FTRL-WRR: Learning-Based Two-Path Scheduler for LEO Networks

by

Daoping Li
B.Eng., Henan University, 2022
B.IT., Victoria University, 2022

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

FTRL-WRR: Learning-Based Two-Path Scheduler for LEO Networks

by

Daoping Li
B.Eng., Henan University, 2022
B.IT., Victoria University, 2022

Supervisory Committee

Dr. Jianping Pan, Supervisor
(Professor, Department of Computer Science)

Dr. Jaya Prakash Champati, Departmental Member
(Assistant Professor, Department of Computer Science)

Supervisory Committee

Dr. Jianping Pan, Supervisor
(Professor, Department of Computer Science)

Dr. Jaya Prakash Champati, Departmental Member
(Assistant Professor, Department of Computer Science)

ABSTRACT

Multipath QUIC is inspired by the resource pooling principle, aiming to make a collection of resources behave as a single pool. However, current multipath schedulers tend to prioritize specific metrics like Round-Trip Time (RTT) or congestion window, often overlooking strategies that enhance overall resource usage and reduce flow completion time. This can lead to resource underutilization in high dynamic settings, such as those involving Low Earth Orbit (LEO) satellites. Addressing this challenge requires efficient traffic allocation to maximize bandwidth utilization. In this thesis, we verify that the relationship between traffic distribution and throughput in a two-path scenario resembles a quasi-concave function. Accordingly, we formulate the traffic allocation across two paths as a 1-dimensional optimization problem. To solve the two-path scheduling problem in dynamic environments, we introduce the FTRL-WRR algorithm. This approach integrates a Follow The Regularized Leader (FTRL) learner, ADWIN2 distribution change detector, and Weighted Round Robin (WRR) scheduler to enhance bandwidth utilization. We validate the effectiveness of the algorithm through extensive emulation and real-world testbed experiments, demonstrating consistent reduction in completion time across a range of scenarios. Additionally, we discuss the algorithm's limitations and suggest directions for future research.

Table of Contents

Supervisory Committee **ii**

Abstract **iii**

Table of Contents **iv**

List of Figures **vi**

Acknowledgements **vii**

Dedication **viii**

1 Introduction **1**
- 1.1 Main Contributions . . . 2
- 1.2 Thesis Overview . . . 4

2 Background **5**
- 2.1 QUIC . . . 5
- 2.2 Multipath QUIC and Schedulers . . . 7
  - 2.2.1 MinRTT . . . 7
  - 2.2.2 BLEST . . . 8
  - 2.2.3 ECF . . . 9
  - 2.2.4 Peekaboo . . . 10
  - 2.2.5 Weighted Round Robin . . . 13
- 2.3 Two-path Transmission Model . . . 14
- 2.4 Bandit Convex Optimization . . . 17
  - 2.4.1 Bandit Gradient Descent . . . 17
  - 2.4.2 Follow The Regularized Leader . . . 19

3 FTRL-WRR **24**

3.1 FTRL-based Learner 24
3.2 ADWIN2-based Distribution Change Detector 29
3.3 WRR-based Scheduler 31
3.4 Complexity Analysis 33

4 Experiments and Evaluations **35**
4.1 Starlink Emulator 35
4.2 Evaluation with Customized Topology 37
4.3 Study with Different Congestion Control Algorithms 39
4.3.1 Environment without Intentional Packet Loss 39
4.3.2 Environment with Intentional Packet Loss 42
4.4 Evaluation on Starlink "Bent-Pipe" + Cellular 43
4.5 Evaluation on Starlink ISL + Cellular 44
4.6 Evaluation on Starlink ISL + Starlink ISL 45
4.7 Evaluation on Starlink ISL + OneWeb 46
4.8 Real Starlink Testbed 48
4.8.1 Pacing 48
4.8.2 Evaluation on Real Starlink + Fiber 48

5 Future Work and Conclusions **50**
5.1 Limitations and Future Work 50
5.2 Conclusions 51

Bibliography **54**

List of Figures

Figure 1.1 QUIC and MPQUIC. 2

Figure 2.1 1-RTT Handshake [12] 6
Figure 2.2 0-RTT Handshake [12] 7
Figure 2.3 Throughput with Different X_t. 15
Figure 2.4 s(x) and f(x). 22

Figure 3.1 Timestep Updating Example. 25
Figure 3.2 Changes of r and η. 29
Figure 3.3 FTRL-WRR. 33

Figure 4.1 Emulator Topology. 36
Figure 4.2 Emulator. 36
Figure 4.3 Flow Completion Time. 38
Figure 4.4 Flow Completion Time (BBRv1). 40
Figure 4.5 Flow Completion Time (BBRv3). 40
Figure 4.6 Flow Completion Time (Cubic). 40
Figure 4.7 Flow Completion Time in Lossy Environment (Cubic). 42
Figure 4.8 Results from Starlink "Bent-Pipe" + 5G. 43
Figure 4.9 Results from Starlink ISL + 5G. 45
Figure 4.10 Results from Starlink ISL + Starlink ISL 46
Figure 4.11 Results from Starlink ISL + OneWeb. 47
Figure 4.12 Results from Starlink + Fiber. 49

ACKNOWLEDGEMENTS

I would like to thank:

my parents, for their unwavering support throughout my academic journey. Their generous financial assistance has enabled me to pursue my studies without worry, allowing me to focus on my research. Their encouragement and belief in me have been a constant source of motivation. I am forever grateful for their love and sacrifices.

Prof. Jianping Pan, for his outstanding mentorship and continuous guidance throughout my Master's program. His depth of knowledge and encouragement of independent thinking has been crucial in helping me refine my ideas and achieve significant progress. He has always been available to offer insightful feedback and challenge me to think critically, which has led to the development of my problem-solving skills and technical expertise. His mentorship extended beyond research, teaching me essential professional skills, including effective communication, collaboration, and maintaining a high standard of work. I am extremely fortunate to have worked under his supervision, and his guidance will undoubtedly have a lasting impact on my future career.

my labmates, for creating an environment of mutual support and collaboration that has been both intellectually stimulating and personally enriching. I am thankful for their feedback during our discussions, their assistance in times of need, and their valuable questions and suggestions during group meetings. Our shared experiences have not only contributed to the progress of my research but also fostered lasting friendships that I will cherish beyond this academic journey.

DEDICATION

To everyone who helped me through my academic journey.

Chapter 1

Introduction

The development of LEO satellite constellations, particularly in mega-constellation networks like Starlink, has attracted significant attention due to their ability to provide high bandwidth and low latency communication. As of July 2024, Starlink has launched over 6,718 satellites, with 6,236 currently in orbit and 6,163 operational [1]. The expansion and integration of the LEO networks with terrestrial networks enhance resource aggregation and resilience to connectivity failure [7]. While Starlink sometimes offers higher bandwidth compared to terrestrial networks such as cellular networks, it often exhibits greater variability in latency and bandwidth than its terrestrial counterpart [21]. In this context, QUIC, an encrypted, multiplexed transport layer protocol built on User Datagram Protocol (UDP), has shown promising advantages. Research highlights its efficiency in Integrated Satellite and Terrestrial Network (ISTN) environments, particularly in 0-RTT handshaking communication and loss recovery [29]. These features, along with its extension to handle multiple paths simultaneously, make QUIC particularly suitable for environments involving LEO satellites.

In a multipath transmission scenario, the scheduler distributes traffic across various interfaces based on specific policies. For transport layer protocols, multipath transmission can be seen as a black box, hiding the details of link components and routing from the sender. The sender schedules packets to different paths and receives statistical feedback on metrics such as RTT and packet loss rate. This feedback is used to update the congestion window (CWND) and guide future scheduling decisions.

Two-path transmission is a representative special case of multipath transmission. Figure 1.1 illustrates QUIC and its multipath extension, multipath QUIC (MPQUIC), specifically demonstrating a two-path scenario.

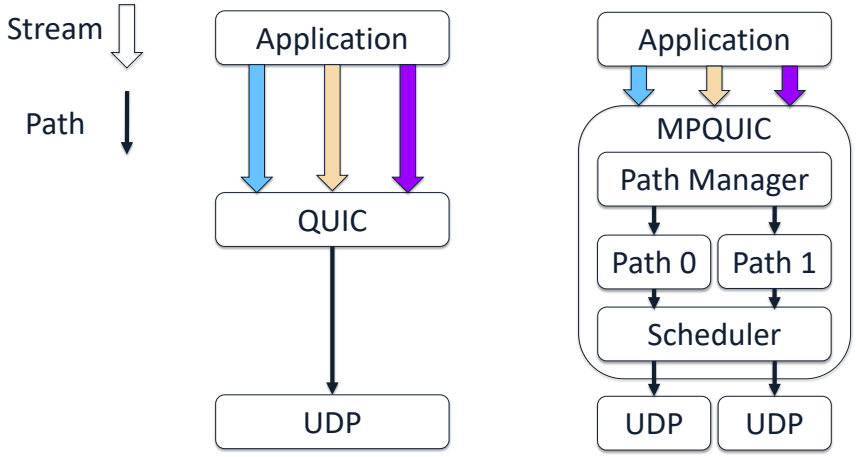


Figure 1.1: QUIC and MPQUIC.

In the two-path scheduler design, there are two primary challenges: 1) handling heterogeneous paths and 2) ensuring effectiveness in a dynamic environment. Although QUIC's multiplexing capability mitigates the impact of packet reordering across streams, QUIC still expects data within an individual stream to be delivered to the application layer in order. The heterogeneous nature of paths can lead to out-of-order delivery, causing segments that arrive first with higher stream offset to wait in the receiving queue until the missing segments are delivered. This phenomenon is known as head-of-line (HoL) blocking. Moreover, heterogeneity in terms of bandwidth could lead to underutilization of available resources. Multipath transmission under dynamic environments is another significant challenge. Schedulers must determine the appropriate distribution of packets across different paths. However, fluctuations in latency and bandwidth can complicate scheduling decision-making due to inaccurate metrics estimates or an overreliance on specific characteristics such as RTT, which can ultimately reduce bandwidth utilization and increase transmission completion time. Optimizing the use of available bandwidth, especially in high dynamic environments, is a key area yet to be explored.

1.1 Main Contributions

In this thesis, we begin by analyzing various two-path scheduling algorithms and identifying their limitations. Following this, we develop a two-path transmission model and conduct a comprehensive study of bandit convex optimization algorithms,

demonstrating their effectiveness in addressing two-path scheduling problems. Based on our system model and insights from bandit convex optimization, we propose the FTRL-WRR algorithm[1], which consists of three main components. The first is a learning-based optimization algorithm that explores the probability distribution for traffic assignment across two paths, converging on a distribution that optimizes bandwidth utilization. The second component is a distribution change detector designed to adjust the learning rate and exploration radius of the learning module. The third component is a weighted round-robin (WRR) scheduler that makes scheduling decisions based on the output of the learning module. Our key contributions are summarized as follows:

- We conduct a detailed analysis of different scheduling algorithms, implement them on Picoquic [11], identify their weaknesses, and evaluate them using both emulation and a real testbed.
- We find that the relationship between throughput and traffic distribution can be modeled as a quasi-concave function. Thus, we formulate traffic allocation across two paths as a 1-dimensional optimization problem.
- We provide an in-depth analysis of bandit convex optimization algorithms, with a focus on gradient-descent-based approaches such as Bandit Gradient Descent (BGD) and FTRL, and demonstrate the effectiveness of FTRL in our optimization problem.
- We propose the FTRL-WRR algorithm, which combines a lightweight FTRL learner with a WRR scheduler, utilizing an ADWIN2 [4] module to detect environmental changes and reset the learning rate and exploration radius accordingly.
- We build a trace-driven emulator based on Mininet [14] and `tc-netem` [17] that emulates Starlink delay, bandwidth, and handover behavior with precision.
- We evaluate the performance of the FTRL-WRR algorithm against other scheduling algorithms in various emulated environments and a real testbed, demonstrating its effectiveness.

[1] `https://github.com/Peter-LiDP/FTRL-WRR`

1.2 Thesis Overview

Chapter 1 introduces the two-path scheduler, discusses the current problems, and outlines the contributions of this thesis.

Chapter 2 provides a detailed overview of existing scheduling algorithms, presents the system model, and analyzes bandit convex optimization algorithms.

Chapter 3 presents the proposed algorithm in detail, explains its effectiveness, and analyzes its complexity.

Chapter 4 evaluates the FTRL-WRR algorithm through emulation and real testbed experiments, compares its performance with other scheduling algorithms, and studies the impact of congestion control.

Chapter 5 concludes the thesis and discusses future work.

Chapter 2

Background

In this chapter, we begin with an introduction to QUIC. Next, we introduce its multipath extension and demonstrate various multipath scheduling algorithms along with their limitations. After that, we present the system model and discuss the feasibility of modeling it as a 1-dimensional optimization problem. Finally, we provide a detailed discussion on bandit convex optimization and offer an analysis.

2.1 QUIC

QUIC is a UDP-based transport layer protocol developed to improve web performance. Prior to QUIC, the widely used transport layer protocols were UDP and Transmission Control Protocol (TCP). UDP is a lightweight, connectionless protocol that transmits data without establishing a connection beforehand. While it offers basic error detection, it does not guarantee packet delivery or ordering, making it suitable for low-latency applications that can tolerate packet loss and reordering. TCP, on the other hand, is a more complex, connection-oriented protocol that ensures reliable data delivery through flow control, congestion control, sequence numbering, and acknowledgment mechanisms. However, TCP introduces higher latency due to its three-way handshake connection setup and is limited to a single stream per connection, which can lead to performance issues such as HoL blocking. Furthermore, TCP does not provide built-in encryption, creating security vulnerabilities. QUIC addresses these limitations by incorporating beneficial features from both UDP and TCP. Similar to TCP, QUIC initiates a connection with a handshake, as illustrated in Fig 2.1 and 2.2.

```
Client                                                    Server

Initial[0]: CRYPTO[CH] ->

                                     Initial[0]: CRYPTO[SH] ACK[0]
                           Handshake[0]: CRYPTO[EE, CERT, CV, FIN]
                                     <- 1-RTT[0]: STREAM[1, "..."]

Initial[1]: ACK[0]
Handshake[0]: CRYPTO[FIN], ACK[0]
1-RTT[0]: STREAM[0, "..."], ACK[0] ->

                                                 Handshake[1]: ACK[0]
              <- 1-RTT[1]: HANDSHAKE_DONE, STREAM[3, "..."], ACK[0]
```

Figure 2.1: 1-RTT Handshake [12]

In its initial connection with an unknown peer, QUIC requires a 1-RTT handshake to exchange cryptographic information. For subsequent connections with the same peer, a 0-RTT handshake can be used, leveraging cached credentials to significantly reduce latency and allow immediate application data transmission [12] following ClientHello (CH). By default, QUIC integrates TLS 1.3 [26], which combines the transport and TLS handshakes into a single process, further reducing latency and enhancing security.

