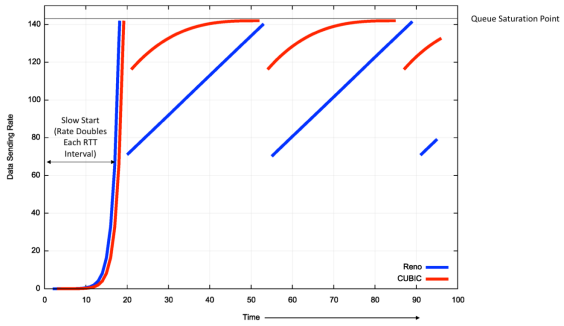
Related Works

Past research has focused on theoretical models and simulations to understand and predict the behavior of factors such as bandwidth, congestion algorithms, and packet loss rates in a network environment. These studies have provided the foundation and guidance for network design. However, they are usually not accompanied by an analysis of the interactions and interdependencies between these factors.

Our project focuses on actual experimental studies rather than just theory and simulation. We created a virtual network environment and conducted a series of experiments in this environment to gain insight into how bandwidth allocation, congestion algorithms (such as Cubic and Reno), and packet loss rates work together to affect network performance. This approach allows us to more closely model real-world network

scenarios and provides specific insights on how to optimize network design to improve performance.

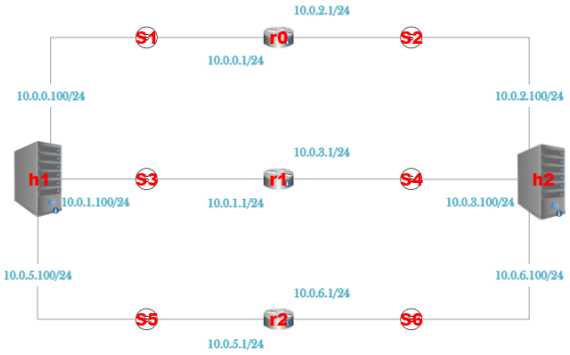
Cubic and Reno



Cubic's window adjustment is based on time rather than the traditional congestion window. It uses a cubic equation to determine the size of the congestion window, which is based on the time since the last congestion occurred.Cubic's algorithm is better adapted to network environments with large bandwidth and high latency.Reno uses congestion avoidance and fast recovery mechanisms to adjust the congestion window size. When network congestion is detected, the Reno algorithm reduces its window size to reduce the number of packets in the network. By comparing Cubic and Reno, our project further understands the different performance of these two congestion control algorithms in handling network congestion and data transmission.

Design

The project first extends the study based on course project 1 by adding complexity to the simulated system by setting up more subflows. Our experimental design revolves around three paths: path1, path2, and path3. We study in detail the effects of different configurations, including varying bandwidth (BW) settings, applying two different congestion control algorithms (Cubic and Reno), and introducing a 10% packet loss rate. We assume that each configuration will have different effects during data transmission, and that certain configurations may exhibit optimal performance under certain conditions.



Path Configuration

We define three unique paths based on the IP source and destination:

- Path1
  ip.src==10.0.0.100 and ip.dst==10.0.2.100
- Path2
  ip.src==10.0.1.100 and ip.dst==10.0.3.100
- Path3
  ip.src==10.0.5.100 and ip.dst==10.0.6.100

Each path is configured to have certain bandwidth settings, allowing us to simulate real-world network conditions where bandwidth can significantly vary.

Experiments

Bandwidth Variation

The first experiment investigates the impact of bandwidth (BW) variation over a path. We monitored network performance using Wireshark, a network protocol analyzer, to observe the data transfer between paths.
Three configurations were used:

- All paths have equal bandwidth (BW).
- Path2 has a lower BW compared to Path1 and Path3.
- Each path has a different BW.

Congestion Control Algorithms

The second experiment introduces two congestion control algorithms (Cubic and Reno) to

examine their influence on data transmission across the paths under specific bandwidth settings. For consistency, we set the bandwidth to each path 20-30-10 for this experiment and keep the other parameters unchanged.

Under Cubic, the transmission rate is expected to be higher in Path1 and Path2. This is because Cubic is a more modern and intelligent congestion control algorithm that makes more efficient use of network bandwidth, especially in high-bandwidth, high-latency network environments.Cubic controls the size of the congestion window by using a cubic time function that allows the window to grow faster to increase throughput. Under Reno, the transmission rate is expected to be relatively low over all paths. Reno is a more traditional congestion control algorithm that uses an additive increase and multiplicative decrease (AIMD) strategy to adjust the congestion window size.Reno is more conservative in the face of network congestion, which may result in its inability to fully utilize the available bandwidth in high bandwidth network environments.