Unlike TCP, QUIC supports multiplexing, enabling multiple streams within a single connection. This feature mitigates HoL blocking, as packet loss affects only the streams involved, while other streams continue to deliver data to the application layer. QUIC's streams are lightweight abstractions, well-suited for efficiently handling discrete messages [13]. Application data is encapsulated in stream frames, and each packet may carry frames from multiple streams, with each frame including a stream ID to indicate its respective stream.

To ensure reliable transmission, QUIC adopts several key features from TCP, including acknowledgment and retransmission mechanisms, flow control, and congestion control. These features are modified where necessary to support QUIC's multiplexing capabilities.

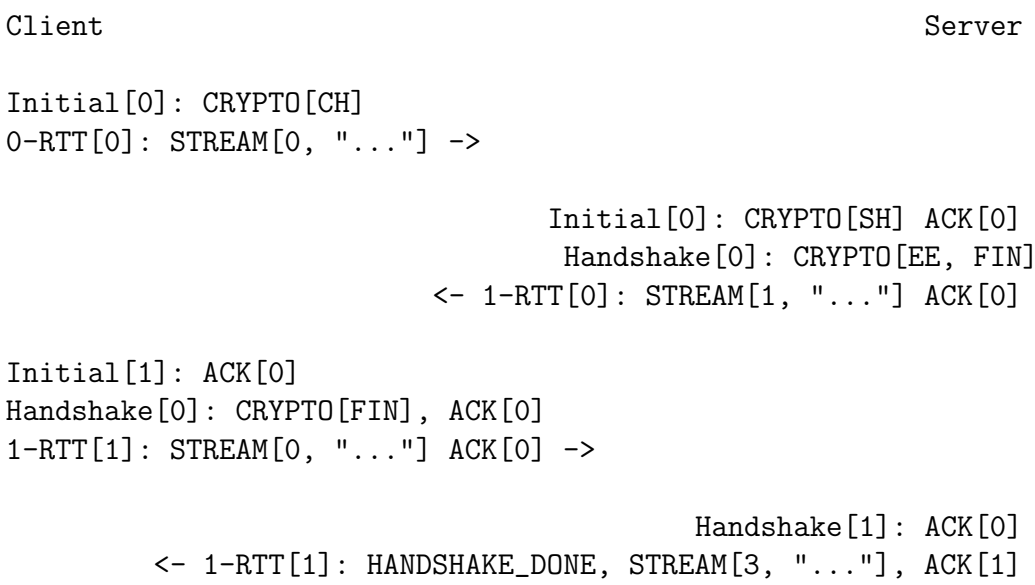


Figure 2.2: 0-RTT Handshake [12]

2.2 Multipath QUIC and Schedulers

Multipath transmission is rooted in the resource pooling principle, which aims to make a collection of resources behave as a single pool [27]. The primary motivations for MPQUIC include: (1) implementing the resource pooling principle and (2) enhancing resilience to connectivity failures [7].

A significant body of research on MPQUIC focuses on designing efficient schedulers that allocate traffic based on current network conditions, minimize flow completion time, and address issues like HoL blocking. This subsection provides an overview of the most widely studied MPQUIC schedulers, with illustrations of how they function in a two-path transmission scenario.

2.2.1 MinRTT

MinRTT serves as the default scheduler for multipath TCP (MPTCP). It prioritizes sending data over the path with the lowest RTT, provided that its CWND has available space. If the CWND is full, the scheduler switches to the path with the next lowest RTT. Let P_f represent the path with the lowest RTT and P_s represent the slower path. The pseudocode for this strategy is shown in Algorithm 1.

This algorithm is both simple and efficient but comes with limitations. In environments with heterogeneous paths, MinRTT may result in significant out-of-order

Algorithm 1 MinRTT

Input: P_f, P_s
if P_f is not congested **then**
  return P_f
else
  return P_s
end if

delivery, as it does not account for the actual delivery time of each packet. Moreover, it can lead to suboptimal bandwidth utilization, especially in scenarios where paths differ greatly in bandwidth, due to its preference for the path with the lower RTT.

2.2.2 BLEST

Similar to MinRTT, Blocking Estimation-based scheduler (BLEST) [9] also prefers the lower RTT path. If that path is not congested, packets can be safely transmitted on it. However, if the lower RTT path is blocked, instead of directly transmitting on the slower path, BLEST first estimates if sending on the slower path will result in out-of-order delivery. If this is the case, it chooses to wait for the faster path to become available. The pseudocode for this strategy is presented in Algorithm 2.

Algorithm 2 BLEST

Input: P_f, P_s, RTT_f, RTT_s, $\texttt{CWND}_f$, $\texttt{SWND}$, $inflight_s$, λ, MSS_f, MSS_s
if P_f is not congested **then**
  return P_f
else
  $rtts = \frac{RTT_s}{RTT_f}$
  $X = MSS_f \cdot \left(\texttt{CWND}_f + \frac{rtts-1}{2}\right) \cdot rtts$
  if $\lambda \cdot X > \texttt{SWND} - MSS_s \cdot (inflight_s + 1)$ **then**
    return wait
  else
    return P_s
  end if
end if

This algorithm assumes that transmitting a segment on the slower path P_s will take RTT_s. Consequently, in-flight segments on P_s, denoted as $inflight_s$, will also require RTT_s to be delivered. When congestion occurs on the faster path, the algorithm estimates the amount of data X that could be transmitted over the faster

path P_f during RTT_s and assesses whether this would exceed the remaining send window (SWND), taking into account the in-flight segments on P_s along with the corresponding Maximum Segment Size (MSS). If it does, indicating the risk of HoL blocking, the algorithm waits until the faster path becomes available.

The author also notes that the estimation of X could be inaccurate. Therefore, they introduced a correction factor λ. λ is tuned during transmission: HoL blocking during RTT_f will cause λ to increase by δ_λ, while the absence of HoL blocking will cause λ to decrease by δ_λ. λ is initialized to 1.0 at the beginning.

However, this design does not account for network dynamics. If network conditions change rapidly, the estimate of X based on the current network state could be significantly inaccurate. This inaccuracy could lead to unnecessary waiting and result in higher flow completion time.

2.2.3 ECF

Earliest Completion First (ECF) scheduler [16] is also a MinRTT-based algorithm. Similar to BLEST, when the fastest path is congested, it may choose to wait for the fastest path to become available instead of transmitting on the slower path immediately. However, instead of estimating potential HoL blocking, ECF bases its decision on estimating the flow completion time. Its pseudocode is presented in Algorithm 3.

Assume there are k packets in the send buffer of all streams. When the fastest path is congested, the time taken to wait until it's available and transmit these k packets over the fastest path can be denoted as $RTT_f + \frac{k}{\texttt{CWND}_f} \cdot RTT_f$. The time taken to transmit k packets over the slower path immediately will be at least RTT_s. If $RTT_f + \frac{k}{\texttt{CWND}_f} \cdot RTT_f < RTT_s$, we can conclude that waiting for the fastest path to become available and then transmitting can result in a lower flow completion time. Otherwise, transmitting on the slower path would be more efficient.

To avoid frequent state changes, the algorithm introduces a hysteresis value β, set to 0.25 by default. It also accounts for the variability of RTT on both the fast and slow paths, denoted as σ_f and σ_s, respectively, and chooses the larger one to represent the margin δ. Therefore, the algorithm checks if $RTT_f + \frac{k}{\texttt{CWND}_f} \cdot RTT_f < (1+\beta) \cdot (RTT_s + \delta)$. If this condition holds, the algorithm further evaluates if it is truly faster to wait for the fast path. The time taken to finish the transmission of k packets over the slower path is $\frac{k}{\texttt{CWND}_s} \cdot RTT_s$, and the time taken for the fast path to transfer will be at least $2 \cdot RTT_f$. If $\frac{k}{\texttt{CWND}_s} \cdot RTT_s \geq 2 \cdot RTT_f + \delta$, the algorithm will choose to wait for the fast

Algorithm 3 ECF

Input: P_f, P_s, RTT_f, RTT_s, $\texttt{CWND}_f$, $\texttt{CWND}_s$, σ_f, σ_s, β, k
if P_f is not congested **then**
  return P_f
else
  $\delta = \max(\sigma_f, \sigma_s)$
  if $RTT_f + \frac{k}{\texttt{CWND}_f} \cdot RTT_f < (1 + \text{waiting} \cdot \beta) \cdot (RTT_s + \delta)$ **then**
    if $\frac{k}{\texttt{CWND}_s} \cdot RTT_s \geq 2 \cdot RTT_f + \delta$ **then**
      waiting = 1
      return wait
    else
      return P_s
    end if
  else
    waiting = 0
    return P_s
  end if
end if

path. Otherwise, it will send on the slow path immediately.

Even though this algorithm accounts for RTT variation between the two paths, it may still be insufficient for high dynamic environments where bandwidth and delay vary over time. Therefore, similar to BLEST, ECF may also result in unstable performance in such scenarios.

2.2.4 Peekaboo

To address dynamic environments, Wu et al. introduced Peekaboo, a learning-based MinRTT algorithm designed to determine whether to transfer data via a slow path or wait for a faster path to become available [28]. Peekaboo divides the transmission process into a learning stage and a deployment stage, using a threshold to iterate between these stages. It employs a LinUCB-based approach combined with a stochastic adjustment parameter to decide between waiting or transmitting. The pseudocode for Peekaboo is presented in Algorithm 4.

Peekaboo initially enters the learning stage, where it employs a pure transmit strategy (always transmitting on the slower subflow when the fastest subflow is congested) and a pure waiting strategy (always waiting for the fastest subflow to become available) to gather data for updating the LinUCB model. Once the model is up-

Algorithm 4 Peekaboo

Input:
    1) BD_0, BD_1: dominant boundaries of the deterministic strategy, i.e., ($p_{wt,tx} = 0, 1$);
    2) $\hat{q}_{th}$: threshold to detect the change of $\hat{q}$;
Learning:
    $h_{p_{wt}=0} =$ `Run&Record`(); // i.e., minRTT, probability to wait is 0
    $h_{p_{tx}=0} =$ `Run&Record`(); // i.e., only use the faster path and always wait for it, probability to transmit is 0
    $\Theta, \hat{q}_{wt}, \hat{q}_{tx} =$ `Run&Learn`$(h_{p_{wt}=0}, h_{p_{tx}=0})$;
    $p_{indif,wt}, p_{indif,tx} = Est(\Theta, h_{p_{wt}=0}, h_{p_{tx}=0})$;
if $\hat{q}_{wt} > BD_1$ **then**
    $p_{wt} = 0.9$;
else if $\hat{q}_{wt} < BD_0$ **then**
    $p_{wt} = 0.1$;
else
    $p_{wt} = p_{indif,wt}$;
end if
if $\hat{q}_{tx} > BD_1$ **then**
    $p_{tx} = 0.9$;
else if $\hat{q}_{tx} < BD_0$ **then**
    $p_{tx} = 0.1$;
else
    $p_{tx} = p_{indif,tx}$;
end if
goto **Deployment**;
Deployment:
while True **do**
    $\hat{q}_{deploy,wt}, \hat{q}_{deploy,tx} = Deploy\&Est(\Theta, p_{wt}, p_{tx})$;
    if $|\hat{q}_{deploy,wt} - \hat{q}_{wt}| > \hat{q}_{th}$ **then**
        goto **Learning**;
    end if
    if $|\hat{q}_{deploy,tx} - \hat{q}_{tx}| > \hat{q}_{th}$ **then**
        goto **Learning**;
    end if
end while

dated, Peekaboo derives Θ from the LinUCB model to estimate rewards based on observed features in the deployment stage. After the learning phase, it also derives $\hat{q}_{wt}$ and $\hat{q}_{tx}$ from historical learning rounds. Here, $\hat{q}$ represents the probability that the action selected by the deterministic policy yields a higher reward than the alternative action. Since it is impossible to execute both actions simultaneously, Peekaboo compares these actions using the closest feature vectors in terms of Mahalanobis distance. Then Peekaboo compares $\hat{q}_{wt}$ and $\hat{q}_{tx}$ with boundaries BD_0 and BD_1, which is set as 0.3 and 0.7, respectively, to determine the stochastic adjustment parameter that will be used in the deployment stage.

In the deployment stage, Peekaboo estimates the reward for both actions based on Θ and the current features, using the stochastic adjustment parameter to determine the probability of selecting the action with the higher reward. It continuously calculates $\hat{q}$, and if $|\hat{q}_{deploy,wt} - \hat{q}_{wt}|$ or $|\hat{q}_{deploy,tx} - \hat{q}_{tx}|$ exceeds the threshold $\hat{q}_{th}$, set at 0.15, Peekaboo returns to the learning stage.

While Peekaboo is designed for dynamic environments by adapting in real-time, our analysis identified several challenges in certain environments. First, Peekaboo requires a learning stage in which it alternates between the waiting and transmitting strategy. The waiting strategy essentially reduces Peekaboo to a single-path algorithm that relies solely on the fastest path, inevitably leading to performance losses. Second, Peekaboo is designed with a 4 MB learning overhead for each learning round. However, in a high bandwidth environment, sending 4 MB of data may be too quick, causing the learning stage to end too early before Peekaboo has adequately explored different actions. As a result, the algorithm may quickly enter the deployment stage but then return to the learning stage almost immediately to test another action. This repetitive cycling can distort the learning process: because both actions involve sending 4 MB of data, the reward is now determined by the amount of data received and acknowledged from these 4 MB of data, rather than the throughput, which was the original metric the algorithm intended to optimize. This confuses the learning model because the waiting strategy sometimes results in lower packet loss, particularly in environments where the fastest path has a lower packet loss rate. As a result, the waiting strategy may appear to yield a higher reward, even though it leads to suboptimal throughput. To adapt Peekaboo to different scenarios, the learning overhead needs to be carefully tuned. Furthermore, Peekaboo's authors confirmed that Peekaboo may struggle to adapt in high dynamic environments where network changes occur more rapidly than the learning process can keep up [28].