Loss Rate and Ping Test

The third experiment introduces a 10% loss rate to observe its effect on data transmission with different congestion control algorithms.

Under Cubic, due to its faster window growth, we expect that Cubic may not perform quickly enough in adapting to network conditions when faced with packet loss. This may result in Cubic not handling packet loss as effectively under certain network configurations. Under Reno, we expect that Reno may perform more robustly in the face of packet loss due to its more conservative handling of network congestion.Reno rapidly reduces the window size when packet loss is detected, thus reducing the impact of packet loss.

Also, we conducted a ping test to evaluate the round trip time (RTT) and packet loss rates under the 10% loss rate and under the two different congestion algorithms. Since Reno is more conservative in handling congestion, we predict that the packet loss rate and RTT may be lower when using Reno's algorithm.

Implementation

In implementing this project, we use Python as the main programming language and operate with Mininet and Wireshark. mininet is used to create virtual networks in a controlled environment to simulate various network paths and bandwidth settings, while Wireshark is responsible for capturing and analyzing the packets sent during the experiments so that we can understand and document the behavior of various congestion control algorithms in different environments.

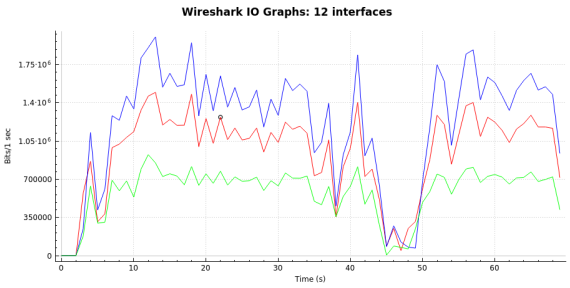
Our experiments are based on the open source code provided by Mininet, to which we extend. Mininet provides the basic framework for creating and managing virtual networks, to which we add control over congestion control algorithms and bandwidth settings, as well as the ability to log and analyze the results of our experiments.

The main challenge we faced during the implementation was to accurately simulate network congestion and packet loss in a virtual network environment. To solve this problem, we make use of the built-in features of the Linux system, such as tc and netem, which can help us precisely control the network bandwidth and packet loss rate.

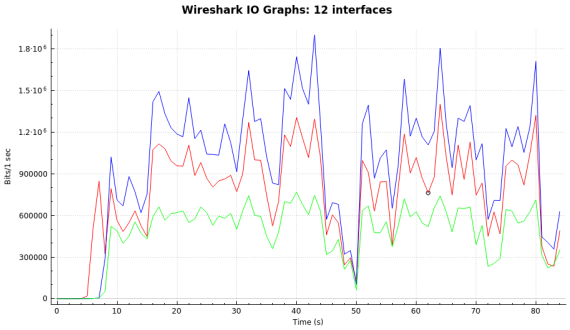
In total, we wrote about 200 lines of Python code throughout the project. The code included sections for creating and managing virtual networks, sections for tuning network bandwidth and congestion control algorithms, and sections for logging and analyzing experimental results.

Results & Analysis

Our experiments yielded interesting results as we observed different performance metrics for two congestion control algorithms (Cubic and Reno) with various bandwidth allocations and loss rates.

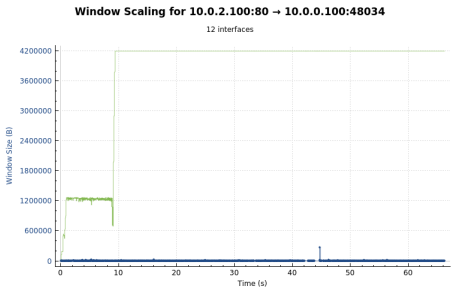


[Fig. Apply Cubic congestion algorithm]

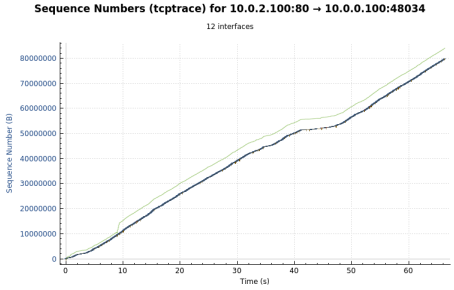


[Fig. Apply Reno congestion algorithm]

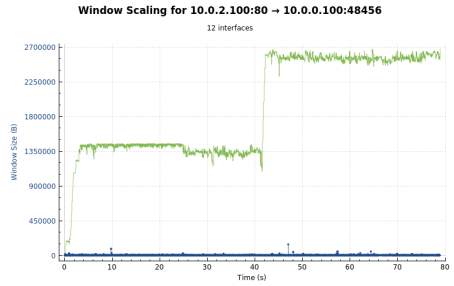
We observe different trends when applying the Cubic and Reno congestion algorithms. With the Cubic algorithm, we note that the data transfer rate is more stable compared to Reno, which exhibits significant fluctuations. These findings reflect the intrinsic nature of these algorithms, where Cubic is a more aggressive algorithm that shows a more stable data transfer rate since it is less sensitive to network changes. This feature is reflected in our experiments, where Cubic shows more consistent performance under equal and different bandwidth allocations and in the face of packet loss. On the other hand, Reno is more conservative and reacts quickly to perceived network congestion, resulting in more fluctuating data transfer rates. As we observed in our experiments, this leads to higher variability in the data transfer rate, especially when packet loss is introduced.



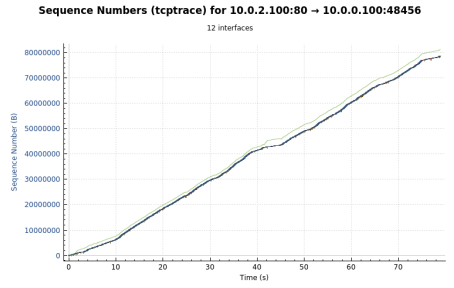
[Fig. Path1 window scaling with Cubic algorithm]



[Fig. Path1 sequence numbers with Cubic algorithm]



[Fig. Path1 window scaling with Reno algorithm]



[Fig. Path1 sequence numbers with Reno algorithm]

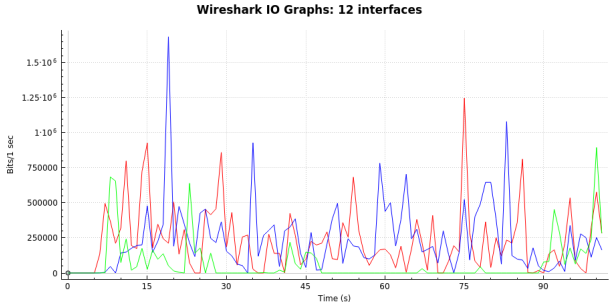
From the data in Experiment 2, we can see the difference in data transmission efficiency and stability between the two congestion control algorithms, Cubic and Reno, and how they perform when dealing with different data paths.

In the Cubic algorithm, we see that in Path1, Cubic quickly reaches a stable window size at startup, and this fast response feature can help to quickly adapt to the network environment and reduce the transmission delay at startup. At the

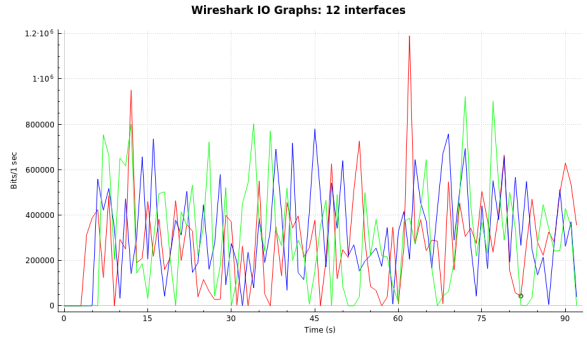
same time, Cubic shows good stability as the Average throughput (bits/s) is basically maintained within a certain range throughout the transmission process. However, the sudden increase in window size and smaller increase in sequence number may reflect Cubic's overreaction when encountering congestion, which may be a characteristic of Cubic as a more aggressive algorithm.

For the Reno algorithm, Path1 starts relatively slowly, which may be due to the nature of Reno's more conservative detection of network bandwidth. However, during operation, Reno shows large fluctuations in both window size and throughput, indicating that Reno is less adaptive in dealing with network congestion and bandwidth changes.

As a whole, Cubic may have better performance in high-speed and changing network environments, but may suffer from overreaction when facing sudden network congestion. Reno, on the other hand, may be better suited for stable network environments, although it is less start-up and adaptive.



[Fig. 10% Loss Rate with Cubic algorithm]



[Fig. 10% Loss Rate with Reno algorithm]

For the 10% packet loss rate with the Cubic algorithm, we observe many peaks in all three paths in Wireshark's IO plot, with no smooth straight lines or more stable fluctuations. This indicates that the data transfer rate is highly variable under such conditions, possibly due to packet loss. In particular, path 3 does not transmit any data most of the time, which may be due to its high packet loss rate, causing TCP to stop sending data. In contrast, path 1 and path 2 remain active, indicating that the Cubic algorithm still has a good performance in handling high packet loss rates.