2.2.5 Weighted Round Robin

Picoquic, as one of the first QUIC libraries to include a multipath extension, also implements a built-in scheduling mechanism. Its scheduler is essentially a Round-Robin (RR) scheduler, which alternates between two paths [11]. However, the number of packets that can be sent in each round is determined by its built-in pacing mechanism, effectively making it a WRR algorithm.

Picoquic applies the leaky bucket algorithm for pacing. First, it calculates the expected pacing rate, given by

$$pacing\_rate = pacing\_gain \times BtlBW \tag{2.1}$$

with BtlBW representing the bottleneck bandwidth derived from the receiving rate. The pacing_gain cycles through the sequence {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.25, 0.75}, periodically probing for available bandwidth or draining the queue. Based on the pacing rate, Picoquic estimates the time required to transmit a packet. Since the actual packet size is not known in advance, it assumes the packet size is equal to the Maximum Transmission Unit (MTU) and uses

$$pacing\_packet\_time = \frac{MTU}{pacing\_rate} \tag{2.2}$$

to estimate the transmission time. If the estimated time is greater than the current smoothed RTT, the smoothed RTT is used instead.

In the leaky bucket algorithm, the bucket is refilled gradually based on the elapsed time, and its size is decreased when a packet is sent. The bucket size increment is based on the time elapsed since the last evaluation, which corresponds to the last round of packet transmissions. However, the bucket size cannot be smaller than $-pacing\_packet\_time$ or exceed the *quantum*. The *quantum* is initially set as $quantum = \frac{CWND}{4}$ and is constrained to be no larger than $16MTU$ and no smaller than $2MTU$.

When making a packet sending authorization decision, Picoquic compares the current bucket size to the pacing packet time. If the bucket size is not greater than the pacing packet time, it blocks the current sending. After a packet is sent, the bucket size is decreased by

$$pacing\_bucket -= \frac{pacing\_packet\_time \times length + (MTU - 1)}{MTU} \tag{2.3}$$

which estimates the actual packet transmission time based on the current pacing rate and the actual packet size.

The primary objective of this algorithm is to prevent sending packets faster than the path's capacity, thus avoiding congestion and ensuring smooth transmission. Its decision-making relies on the pacing rate, derived from bandwidth estimate, and adjusts the bucket size based on idle time. However, in high dynamic environments, this approach may be overly conservative and can lead to underutilization of available bandwidth as it heavily relies on the current bandwidth estimate.

2.3 Two-path Transmission Model

To comprehensively understand the two-path transmission behavior for designing a scheduler, we first conducted several experiments and tried to build a system model. Two-path transmission, as a specific case of multipath transmission, requires the proportion of traffic assigned to both paths to sum to 1. Let path 0 and path 1 be represented as P_0 and P_1, respectively.

Assume the entire transmission occurs over T timesteps. Let X_t represent the proportion of traffic assigned to P_0 at timestep t, where $X_t \in (0, 1)$. Accordingly, the proportion of traffic assigned to P_1 at timestep t is $1 - X_t$. Denote the total bandwidth of the two paths at timestep t as bw_t. The bandwidth ratios of P_0 and P_1 are represented by b_t and $1 - b_t$, respectively. Assume that at the current timestep, the variable X_t is equal to actual bandwidth ratio b_t.

Derived from the resource pooling principle, the maximum achievable throughput is the total bandwidth available across both paths. Consider an ideal environment without any packet loss or delay, where the maximum achievable throughput for P_0 and P_1 are represented as tp_{p0} and tp_{p1}, respectively. The maximum achievable total throughput during timestep t, denoted as tp_t, can be expressed as:

$$tp_t = tp_{p0} + tp_{p1} = bw_t. \tag{2.4}$$

Assume a scenario where a learner determines the traffic distribution X_t and $1-X_t$ rather than having it perfectly align with the respective bandwidth ratios. The maximum achievable throughput in this context is denoted as $\hat{tp_t}$. When $X_t \geq b_t$, path P_0 is either fully utilized or overloaded. Since path P_0 is unable to handle more traffic than its bandwidth allows, attempting to distribute traffic according to X_t will

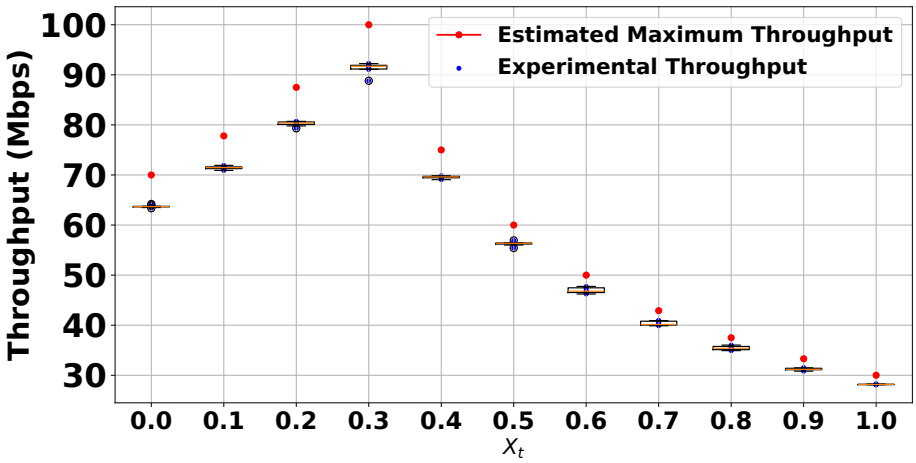


Figure 2.3: Throughput with Different X_t.

inevitably cause underutilization of the bandwidth on path P_1. As a result, the total throughput is constrained by path P_0. This leads to:

$$\hat{tp_t} = \frac{bw_t \cdot b_t}{X_t}. \tag{2.5}$$

On the other hand, when $X_t < b_t$, path P_1 becomes overloaded, which then constrains the total throughput. In this case, the total throughput is determined by path P_1, leading to:

$$\hat{tp_t} = \frac{bw_t \cdot (1 - b_t)}{1 - X_t}. \tag{2.6}$$

Based on our assumption, under varying traffic distributions, the maximum achievable throughput will differ from the nominal throughput. Considering the relationship between X_t and b_t, the maximum achievable throughput can be expressed as:

$$\hat{tp_t} = \begin{cases} \dfrac{bw_t \cdot b_t}{X_t} & \text{if } b_t \leq X_t < 1 \\ \dfrac{bw_t \cdot (1 - b_t)}{1 - X_t} & \text{if } 0 < X_t < b_t \end{cases}. \tag{2.7}$$

We verified this assumption via Mininet emulation, assuming the bandwidths of the two paths are 30 Mbps and 70 Mbps, respectively. The propagation delay and packet loss rate were both set to 0. We experimented with various traffic distributions by modifying the value of X_t, the proportion of traffic assigned to P_0 during transmission. Each configuration was tested over ten runs, and the throughput results, calculated based on the delivered packets, are shown in Figure 2.3.

The red points represent the estimated maximum throughput at different traffic distributions according to Equation 2.7. The results indicate that changes in throughput under different distributions largely align with our assumptions. However, none of the experimental results achieve the estimated maximum throughput. This limitation arises from the complexity of real-world network behaviors. Even without propagation delay and intentional packet loss, other factors such as processing delay, congestion control, and packet loss due to buffer overflow still influence the actual observed throughput. We consider these deviations in observed throughput as noise, denoted by ϵ_t, where $\epsilon_t \leq 0$.

The assumptions above rely on traffic being allocated exactly according to X_t and $1-X_t$. However, in practical implementations, schedulers often redirect traffic to fully utilize the congestion window across all available paths [5]. This redirection can help mitigate the impact of deviations between X_t and the actual bandwidth ratio b_t, as the traffic distribution is influenced not only by the scheduler but also by congestion control algorithms (CCAs). Additionally, packet loss can further affect the traffic allocation. We define these deviations as β_t, representing the real-world network conditions that prevent traffic from being allocated as expected. Thus, the actual traffic distribution across the two paths can be expressed as $X_t + \beta_t$ and $1 - X_t - \beta_t$, where $\beta_t \in (-1, 1)$.

Consequently, we can define the relationship between the assigned distribution X_t and observed throughput with the function $T(X_t)$, represented as follows:

$$T(X_t) = \begin{cases} \dfrac{bw_t \cdot b_t}{X_t + \beta_t} + \epsilon_t & \text{if } b_t - \beta_t \leq X_t < 1 \\ \dfrac{bw_t \cdot (1 - b_t)}{1 - X_t - \beta_t} + \epsilon_t & \text{if } 0 < X_t < b_t - \beta_t \end{cases}. \tag{2.8}$$

When $b_t - \beta_t \leq 0$, $T(X_t)$ monotonically decreases over the interval $(0, 1)$, reaching its supremum as X_t approaches 0 from the right. When $b_t - \beta_t \geq 1$, $T(X_t)$ monotonically increases over the interval $(0, 1)$, reaching its supremum as X_t approaches 1 from the left. When $b_t - \beta_t \in (0, 1)$, $T(X_t)$ monotonically increases over the interval $(0, b_t - \beta_t)$ and decreases over the interval $(b_t - \beta_t, 1)$, with a global peak at $X_t = b_t - \beta_t$. Additionally, $T(X_t)$ is continuous at $X_t = b_t - \beta_t$. Therefore, $T(X_t)$ is a univariate unimodal function and always has a global supremum or peak, making it quasi-concave. Since the negative of a quasi-concave function is quasi-convex, we can construct a quasi-convex loss function based on $T(X_t)$, as discussed in the next chapter. The quasi-convex nature of the loss function guarantees a unique global min-

imum. However, during actual transmission, the exact throughput function remains hidden. At each timestep, we select an action X_t and observe the corresponding throughput. This scenario presents an optimization problem with bandit feedback. Rather than employing a first-order optimization method that requires direct access to the gradient, we can utilize zeroth-order optimization, which estimates the gradient based on the observed loss values. Moreover, a surrogate function is constructed to provide smoothness and guide the optimization process toward the optimal point.

2.4 Bandit Convex Optimization

Bandit problems are a class of sequential decision-making problems where a learner selects from a set of actions and receives an outcome based on the chosen action without full information about the outcomes of other possible actions. Bandit problems are also closely related to optimization problems. In bandit convex optimization, the set of actions forms a convex subset of Euclidean space, and the function mapping actions to losses is assumed to be convex. The learner chooses actions and observes the corresponding loss, possibly with additive noise [15]. Bandit convex optimization is closely related to zeroth-order optimization, which is the gradient-free counterpart of first-order optimization methods [18]. In zeroth-order optimization, the actual gradient is hard or impossible to compute, so function values are used to approximate actual gradients.

2.4.1 Bandit Gradient Descent

BGD is a zeroth-order variant of gradient descent designed for convex optimization in adversarial environments where explicit gradient information is inaccessible. Unlike traditional gradient descent, which relies on direct access to gradients, BGD operates under bandit feedback. This implies that the algorithm observes only the loss at a randomly sampled point, using that observation to estimate the gradient. An abstract version of BGD is shown in Algorithm 5 [15].

Algorithm 5 Bandit Gradient Descent

args: learning rate $\eta > 0$
initialize $x_1 \in K$
for $t = 1$ to n **do**
    sample X_t from some distribution based on x_t
    observe $Y_t = f_t(X_t)$
    compute gradient estimate g_t using x_t, X_t, and Y_t
    update $x_{t+1} = \Pi_K(x_t - \eta g_t)$
end for

The first step involves sampling and selecting a point X_t based on x_t for exploration. The loss is evaluated at X_t, but the information is used to estimate the gradient at x_t. In a typical d-dimensional BGD, this process involves defining a surface $\mathbb{S}_r^{d-1} = \{x \in \mathbb{R}^d : \|x\| = r\}$, which is the surface of a sphere $\mathbb{B}_r^d = \{x \in \mathbb{R}^d : \|x\| \leq r\}$ with radius r centered at x_t, and then randomly sampling a point $X_t = x_t + U$, where U is a random vector uniformly distributed on $\mathbb{S}_r^{d-1}$. After playing X_t, the algorithm observes the corresponding loss Y_t, generated by the underlying function f_t.

The next step is to estimate the gradient g_t using the sampled point. It is important to note that g_t may not necessarily be an estimate of the true gradient $f_t'(x_t)$, which might not exist, particularly if the loss function is non-differentiable at certain points. Instead, g_t approximates the gradient of a surrogate loss function s_t, which is a smoothed version of $f_t(x_t)$. To define this, let $s(x)$ be the convolution between $f_t(x_t)$ and a uniform distribution over the sphere $\mathbb{B}_r^d$. Using the mean value theorem for integrals, $s(x)$ is expressed as:

$$s(x) = \frac{1}{\text{vol}(\mathbb{B}_r^d)} \int_{\mathbb{B}_r^d} f(x+u)\, du, \tag{2.9}$$

where u is a vector inside $\mathbb{B}_r^d$. Similarly, applying the mean value theorem for integrals and Stokes' theorem, the gradient $s'(x)$ can be expressed as:

$$s'(x) = \frac{1}{\text{vol}(\mathbb{B}_r^d)} \int_{\mathbb{B}_r^d} f'(x+u)\, du \tag{2.10}$$

$$= \frac{d}{r} \left[\frac{1}{\text{vol}(\mathbb{S}_r^{d-1})} \int_{\mathbb{S}_r^{d-1}} f(x+u) \frac{u}{r}\, du \right] \tag{2.11}$$

$$= \frac{df(X_t)U}{r^2} \tag{2.12}$$

$$= \frac{dY_tU}{r^2}. \tag{2.13}$$

Thus, based on the observed loss Y_t at X_t, the gradient estimate can be expressed as $g_t = \frac{dY_tU}{r^2}$ [15].