With a 10% packet loss rate with the Reno algorithm, we observe that the data transfer rates of all three paths in Wireshark's IO graph are very active, peaking quickly from 0 or a lower rate, and then dropping rapidly in a repeated cycle. This indicates that the Reno algorithm tries to increase the transmission rate by increasing the window size in the face of high packet loss, but this also leads to a further increase in packet loss, which results in large fluctuations in the rate. At the same time, we also see that the Reno algorithm is able to maintain data transmission for all paths in the case of high packet loss rate, which indicates that the Reno algorithm is more aggressive in dealing with packet loss and tries to achieve the optimal transmission rate by continuously adjusting the window size.

Parameter	Cubic	Reno
Packets Transmitted	100	100
Packets Received	84	86
Packet Loss Rate	16%	14%
Transmission Time (ms)	105741	102224
Minimum RTT (ms)	0.040	0.048
Average RTT (ms)	0.076	0.541
Maximum RTT (ms)	0.256	37.478
RTT Standard Deviation (ms)	0.029	4.008

We tested the RTT and packet loss rate of both algorithms. The results show that the Cubic algorithm has a lower average RTT but a higher packet loss rate, while the Reno algorithm has a higher average RTT but a lower packet loss rate. This indicates that the Cubic algorithm is able to maintain a low RTT in the case of handling high packet loss, but it may lead to an increase in packet loss, while the Reno algorithm has a higher RTT in the case of handling high packet loss, but the packet loss can be better controlled.

Through this experiment, we can see that the two congestion control algorithms, Cubic and Reno, have different processing strategies and performance in the face of high packet loss. the Cubic algorithm may increase packet loss while maintaining a low RTT, while the Reno algorithm may cause an increase in RTT while controlling packet loss.

Lessons Learned

In this project, we learned and practiced in depth to understand how different congestion control algorithms perform in various network environments. Initially, the main challenge we faced was to understand the complexity of these algorithms and the multiple factors that affect their performance. However, through a comprehensive literature study and a series of hands-on experiments, we successfully overcame these challenges and gained a deeper understanding of these complex concepts. Further challenges came from the interpretation and analysis of a large amount of experimental data. Initially, we had difficulties with tools like Wireshark, but with continued practice and learning, we mastered these tools and were able to accurately identify and understand subtle differences and trends in the data, again highlighting the importance of data analysis in network engineering. Overall, this project was an insightful learning process that deepened our understanding of the value of deep understanding, practical application, data interpretation, and effective teamwork in network engineering.

This course was our first exposure to computer networking-related content, and we had limited prior knowledge in field. To compensate for this, we dedicated a significant amount of time to studying foundational concepts and actively engaged with open-source communities to learn about MPTCP-related projects. While we made every effort to implement the experiment in detail, there were still gaps in our understanding, which we aim to address through continued learning. In terms of contributions, both team members made equal efforts. We collaborated during the mid-term project to assist each other, analyze the collected data, and determine the project's direction. Finally, we summarized and presented our findings.

Reference

[1] Mininet. (n.d.). linuxrouter.py. GitHub. Retrieved from https://github.com/mininet/mininet/blob/master/examples/linuxrouter.py

[2] Raiciu, C., Pluntke, C., Barre, S., Greenhalgh, A., Wischik, D., & Handley, M. (2011). Multipath TCP: From theory to practice. In Network Protocols (ICNP), 2011 19th IEEE International Conference on (pp. 1-10). IEEE. Retrieved from https://www.cs.ucr.edu/~jiasi/teaching/cs204_spring19/papers/MPTCP11.pdf

[3] DemonLee. (n.d.). TCP congestion control algorithm: Cubic. DemonLee Tech. Retrieved from https://demonlee.tech/archives/2209001

[4] Sanders, C. (2017). Practical Packet Analysis: Using Wireshark to Solve Real-World Network Problems (3rd ed.).

[5] Huston, G. (2017, May 9). BBR - the new kid on the TCP block. APNIC Blog. Retrieved from https://blog.apnic.net/2017/05/09/bbr-new-kid-tcp-block/

[6] Peterson, L. L., Brakmo, L., & Davie, B. S. (2022). TCP Congestion Control: A Systems Approach. Systems Approach, LLC.