After computing the gradient estimate, x_t is updated in the direction opposite to the estimated gradient as in traditional gradient descent, but it needs to be projected back onto the feasible set if it exceeds the set.

2.4.2 Follow The Regularized Leader

Despite the efficiency and simplicity demonstrated by BGD in solving convex optimization problems within a bandit setting, the method also presents some limitations. One key assumption underlying BGD is that the loss function is Lipschitz continuous [15]. This implies that the change in loss values is bounded by a constant L, as defined by:

$$|f(x_1) - f(x_2)| \leq L|x_1 - x_2|. \tag{2.14}$$

However, when applying this method to two-path QUIC scheduler design, where throughput is used to design the loss function, this assumption becomes difficult to uphold. Throughput is sometimes subject to significant fluctuations due to factors such as CCAs and packet loss rate, as discussed in Section 2.3. These variations can cause abrupt changes in the loss function, violating the Lipschitz condition. Additionally, BGD faces challenges due to boundary effects [15]. Since the selection of x_t must remain within the feasible set K, and because exploration is required, x_t is projected onto the subset $J = \{x \in K : x + \mathbb{B}_r^d \subseteq K\}$. This projection restricts the algorithm to a region away from the boundary of K, which can limit the algorithm's ability to fully utilize the potential smoothness of the loss function, particularly near the boundary of the feasible set.

To address these challenges, Lattimore proposed a variant of FTRL with a regularization term $R(x)$, which acts as a self-concordant barrier to restrict exploration within the feasible set. The general structure of FTRL is shown in Algorithm 6 [15].

Algorithm 6 Follow The Regularized Leader

args: learning rate $\eta > 0$
initialise $x_1 = \arg\min_{x \in \text{int}(K)} R(x)$
for $t = 1$ to n **do**
    compute $x_t = \arg\min_{x \in \text{int}(K)} \left[R(x) + \sum_{u=1}^{t-1} \eta \langle g_u, x \rangle \right]$
    sample X_t based on x_t and observe Y_t
    compute gradient estimate g_t using x_t, X_t, and Y_t
end for

Similar to BGD, this variant of FTRL utilizes a surrogate loss function $s(x)$, which is defined as:

$$s(x) = \frac{1}{\text{vol}(E)} \int_E \left(2f\left(\frac{1}{2}z + \frac{1}{2}x \right) - f(z) \right) dx, \tag{2.15}$$

where $E = \{y \in \mathbb{R}^d : |x - y|_{\Sigma^{-1}} \leq 1\}$ represents an ellipsoidal neighborhood around x, and Σ is a positive definite matrix that defines the shape of the ellipsoid. In the variant of FTRL we used, Σ is set to $\Sigma = r^2 \left[R''(x) \right]^{-1}$. After simplifying, as in the process used in BGD, the gradient of $s(x)$ can be expressed as:

$$s'(x) = 4d\Sigma^{-1}\mathbb{E}[f(X)(X - x)]. \tag{2.16}$$

An unbiased estimator of $s'(x)$ is:

$$g = 4d\Sigma^{-1}Y(X - x). \tag{2.17}$$

By applying $\Sigma = r^2 \left[R''(x) \right]^{-1}$, it can be further expressed as:

$$g_t = \frac{4dY_t R''(x_t)(X_t - x_t)}{r^2}. \tag{2.18}$$

In addition, instead of updating x_t based solely on the current gradient, FTRL uses the cumulative gradient from the start. The update equation

$$x_t = \arg \min_{x \in \text{int}(K)} \left[R(x) + \sum_{u=1}^{t-1} \eta \langle g_u, x \rangle \right] \tag{2.19}$$

can be seen as a variant of the gradient descent formula $x_t = \Pi_K(x_{t-1} - \eta g_{t-1})$. In this variant of FTRL, since $R(x)$ is defined as a self-concordant barrier that naturally keeps x_t within the feasible set, projection onto K is no longer necessary.

A self-concordant barrier $R(x)$ is defined as:

$$(a) \quad D^3R(x)[h,h,h] \leq 2\left(D^2R(x)[h,h]\right)^{3/2} \quad \text{for all } x \in \text{int}(K) \text{ and } h \in \mathbb{R}^d; \quad (2.20)$$

$$(b) \quad R \text{ is a barrier}: \quad R(x_t) \to \infty \text{ whenever } x_t \to \partial K. \quad (2.21)$$

In our work, the feasible set is the interval $(0,1)$. We define $R(x) = -\ln(x - x^2)$.
Lemma. $R(x) = -\ln(x - x^2)$ is a self-concordant barrier within the domain $(0,1)$.

Proof. Consider the function $R(x) = -\ln(x - x^2)$ for $x \in (0,1)$. Since this is a 1-dimensional space, h becomes a scalar, condition (a) can be simplified to

$$|R'''(x)| \leq 2\left(R''(x)\right)^{\frac{3}{2}} \quad \text{for all } x \in (0,1)\ .$$

Define $f_1(x) = -\ln x$ for $x \in (0, +\infty)$:

$$2 \cdot \left(f_1''(x)\right)^{\frac{3}{2}} = \frac{2}{x^3}, \quad |f_1'''(x)| = \frac{2}{x^3}.$$

$|f_1'''(x)| = 2\left(f_1''(x)\right)^{\frac{3}{2}}$ for all $x \in (0, +\infty)$.

Define $f_2(x) = -\ln(1-x)$ for $x \in (-\infty, 1)$:

$$2 \cdot \left(f_2''(x)\right)^{\frac{3}{2}} = \frac{2}{(1-x)^3}, \quad |f_2'''(x)| = \frac{2}{(1-x)^3}.$$

$|f_2'''(x)| = 2\left(f_2''(x)\right)^{\frac{3}{2}}$ for all $x \in (-\infty, 1)$.

Since $R(x) = f_1(x) + f_2(x)$ for $x \in (0,1)$, and both functions satisfy condition (a), $R(x)$ also satisfies condition (a) [22].

Next, for condition (b), we note that:

$$\lim_{x \to 0^+} R(x) = +\infty \quad \text{and} \quad \lim_{x \to 1^-} R(x) = +\infty.$$

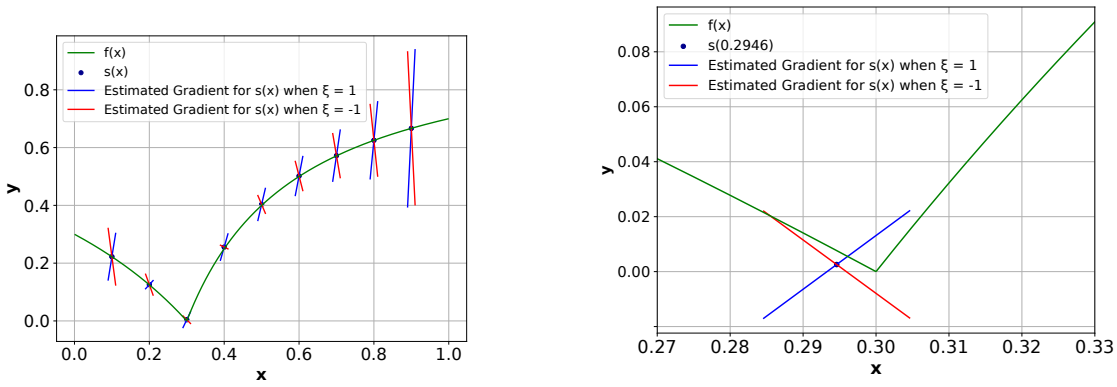
Thus, as $x_t \to \partial K$, $R(x_t) \to \infty$, satisfying condition (b). Therefore, $R(x) = -\ln(x - x^2)$ is a self-concordant barrier on the domain $(0,1)$. □

To demonstrate the effectiveness of FTRL in optimizing the defined quasi-convex

loss function, we present a simplified example of this process. Referring to the function 2.7 defined in the system model, assume we have a loss function $f(x)$ expressed as $f(x) = 1 - \frac{t\hat{p}_t}{bw_t}$. Using the same example with two paths having bandwidths of 30 Mbps and 70 Mbps, $f(x)$ can be written as:

$$f(x) = \begin{cases} 1 - \frac{0.3}{x} & \text{if } 0.3 \leq x < 1 \\ 1 - \frac{0.7}{1-x} & \text{if } 0 < x < 0.3 \end{cases}.$$

This is illustrated in Figure 2.4a.



(a) s(x) and f(x) when r = 0.5

(b) s(x) and f(x) when r = 0.1

Figure 2.4: s(x) and f(x).

The function is quasi-convex but exhibits some concavity. To demonstrate the optimization of the loss function $f(x)$, we take 9 sample points from 0.1 to 0.9 to illustrate the surrogate function $s(x)$ and its gradient at these points. In practice, the points are chosen using the formula 2.19. Assume each selected point x represents an iteration. In the actual algorithm and implementation, only a single X_t is sampled in each iteration. However, for demonstration purposes, assume that in each iteration, two points, X_{-1} and X_1, are sampled around x within an ellipsoidal region E, following

$$X_\xi = x + \frac{r}{2} \left(R''(x)\right)^{-\frac{1}{2}} \xi, \tag{2.22}$$

where ξ is -1 or 1, respectively. Each time an X_ξ is sampled, we calculate an estimated gradient g_ξ. With different selections of ξ, we observe two distinct gradients, g_{-1} and g_1, both unbiased estimates of the actual gradient of $s(x)$.

As shown in Figure 2.4, in the region $(0, 0.3)$, the estimated gradient in the negative direction is consistently larger than in the positive direction, driving x to optimize

towards the positive direction. Conversely, in the region $[0.3, 1)$, the estimated gradient in the negative direction is smaller than in the positive direction, causing x to optimize towards the negative direction. Thus, the sum of the estimated gradients always provides a meaningful direction towards minimizing $f(x)$. Convergence of $s(x)$ occurs when the gradients in both directions are equal in magnitude but opposite in sign, canceling each other out. In this example, when $r = 0.5$, convergence occurs around $x = 0.2744$, slightly deviating from the actual minimum of $f(x)$, which is at $x = 0.3$. However, as r decreases over time, the deviation decreases. As shown in Figure 2.4b, when $r = 0.1$, the convergence point moves closer to the minimum, around $x = 0.2946$.

Chapter 3

FTRL-WRR

In this chapter, we present our FTRL-WRR algorithm in detail, consisting of three major components. The first is the core of the FTRL algorithm, focusing on its loss function design and interaction with the network environment. We provide justification for the loss design and demonstrate FTRL's robustness during transmission. Next, we integrated an ADWIN2-based distribution change detector [4], which adjusts the learning rate and exploration radius to enhance FTRL's performance in high dynamic environments while minimizing exploration overhead. Finally, we describe the WRR-based scheduler, which handles scheduling decisions by distributing packets according to the weights generated by the FTRL module. We conclude with a complexity analysis of the proposed algorithms.

3.1 FTRL-based Learner

In our setting, the entire transmission is divided into T timesteps. The first timestep is fixed at 200 ms to account for initial probing and CWND initialization. The duration of subsequent timesteps is determined by the acknowledgment time of the last packet sent on each path during the previous timestep. Consequently, the interval for each timestep t lies within the range $[RTT_l, RTT_h + \sigma]$, where σ accounts for the additional delay caused by QUIC's delayed feedback. Here, RTT_l and RTT_h represent the RTT of the faster and slower paths, respectively. Figure 3.1 illustrates this process. In the figure, Path0_Packet(t) and Path1_Packet(t) refer to the last packets sent at timestep t from path 0 and path 1, respectively. An update is triggered only after both of these packets are acknowledged. If a specific packet is lost, the update is triggered when

Algorithm 7 FTRL with Ellipsoidal Smoothing

args: learning rate η_t, smoothing parameter r_t
initialize $x_1 = 0.5$, $R(x) = -\ln(x - x^2)$
for $t = 1$ to n **do**
  compute $\eta_t = 2^{-\frac{1}{2}}(t+1)^{-\frac{3}{4}}\ln(t+1)^{\frac{3}{4}}$
  compute $r_t = \min\left(1, 2^{\frac{1}{2}}(t+1)^{-\frac{1}{4}}\ln(t+1)^{\frac{1}{4}}\right)$
  compute $a = \sum_{u=1}^{t-1}(\eta \cdot g_u)$ if $t > 1$, else $a = 0$
  update $x_t = \begin{cases} 0.5, & \text{if } a = 0 \\ \frac{a+2-\sqrt{a^2+4}}{2a}, & \text{otherwise} \end{cases}$
  sample ξ_t uniformly from $\{-1, 1\}$
  play $X_t = x_t + \frac{r_t}{2}\left(R''(x_t)\right)^{-\frac{1}{2}}\xi_t$ and observe Y_t
  compute gradient estimate $g_t = \frac{4Y_t R''(x_t)(X_t - x_t)}{r_t^2}$
end for

subsequent packets are acknowledged, indicating the loss of the previous packet.

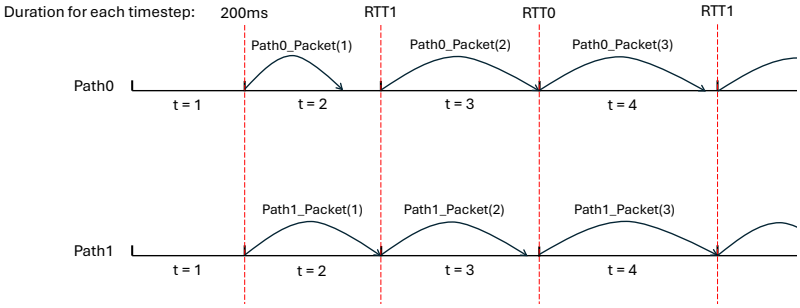


Figure 3.1: Timestep Updating Example.

Since we focus on macroscopic traffic distribution rather than per-packet scheduling, the time interval is sufficiently long to distribute traffic proportionally as expected. The weight X_t is generated by the FTRL algorithm, as shown in Algorithm 7.

Our optimization objectives include: 1) minimizing the difference between throughput and bandwidth and 2) minimizing bias β_t in environments where distribution is affected by congestion control and packet loss. We denote the observed throughput at the current timestep as tp_o and the bandwidth estimate as bw_e. To design a loss function that satisfies both objectives, we need to distinguish the proportion of data actually sent based on the expected distribution from observed throughput tp_o. To address this, we define a new metric called effective throughput, denoted as tp_{eff}, which can be calculated as follows:

$$tp_{\text{eff}} = \begin{cases} \dfrac{tp_o \cdot (X_t + \beta_t)}{X_t} & \text{if } -X_t \leq \beta_t \leq 0 \\ \dfrac{tp_o \cdot (1 - X_t - \beta_t)}{1 - X_t} & \text{if } 0 \leq \beta_t < 1 - X_t \end{cases}. \tag{3.1}$$

We define the loss:

$$Y_t = \left(1 - \frac{tp_{\text{eff}}}{bw_e}\right)^2. \tag{3.2}$$

This loss formulation helps the model optimize toward the expected optimal point, $x_t = b_t - \beta_t$. In the example provided in Section 2.4.2, the loss function may exhibit some concavity but can still be updated effectively. However, when the optimal point $x_t = b_t - \beta_t$ is very close to the boundary at 0 or 1, the concavity may become too severe, potentially confusing the optimization process and making it difficult to utilize the differences between sampled points for updates. While it is rare in practice for the optimal points to be too near to these boundaries, we must still ensure some level of robustness in such extreme cases. Adding a square term enhances the overall convexity of the loss function. Although increasing the power makes the loss function more convex, it also reduces the magnitude of differences between sampled points, thereby slowing the update process. Therefore, we add a square term as a tradeoff to handle boundary cases while maintaining convergence speed.

In the loss calculation, tp_o corresponds to $T(X_t)$, as defined in Equation 2.8. To demonstrate that the loss design is meaningful and will always guide us to the global minimum, we construct a model for the loss based on Equation 2.8 for analysis. The deviations ϵ_t and the square term affect only the loss value, not the location of the minimum. Therefore, for simplicity, we exclude ϵ_t and the square term from the analysis. Furthermore, in this analysis, we assume that the bandwidth estimate bw_e equals the actual bandwidth bw_t. While there may be inaccuracies in bw_e, they do not affect the location of the minimum. We will discuss these inaccuracies later. For analysis purposes, we construct a theoretical loss T_t without the square term:

$$T_t = \begin{cases} 1 - \dfrac{(1-b_t)(X_t+\beta_t)}{X_t(1-X_t-\beta_t)} & -X_t \le \beta_t \le 0, \quad 0 < X_t < b_t - \beta_t \\ 1 - \dfrac{b_t}{X_t} & -X_t \le \beta_t \le 0, \quad b_t - \beta_t \le X_t < 1 \\ 1 - \dfrac{1-b_t}{1-X_t} & 0 < \beta_t < 1 - X_t, \quad 0 < X_t < b_t - \beta_t \\ 1 - \dfrac{b_t(1-X_t-\beta_t)}{(X_t+\beta_t)(1-X_t)} & 0 < \beta_t < 1 - X_t, \quad b_t - \beta_t \le X_t < 1 \end{cases}. \tag{3.3}$$

We first analyze the scenarios where $-X_t \le \beta_t \le 0$. For $0 < X_t < b_t - \beta_t$, the derivative of T_t is given by:

$$T_t' = -\frac{(1-b_t)(X_t+\beta_t)}{X_t(-X_t-\beta_t+1)^2} - \frac{1-b_t}{X_t(-X_t-\beta_t+1)} + \frac{(1-b_t)(X_t+\beta_t)}{X_t^2(-X_t-\beta_t+1)}. \tag{3.4}$$

Upon examining this expression, it is evident that the derivative is consistently negative. Hence, T_t is monotonically decreasing over the interval $X_t \in (0, b_t - \beta_t)$.

For $b_t - \beta_t < X_t < 1$, the derivative is:

$$T_t' = \frac{b_t}{X_t^2}, \tag{3.5}$$

which is always positive, indicating that Y_t is monotonically increasing over $X_t \in (b_t - \beta_t, 1)$. Consequently, when $-X_t \le \beta_t \le 0$, the loss function is univariate unimodal, with a global minimum at $X_t = b_t - \beta_t$, making T_t quasi-convex.

Next, we consider the scenarios where $0 \le \beta_t \le 1 - X_t$. For $0 < X_t < b_t - \beta_t$, the derivative of T_t is:

$$T_t' = -\frac{1-b_t}{(1-X_t)^2}. \tag{3.6}$$

This expression is consistently negative, meaning T_t is monotonically decreasing for $X_t \in (0, b_t - \beta_t)$.

For $b_t - \beta_t < X_t < 1$, the derivative becomes:

$$T_t' = \frac{b_t}{(1-X_t)(X_t+\beta_t)} + \frac{b_t(-X_t-\beta_t+1)}{(1-X_t)(X_t+\beta_t)^2} - \frac{b_t(-X_t-\beta_t+1)}{(1-X_t)^2(X_t+\beta_t)}. \tag{3.7}$$

This expression can be shown to be consistently positive, implying that T_t is monotonically increasing for $X_t \in (b_t - \beta_t, 1)$. As a result, when $0 \leq \beta_t \leq 1 - X_t$, the loss function is univariate unimodal with a global minimum at $X_t = b_t - \beta_t$, and T_t is quasi-convex. Therefore, the loss Y_t drives X_t towards the minimum at $X_t = b_t - \beta_t$.

Additionally, if the traffic distribution is consistently influenced by congestion control, assume that whenever X_t is updated, the observed distribution $X_t + \beta_t$ remains fixed at a constant value, $X_t + \beta_t = c$. In this scenario, the loss function will drive X_t toward c to minimize β_t. However, when $X_t = c$, and c differs from the actual optimal point $b_t - \beta_t$, the loss will not reach zero but will retain a non-zero gradient, encouraging further updates toward the true optimal point.

In our setting, bw_e is derived from the highest instantaneous receiving rate recorded within timestep t, ensuring that $tp_o \leq bw_e$. Since $tp_{\text{eff}} \leq tp_o$, it follows that $tp_{\text{eff}} \leq bw_e$, which keeps the loss Y_t bounded within $[0, 1]$. During optimization, as X_t approaches the optimal point, throughput increases, resulting in a more accurate bandwidth estimate, bw_e. We define the bias between actual and estimated bandwidth as $\alpha = \frac{bw_t}{bw_e}$. Given that bw_e is derived from the receiving rate, which cannot exceed the physical bandwidth, we ensure that $\alpha \geq 1$. The upper bound of α decreases as X_t nears $b_t - \beta_t$. When $X_t = b_t - \beta_t$, α reaches its lowest upper bound, satisfying $1 \leq \alpha \leq 1 - \frac{\epsilon_t}{bw_t}$.

By incorporating the bandwidth estimation bias α into the loss function, the loss can also be expressed as:

$$Y_t = \left(1 - \alpha \cdot \frac{tp_{\text{eff}}}{bw_t}\right)^2. \tag{3.8}$$

While α scales the loss value, it does not affect the location of the minimum. Thus, inaccuracies in the bandwidth estimate are tolerable during optimization.

In Algorithm 1, η and r represent the learning rate and exploration radius, respectively, as outlined in Lattimore's work [15]. While η and r were originally set as fixed values, we adjust them to decrease over time since the number of iterations is not known in advance. Our analysis in Section 2.4.2 shows that reducing r brings the convergence point closer to the actual minimum. However, smaller values for η and r can slow down exploration over time, making it difficult for the algorithm to adapt to environmental changes in later stages. To address this, we designed and implemented a resetting mechanism that detects abrupt environmental changes and resets both η and r to their initial values (at $t = 1$) when necessary, as discussed in Section 3.2.

As discussed in Section 2.4.2, the algorithm optimizes the surrogate function $s(x)$ by observing the gradient at x_t. To construct an unbiased estimate of the gradient at x_t, we sample a point X_t around x_t within an ellipsoidal region. Throughout the optimization process, X_t serves as the actual action taken. Let a denote the cumulative gradient up to timestep $t-1$. To compute x_t, we solve $\nabla_x \left(ax - \ln(x - x^2)\right) = 0$ to find the minimum. If $a = 0$, we set $x_t = 0.5$. A point ξ_t is then sampled uniformly from $\{-1, 1\}$, with each value used for two consecutive timesteps. Finally, a single X_t is sampled for each iteration, and the observed loss is used to calculate the unbiased gradient estimate of $s(x)$ at x_t.

3.2 ADWIN2-based Distribution Change Detector

In the FTRL algorithm, because exploration never ceases, the actual action X_t played will always be close to the optimized point x_t. The distance between X_t and x_t is controlled by the exploration radius r. Furthermore, the learning rate η dictates how much x_t moves in response to the cumulative gradient. In our setting, as the timestep t increases, both r and η decrease, leading to more stable exploration and bringing the played action closer to x_t. The variations in r and η are shown in Figure 3.2.

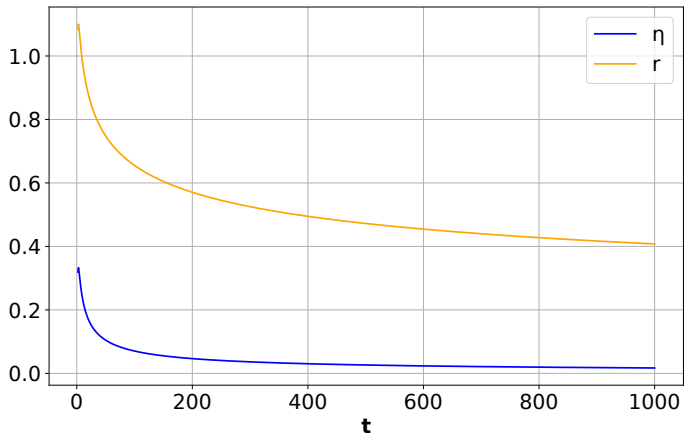


Figure 3.2: Changes of r and η.

While the decrease of r and η can improve precision over time, it may also introduce potential issues. As the timestep increases, r and η can become so small that exploration becomes insufficient. It is unrealistic to assume that network conditions

will stabilize as time progresses, which could hinder the model's ability to adapt to network changes. In high dynamic environments, even though the FTRL algorithm can update as bandwidth fluctuates, it might spend too much time exploring bandwidth variations. This issue is particularly pronounced in environments with high RTT, as the duration of each timestep depends on the RTT. Longer timesteps reduce the frequency of updates, making it even more difficult for the model to adapt to rapid changes in the environment, and can result in excessive exploration of the optimal bandwidth proportion.

Therefore, a dynamic detection algorithm is needed to identify abrupt environmental changes and prompt the model to reset r and η as necessary. Bifet et al. proposed ADWIN2, a time- and memory-efficient version of their distribution change detection algorithm, ADWIN [4]. This algorithm maintains a sliding window of recent data, divided into two sub-windows: one representing older data and the other newer data. It checks whether the difference in the averages of these sub-windows exceeds a threshold. If so, the algorithm discards the older sub-window while continuing to add new data to the sliding window. ADWIN does not require an explicit threshold or predefined window size; the only input is a confidence value δ. However, a significant computational cost arises from dividing the sub-windows, as the algorithm ideally needs to check all large enough sub-windows for possible cuts. Since the window size is unconstrained, it can grow substantially, especially in stable environments.

ADWIN2 is inspired by data stream algorithms and uses exponential histograms to group data, where the group sizes follow the pattern: 1, $\lfloor c \rfloor$, $\lfloor c^2 \rfloor$, ..., $\lfloor c^i \rfloor$, starting with the most recent data. The constant c can be adjusted based on the available memory, and in this work, we define $c = 2$. Additionally, instead of storing the raw data, we store the count and the sum of the data in each group, allowing us to efficiently compute average values. The ADWIN2 algorithm is presented in Algorithm 8 [4].

The core idea of the ADWIN2 algorithm is that when two sub-windows exhibit significantly different averages, the older window is discarded. In our work, we integrate ADWIN2 with the FTRL algorithm. Specifically, we use ADWIN2 to detect changes in the bandwidth ratio, bw_e/bw_{sum}, where bw_{sum} represents the sum of the estimated bandwidths for two paths. Although the estimated bandwidth is based on the receiving rate and may not always be perfectly accurate, it effectively captures abrupt environmental changes. Each time a new ACK is received, the bandwidth estimate is updated, and the bandwidth ratio is passed to the ADWIN sliding window,

Algorithm 8 ADWIN2

Initialize window W
for each $t > 0$ **do**
  $W \leftarrow W \cup \{x_t\}$ {Add x_t to the head of W}
  repeat
    Split W into W_0 and W_1 based on exponential histograms
    Calculate $n_0 = |W_0|$ and $n_1 = |W_1|$
    Calculate $m = \frac{1}{\frac{1}{n_0}+\frac{1}{n_1}}$
    Calculate $\delta' = \frac{\delta}{n}$
    Calculate $\epsilon_{\text{cut}} = \sqrt{\frac{1}{2m}\log\frac{4}{\delta'}}$
    Drop W_0
    Inform FTRL to reset η and r
  until $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \epsilon_{\text{cut}}$ holds for every split of W into $W = W_0 \cdot W_1$
end for
Output $\hat{\mu}_W$

W. ADWIN2 operates continuously in the background, identifying changes in the bandwidth ratio. When a change is detected, the older sub-window is discarded, and FTRL is informed to reset lr and r. A further benefit of ADWIN2 is its ability to store the mean values of both recent and older data. When these values shift from the range [0, 0.5] to [0.5, 1], or vice versa, the cumulative gradient in FTRL can be reset to zero, reducing the time spent on exploration. This enhances FTRL's performance in high dynamic environments by accelerating its exploration process.

3.3 WRR-based Scheduler

After the learner determines the weights for the two paths in the current timestep, packets should be sent according to those weights. A common approach to achieve this is by applying WRR. To send packets in a more discrete manner, ensuring both paths are utilized effectively and reducing idle time, we applied WRR [20] to allocate packets.

In the transmission model, we assume the transmission can be treated as fluid. However, in QUIC, the smallest unit that can be counted for a sequence number is a packet, and packet sizes can vary between paths. For file transfers, most packets sent from the sender after the initial probing stage follow the size of the MTU. Therefore, we use the MTU of the two paths to recalculate the weights for distributing packets.

The algorithm used to distribute traffic is shown in Algorithm 9.

Algorithm 9 WRR-based Packet Scheduling Decision Maker

if FTRL model updated **then**
  $P_0\_MTU\_Weight \leftarrow \frac{MTU_0}{MTU_0+MTU_1}$
  $P_1\_MTU\_Weight \leftarrow \frac{MTU_1}{MTU_0+MTU_1}$
  $Scaled\_Weight_0 \leftarrow \frac{X_t}{P_0\_MTU\_Weight}$
  $Scaled\_Weight_1 \leftarrow \frac{1-X_t}{P_1\_MTU\_Weight}$
  $P_0\_Weight \leftarrow \frac{Scaled\_Weight_0}{Scaled\_Weight_0+Scaled\_Weight_1}$
  $P_1\_Weight \leftarrow \frac{Scaled\_Weight_1}{Scaled\_Weight_0+Scaled\_Weight_1}$
  $Weight_e \leftarrow [P_0\_Weight, P_1\_Weight]$
  $Weight_c[0] \leftarrow 0$
  $Weight_c[1] \leftarrow 0$
end if
$Select \leftarrow -1$
$MaxCurrentWeight \leftarrow 0$
for $i \leftarrow 0$ **to** 1 **do**
  $Weight_c[i] \leftarrow Weight_c[i] + Weight_e[i]$
  if $Weight_c[i] > MaxCurrentWeight$ **then**
    $MaxCurrentWeight \leftarrow Weight_c[i]$
    $Select \leftarrow i$
  end if
end for
$Weight_c[Select] \leftarrow Weight_c[Select] - 1$
return $(Select, t)$

Each time the learning model is updated, the parameters for WRR are reset, and the effective weight $Weight_e$ is recalculated. The current weight $Weight_c$ for both paths is reset to zero. For each selected path, the corresponding FTRL model timestep t is returned to associate each packet with the current timestep. To reduce processing overhead, we can run the WRR algorithm in the background, executing it 1000 times and storing the decisions for the next 1000 rounds in a queue. When the scheduler needs a decision, it can immediately pop from the queue, while the WRR module continues to generate new decisions in the background.

3.4 Complexity Analysis

The overall structure of FTRL-WRR, combining all modules, is depicted in Figure 3.3.

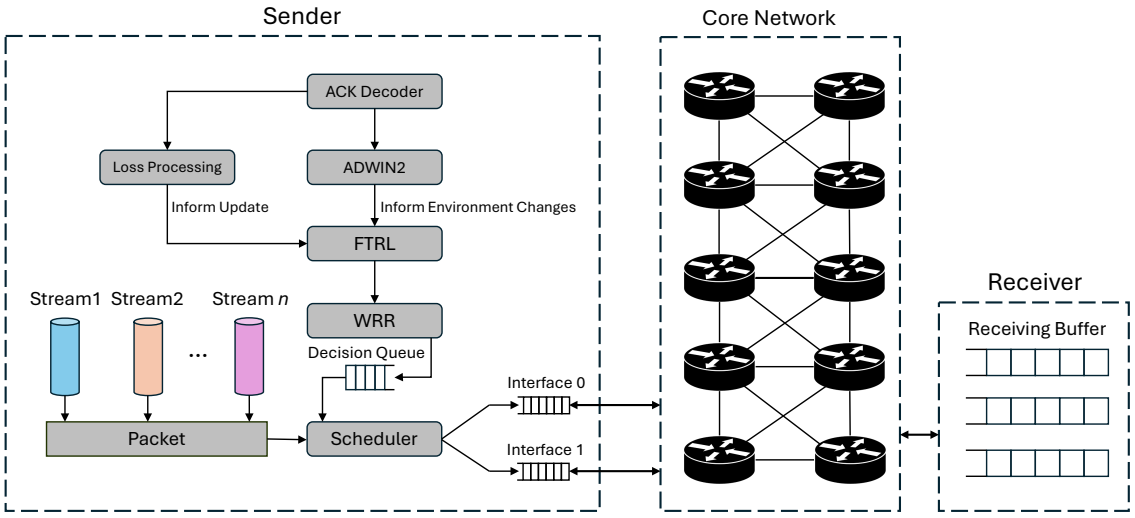


Figure 3.3: FTRL-WRR.

We first examine the complexity of the FTRL module. FTRL is a lightweight learning algorithm, where nearly all operations are arithmetic, leading to a time complexity of $O(1)$ per time step. Although the calculation of a involves summing all historical gradients, in practice, we maintain an incremental cumulative sum of a, which is updated efficiently. As a result, for n time steps, the overall time complexity of FTRL becomes $O(n)$. Furthermore, due to the delayed feedback in QUIC, whenever FTRL receives an update, it applies the update to the second-to-last time step, as shown in Figure 3.1. Consequently, after each update, the current information is stored for the next two time steps. Since only a fixed number of time steps are retained, there is no need to store additional historical data, resulting in a space complexity of $O(1)$.

ADWIN2 is a time- and memory-efficient algorithm. As discussed by Bifet et al. [4], when a new element arrives, ADWIN2 processes it in $O(1)$ amortized time or $O(\log W)$ in the worst case. This is because ADWIN2 maintains $O(\log W)$ cutpoints, and the arrival of a new element could trigger cascading updates throughout the data structure. Thus, the total time complexity ranges from $O(\log W)$ to $O(\log^2 W)$ in the worst case. Furthermore, ADWIN2's space complexity is $O(\log W)$ memory words, as it tracks $O(\log W)$ cutpoints.

The WRR module only involves arithmetic operations, giving it a time complexity of $O(1)$. Additionally, since it operates on a fixed number of variables and fixed-size arrays, its space complexity is $O(1)$. Once WRR makes a decision, that decision is stored in a queue. When the scheduler requires a decision, it can directly retrieve one from the queue without interacting with the WRR module. This completely isolates the scheduler from the background modules, ensuring that they do not introduce any transmission delays or block the transmission in any case. The queue has a fixed size of 1000, and when it is full, WRR temporarily stops generating new decisions and waits until the scheduler consumes entries from the head of the queue. As the queue has a fixed size, the space complexity of maintaining it is $O(1)$. Overall, the two-path scheduler operates with a time complexity of $O(1)$, as elements can be retrieved from the queue in constant time. For the complete FTRL-WRR algorithm, the time complexity ranges from $O(\log W)$ to $O(\log^2 W)$, primarily due to the ADWIN2 module. The space complexity is $O(\log W)$, also attributed to the ADWIN2 module.

Chapter 4

Experiments and Evaluations

In this chapter, we present experiments to comprehensively evaluate the performance of FTRL-WRR across various scenarios and compare it with other scheduling algorithms. First, we conduct experiments in a customized environment to highlight the advantages of FTRL-WRR. Then, we examine the behavior of various scheduling algorithms in combination with different CCAs. Next, we focus on evaluations within LEO networks, starting with emulation-based testing. We use Mininet and `tc-netem` to accurately replay real LEO satellite network behavior, utilizing real-world Starlink traces for delay and bandwidth collected by Zhao et al. [30].

Starlink operates under two primary architectures: "Bent-Pipe" and Inter-Satellite Link (ISL) [30]. In the "Bent-Pipe" architecture, a single satellite is used between the user terminal (UT) and the ground station (GS), resulting in relatively stable delay variations. In contrast, ISL involves multiple satellites between the UT and GS, introducing greater dynamics. Our evaluation focuses on four representative scenarios: Starlink "Bent-Pipe" + Cellular, Starlink ISL + Cellular, Starlink ISL + Starlink ISL, and Starlink ISL + OneWeb.

Finally, we introduce a pacing mechanism to enhance FTRL-WRR's robustness in real-world scenarios and evaluate the algorithm on a real testbed that involves Starlink and Fiber.

4.1 Starlink Emulator

Maintaining accuracy and precision is crucial for our emulation-based experiments. To achieve this, we developed a trace-driven emulator that replays real-world traces

as input[2] . While Mininet offers a reliable platform for standard network emulation, it lacks built-in dynamic support. To address this, we use `tc-netem`, a tool provided by the Linux kernel, to introduce dynamic behavior based on real traces, allowing us to accurately emulate network conditions. The emulation topology is illustrated in Figure 4.1.

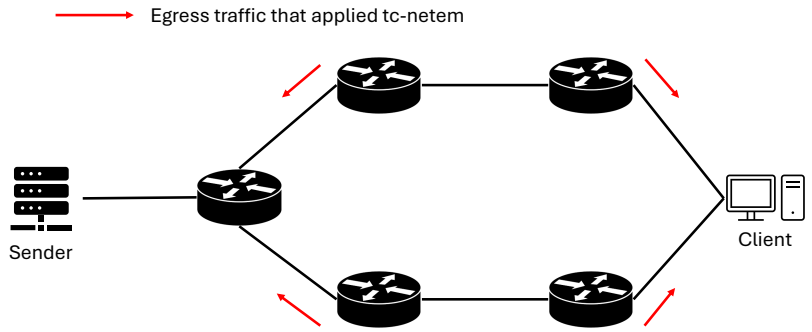
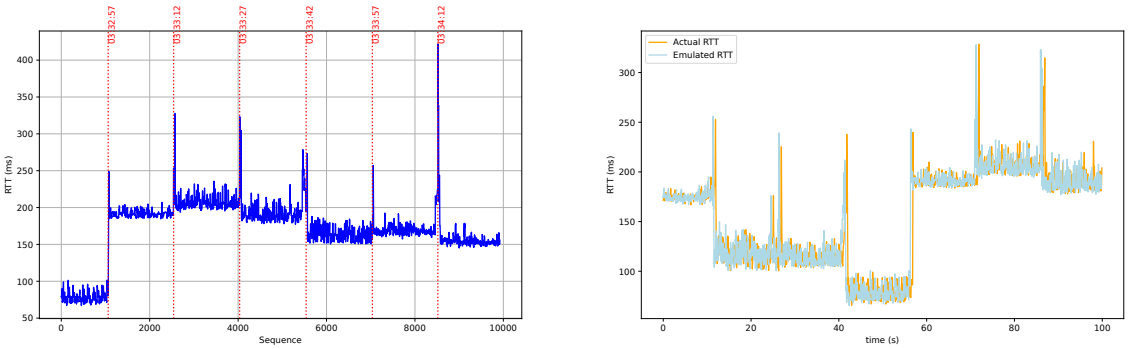


Figure 4.1: Emulator Topology.

We have two hosts and five routers, with four routers configured using `tc-netem` to emulate dynamic network conditions. Since `tc-netem` only affects egress traffic, it is necessary to specify the interface of each router where changes will be applied. The red lines in the figure indicate the traffic dynamics in two directions across two paths.



(a) Wall Time Emulation

(b) Trace-Driven Emulation

Figure 4.2: Emulator.

Additionally, Starlink exhibits global handover behavior at 12-27-42-57 seconds of each minute, during which metrics like latency or bandwidth may change abruptly. Therefore, synchronizing with real time is critical when emulating Starlink. To ensure

[2] `https://github.com/Peter-LiDP/Starlink-Emulator`

precision, we incorporate a virtual timestamp when processing data and a wall time synchronization mechanism to emulate handovers precisely at the 12-27-42-57 seconds marks of each minute. Users can also specify a start time, allowing them to begin at any point in time and still observe the handover pattern every 15 seconds. To maintain long-term precision and prevent cumulative deviations caused by updates, a self-correction mechanism adjusts for offset deviations between the start time and wall time. If this deviation exceeds 50 ms, the system triggers an self-correct to ensure sustained accuracy. Some results of the emulation are shown in Figure 4.2. During the LEO satellite emulation, we applied a 2% loss rate, while cellular emulation used a 1% loss rate. Both latency and bandwidth followed the data trace, updating every 100 ms.

4.2 Evaluation with Customized Topology

To thoroughly evaluate the behavior of FTRL-WRR, we designed a customized testing environment using Mininet to compare its performance against various scheduling algorithms. The environment consisted of two paths: Path 1 with a bandwidth of 150 Mbps, a default one-way delay (OWD) of 100 ms, and a packet loss rate of 2%, and Path 2 with a bandwidth of 50 Mbps, a default OWD of 20 ms, and a packet loss rate of 1%. We tailored this environment into three scenarios representing low, medium, and high dynamics by varying the OWD of Path 1 by 0%, 10%, and 20% from the default value, respectively, while Path 2 consistently experienced a 10% variation in OWD. A 150 MB file was transferred using different scheduling algorithms, with each scenario repeated 50 times to ensure statistical significance. Jahandar et al. demonstrated that BBR outperforms CUBIC in a Starlink environment, achieving faster link capacity utilization, especially in scenarios with high packet loss [8]. Additionally, Austria et al. [3] suggested that BBR is a suitable congestion control algorithm for environments with paths exhibiting large latency differences, such as satellite networks. Therefore, we used BBRv1 as the default CCA. The impact of different CCAs will be analyzed in Section 4.3. We implemented different scheduling algorithms on Picoquic. The flow completion times and the corresponding Cumulative Distribution Function (CDF) are shown in Figure 4.3.

For evaluation, we use flow completion time as the primary metric, which indicates the time taken to transmit a fixed-size file. In this customized setting, FTRL-WRR demonstrates a significant decrease in flow completion time, consistently outperform-

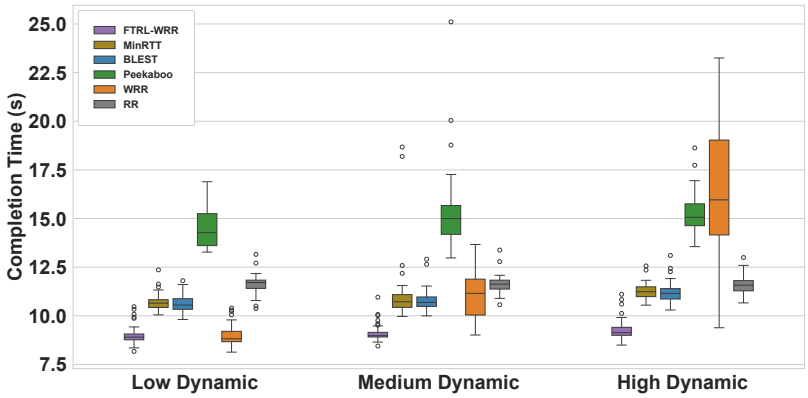


Figure 4.3: Flow Completion Time.

ing other algorithms. In a low dynamic environment, both FTRL-WRR and WRR show promising performance, highlighting the effectiveness of WRR-based strategies in proportionally distributing traffic in environments with heterogeneous bandwidth. FTRL-WRR outperforms WRR in terms of mean flow completion time by 0.09% and MinRTT by 15.95%. Notably, in our implementation, RR is designed as the FTRL-WRR algorithm without the learning module. After initializing X_t to 0.5, the transmission no longer updates this weight, but all other components remain the same as in the original FTRL-WRR algorithm. This approach results in a strategy similar to RR, where traffic is evenly distributed between the two paths. However, slight adjustments based on packet size and redirection of traffic during congestion, as in the original FTRL-WRR, still occur. This setup was designed to evaluate the impact of the learning module. In the low dynamic environment, FTRL-WRR decreases the mean flow completion time by 22.38% compared to RR, demonstrating the effectiveness of the FTRL module.

In the medium dynamic environment, WRR's performance is significantly impacted by its conservative pacing mechanism, which relies entirely on the current bottleneck bandwidth estimate. In this setting, FTRL-WRR reduces mean flow completion time by 17.78% compared to MinRTT, 15.51% compared to BLEST, 40.54% compared to Peekaboo, 17.97% compared to WRR, and 21.7% compared to RR. In the high dynamic environment, FTRL-WRR reduces mean flow completion time by 17.65% compared to MinRTT and 42.46% compared to WRR. This confirms the effectiveness of FTRL-WRR, particularly in high-dynamic environments, as the algorithm consistently functions well to maximize bandwidth utilization.

Additionally, it is notable that Peekaboo does not achieve ideal performance across

all three scenarios. We discussed Peekaboo's limitations, including inevitable performance loss due to the exploration of different actions and a fixed learning overhead that may potentially confuse the learning process. Moreover, the authors confirm that in high dynamic environments, where network changes outpace the learning speed, Peekaboo may struggle to perform optimally [28]. Therefore, in the subsequent evaluation with LEO satellite traces, which exhibit even more dynamic changes in both bandwidth and delay, we will exclude Peekaboo due to its limitations in such environments.

4.3 Study with Different Congestion Control Algorithms

In previous experiments, we selected BBRv1 as the default congestion control algorithm due to its resilience to packet loss and robust performance in LEO satellite environments. However, it remains crucial to study how different congestion control algorithms affect scheduling behavior. BBRv1 primarily uses bottleneck bandwidth estimates and RTT as congestion signals, while BBRv3 incorporates ECN and packet loss into its congestion control, making it more sensitive to loss events. Meanwhile, CUBIC primarily relies on packet loss as a congestion signal.

In this section, we tested different algorithms—BBRv1, BBRv3, and CUBIC—under a customized environment similar to Section 4.2, where the bandwidth of path 0 and path 1 was set to 150 Mbps and 50 Mbps, and the base delay to 100 ms and 20 ms, respectively. In a low dynamic environment, the delay of path 0 remained constant, while in a medium dynamic environment, the delay of path 0 varied by 10% based on the base delay. In a high dynamic environment, the delay of path 0 varied by 20%. The delay of path 1 always varied by 10% across all three environments. For evaluation, we transferred a 150 MB file and repeated the transfer 10 times for each algorithm under each scenario. Since some of these algorithms are sensitive to packet loss, we considered two scenarios: one without intentional packet loss and another with manually set packet loss rates of 2% and 1% on the two paths, respectively.

4.3.1 Environment without Intentional Packet Loss

We begin by running different algorithms in low, medium, and high dynamic scenarios with BBRv1. The results are presented in Figure 4.4. Under BBRv1, the results

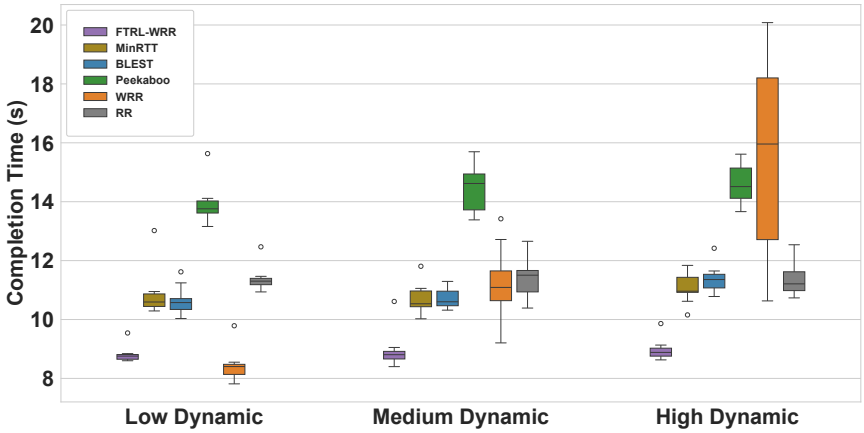


Figure 4.4: Flow Completion Time (BBRv1).

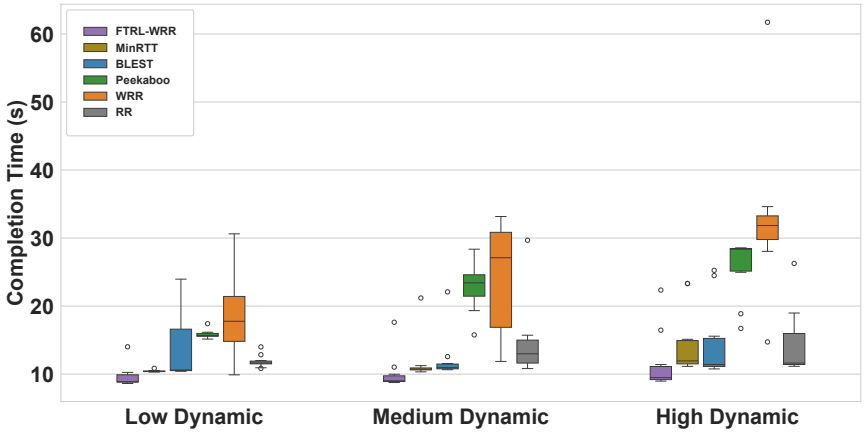


Figure 4.5: Flow Completion Time (BBRv3).

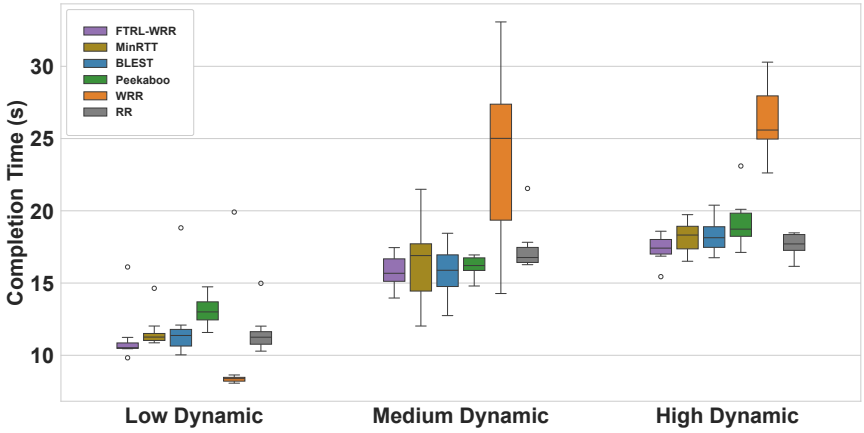


Figure 4.6: Flow Completion Time (Cubic).

are similar to those in Section 4.2. In a low dynamic environment, FTRL-WRR performs slightly worse than WRR due to the exploration process, which introduces additional overhead. As the dynamic level increases, WRR is significantly affected, whereas FTRL-WRR remains stable. Additionally, FTRL-WRR achieves a lower flow completion time compared to MinRTT, reducing the mean flow completion time by more than 16.4% across all three dynamic levels.

The results using BBRv3 as the CCA are presented in Figure 4.5. Compared to BBRv1, BBRv3 has made several adjustments, such as incorporating additional states and refining the startup and recovery mechanisms. One of the most significant changes is that BBRv3 responds more aggressively to loss and ECN signals, reducing the congestion window more swiftly in reaction to these events. Since we did not introduce additional packet loss, most of the results are similar to BBRv1. However, it is notable that in some trials, the flow completion time was significantly longer, indicating BBRv3's aggressive response to occasional packet loss. Across all three dynamic levels, FTRL-WRR consistently outperformed the others. Specifically, FTRL-WRR reduced the mean flow completion time by 7.92%, 14.03%, and 20.35% compared to MinRTT in low, medium, and high dynamic environments, respectively. Additionally, FTRL-WRR reduced the mean flow completion time by 49.03%, 58.15%, and 64.75% compared to WRR in low, medium, and high dynamic environments, respectively.

We also conducted tests using Cubic as the CCA, and the results are presented in Figure 4.6. Unlike BBR, which is a model-based CCA that continuously probes bottleneck bandwidth and RTT to update the CWND, Cubic is primarily loss-based, using loss and ECN as its primary congestion signals, with significant RTT increases considered as auxiliary signals. This makes Cubic more sensitive to random packet loss, potentially leading to unnecessary reductions in CWND. From the results, it is evident that for most trials, the flow completion time is longer compared to BBR. Even though we did not intentionally introduce additional packet loss, the increase in dynamic levels may still have caused more random packet loss, triggering CWND reductions. In the low dynamic environment, FTRL-WRR exhibits higher flow completion times than WRR. This is because WRR includes a pacing mechanism based on the bottleneck bandwidth estimate by default, resulting in a more conservative strategy with fewer packet losses, which improves performance under loss-based CCA. FTRL-WRR does not include pacing by default, though we introduced a method to incorporate pacing into FTRL-WRR, as discussed in Section 4.8. As the dynamic level increases, WRR becomes significantly affected. FTRL-WRR generally achieves

lower flow completion times in these scenarios, but the difference is less pronounced compared to when BBR is used as the CCA.

4.3.2 Environment with Intentional Packet Loss

We also conducted tests in an environment with intentional packet loss of 2% and 1% on two paths, respectively. The results of these tests under BBRv1 are discussed in Section 4.2. Additionally, we investigated the behavior of different scheduling algorithms using a loss-based CCA in a lossy environment. For this, we selected Cubic to perform our tests. The results are presented in Figure 4.7.

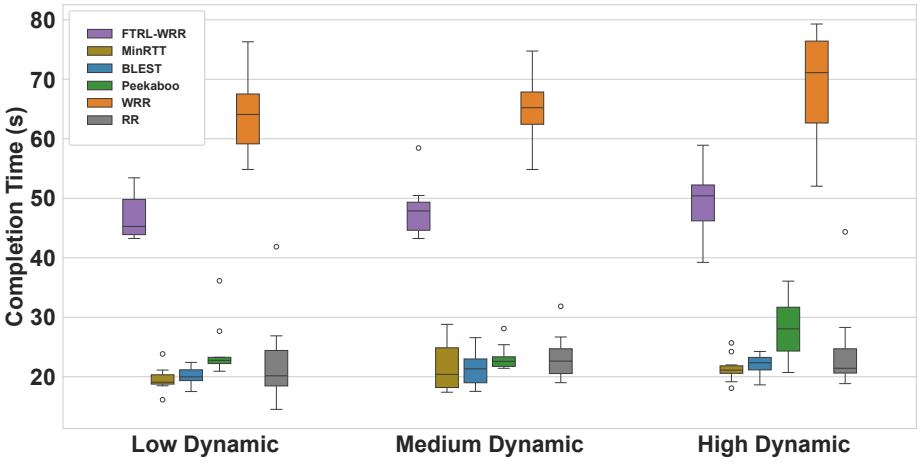


Figure 4.7: Flow Completion Time in Lossy Environment (Cubic).

FTRL-WRR exhibits significant performance degradation, similar to WRR. The primary reason for this lies in the CCA. On path 0, which offers high bandwidth but experiences a high packet loss rate, assigning more traffic increases the likelihood of CWND reduction due to the loss-based CCA. Consequently, increasing traffic on path 0 may not improve throughput and could even reduce it, which misguides the optimization algorithm. This issue is evident when even RR, which is FTRL-WRR without the learning module, achieves a lower flow completion time—a clear sign that the learning module is misled by the CCA. Therefore, applying FTRL-WRR requires careful selection of a suitable CCA to prevent it from adversely affecting the learning process.

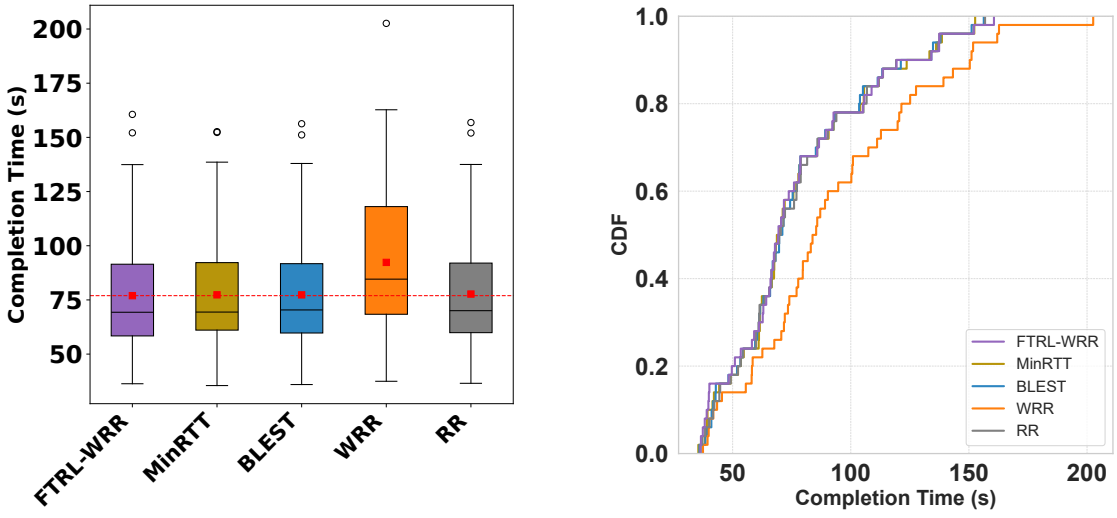
Additionally, the algorithm design accounts for congestion control. When congestion control constrains performance, the model attempts to adapt to the traffic distribution dictated by the CCA. Under these conditions, the scheduler cannot man-

age traffic distribution as intended, as discussed in Section 3.1. When CCAs operate ineffectively and consistently deviate from the true path capacity, the algorithm cannot converge to the optimal distribution as expected.

In the same environment, BBRv1 delivers the expected performance, as described in Section 4.2. From various experiments, we observed that BBRv1 generally performs better due to its resilience to packet loss. Therefore, BBRv1 is selected as the default CCA for the following evaluation.

4.4 Evaluation on Starlink "Bent-Pipe" + Cellular

In this setting, we test a scenario where users are connected to both Starlink and cellular networks, with Starlink operating under a "Bent-Pipe" architecture. For most regions, Starlink employs the "Bent-Pipe" architecture [30], where there is only a single satellite between the User Terminal (UT) and the Ground Station (GS). This architecture typically results in lower delays and reduced dynamics. For the Starlink "Bent-Pipe" data, we utilized traces collected in Victoria by Zhao et al. [30]. The cellular data, which includes RTT and bandwidth measurements during 5G file downloads, was collected by Raca et al. [24]. Our evaluation involved continuously running the emulator and triggering a 1 GB file transfer every 200 seconds, repeated 50 times. The results are shown in Figure 4.8.



(a) Flow Completion Time

(b) CDF Plot

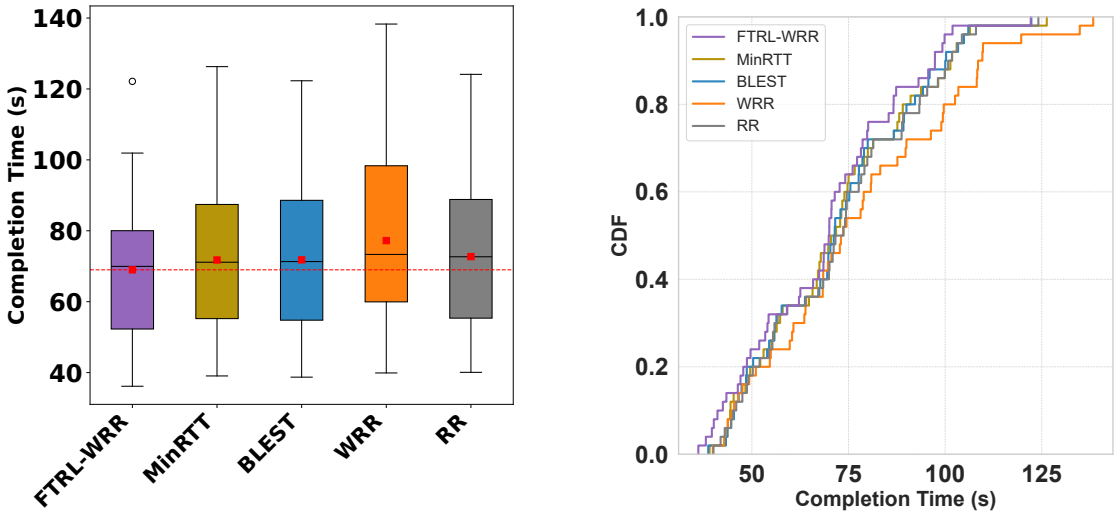
Figure 4.8: Results from Starlink "Bent-Pipe" + 5G.

In this setting, most schedulers demonstrate very similar performance, with the exception of WRR, which shows a noticeable performance loss compared to the others. Specifically, FTRL-WRR achieves up to a 16.63% improvement in average flow completion time over WRR and a 0.47% improvement compared to BLEST. Notably, even RR, which is FTRL-WRR without the learning model, is only 0.96% slower than FTRL-WRR in average flow completion time. The RR algorithm is primarily influenced by CCAs, as its scheduling strategy only aims to distribute traffic evenly. This indicates that in a low-dynamic environment with relatively low delay, CCAs are sufficient to help most algorithms achieve strong performance by proportionally distributing traffic, leaving limited room for further improvements through advanced scheduling strategies. Nonetheless, FTRL-WRR still achieves the lowest median and average flow completion times, underscoring the effectiveness of its learning module in optimizing traffic distribution.

4.5 Evaluation on Starlink ISL + Cellular

In this scenario, a user connects to both Starlink and 5G networks to achieve low-latency and high-bandwidth transmission, leveraging the strengths of each technology. In certain regions, Starlink utilizes an inter-satellite link (ISL) architecture, which typically provides higher bandwidth but with more variable latency. To capture these characteristics, we used latency and bandwidth data collected by Zhao et al. [30] from a Starlink dish utilized ISL in Seychelles. For the cellular network emulation, we employed 5G data based on RTT and bandwidth measurements during file downloads, as gathered by Raca et al. [24]. In this context, we assess the performance of schedulers operating in a Starlink ISL + 5G setup. Similarly, we ran the emulator continuously, sending a 1GB file every 200 seconds for a total of 50 transmissions. The results are shown in Figure 4.9.

It is evident that FTRL-WRR demonstrates clear performance improvements. For FTRL-WRR, the highest, lowest, median, and average flow completion times are all lower than those of other algorithms. The FTRL-WRR scheduling algorithm consistently outperforms others, achieving up to a 10.66% reduction in average flow completion time compared to WRR and a 3.86% improvement over MinRTT. This trend is especially pronounced in the CDF plot, where trials with higher flow completion times, which indicate more challenging environments, show even greater improvements with FTRL-WRR compared to other algorithms. Even in other scenarios,



(a) Flow Completion Time

(b) CDF Plot

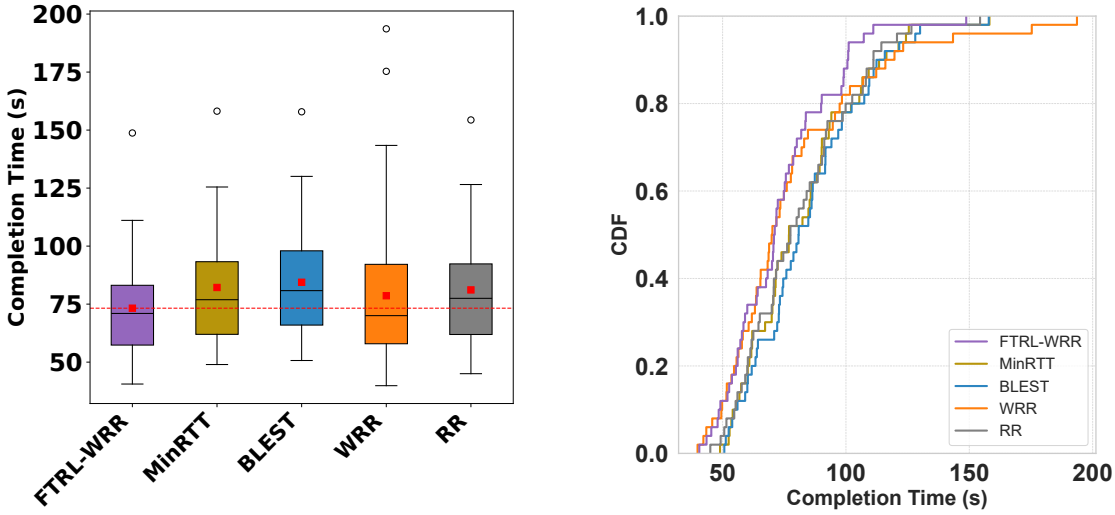
Figure 4.9: Results from Starlink ISL + 5G.

FTRL-WRR still delivers general performance gains. In contrast, WRR, which bases its weights on the current bottleneck bandwidth estimate, struggles in these more difficult environments, resulting in a notable performance loss compared to the other algorithms. These results show that by using the convex optimization model to continuously seek the global minimum from the loss function, FTRL-WRR can effectively optimize traffic distribution, even when bottleneck bandwidth estimates are inaccurate. Furthermore, RR, implemented as FTRL-WRR without the learning model, allocates traffic evenly but redirects traffic during congestion. The performance improvements of FTRL-WRR over RR highlight the significance and effectiveness of the learning module.

4.6 Evaluation on Starlink ISL + Starlink ISL

In this scenario, a user establishes two Starlink connections using two side-by-side dishes. Since these dishes are likely connected to the same satellite due to the satellite selection strategy [30], similar latency and bandwidth can be expected across both paths. In our emulation, we used data collected from Seychelles for both paths, starting from the same timestamp. The emulation runs continuously, transferring a 1 GB file every 200 ms, repeating this process 50 times for each algorithm. Since the same data trace is used and both paths start from the same point, the latency and

bandwidth are expected to be identical across the two paths. Packet loss was set to 2% on both paths. The results are presented in Figure 4.10.



(a) Flow Completion Time

(b) CDF Plot

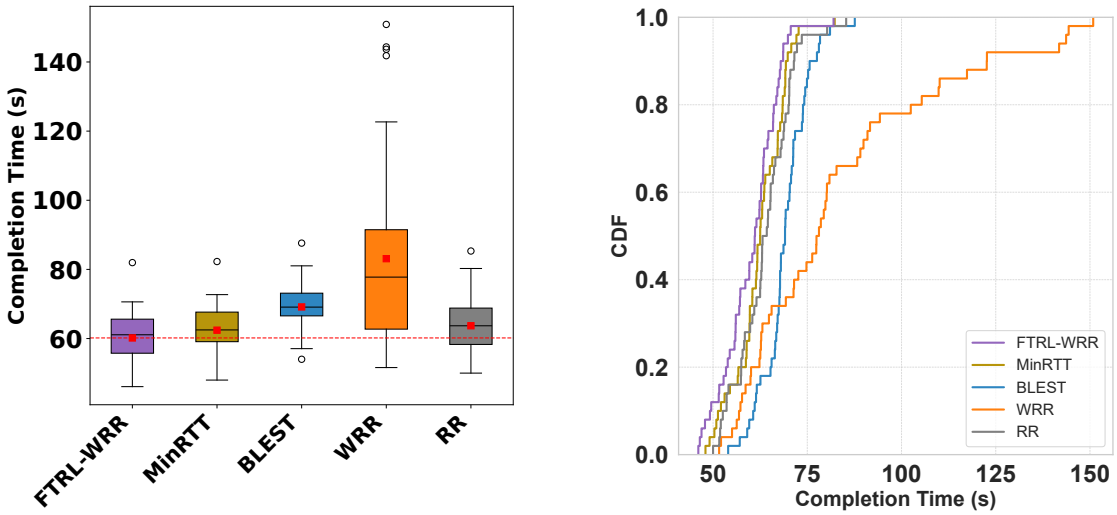
Figure 4.10: Results from Starlink ISL + Starlink ISL

In this scenario, FTRL-WRR shows a clear decrease in flow completion time compared to the other algorithms. Specifically, FTRL-WRR reduces mean flow completion time by 6.83% compared to WRR, 9.71% compared to RR, 10.86% compared to MinRTT, and 13.23% compared to BLEST. This trend is particularly evident in trials with generally longer flow completion times, as shown in Figure 4.10b. Notably, this scenario involves two paths with identical delay and bandwidth, so the optimal traffic distribution is expected to be close to 50/50. As a result, RR performs better than MinRTT. However, FTRL-WRR implicitly considers both paths jointly in finding the optimal traffic distribution, rather than treating them separately, allowing it to balance traffic more effectively and achieve significantly better performance than the others.

4.7 Evaluation on Starlink ISL + OneWeb

In addition to Starlink, OneWeb is a project that provides LEO networks. A scenario that combines both Starlink's ISL and OneWeb's network is particularly interesting, as it leverages two LEO networks to deliver high-bandwidth transmission in remote regions. OneWeb operates most of its satellites at an altitude of 1,200 km [10], which

is slightly higher than Starlink's 550 km [19], potentially resulting in more stable latency. Zhao et al. [30] collected OneWeb delay traces from Alaska, spanning an hour, and we incorporated these traces into our emulation. OneWeb can achieve up to 195 Mbps bandwidth [23], and in our emulation, we randomly generate bandwidth for each trace between 80 Mbps and 120 Mbps. The Starlink ISL emulation used the same traces as in previous experiments, collected from Seychelles by Zhao et al. [30]. Packet loss was set to 2% on both paths. Our emulator ran continuously, sending a 1GB file every 200 seconds, repeated 50 times for all algorithms. The results are shown in Figure 4.11.



(a) Flow Completion Time

(b) CDF Plot

Figure 4.11: Results from Starlink ISL + OneWeb.

In this scenario, FTRL-WRR demonstrates the best performance among the algorithms. It reduces the mean flow completion time by 3.6% compared to MinRTT and by 27.59% compared to WRR. WRR suffers from significant performance loss in this environment due to the heterogeneous nature of the paths, with both experiencing dynamic bandwidth levels. As shown in the CDF plot, FTRL-WRR consistently outperforms the others, indicating a clearer trend of improvement. These results further confirm FTRL-WRR's effectiveness in LEO networks, particularly in environments with higher dynamics.

4.8 Real Starlink Testbed

4.8.1 Pacing

To conduct evaluations on a real Starlink testbed instead of a simple emulation topology, it is essential to enable pacing. Pacing helps mitigate traffic bursts, which can cause congestion and buffer overflows and negatively affect transmission. Picoquic originally includes a built-in pacing mechanism, though we disabled it in previous evaluations and replaced it with the WRR module, as discussed in Section 3.3. The original pacing mechanism, based on the leaky bucket algorithm, is detailed in Section 2.2.5, with its pacing rate determined by the estimated bottleneck bandwidth. We can modify this bandwidth estimate to integrate pacing with FTRL-WRR.

FTRL-WRR returns a weight at each timestep. The bottleneck bandwidth estimate in pacing can be adjusted using this weight. Assume Path 0 and Path 1 have bandwidth estimates bw_{e0} and bw_{e1}, respectively. The proportion of these bandwidth estimates naturally becomes the weights for each path. Let w_i represent the weight for path i. The weights for Path 0 and Path 1 can then be expressed as:

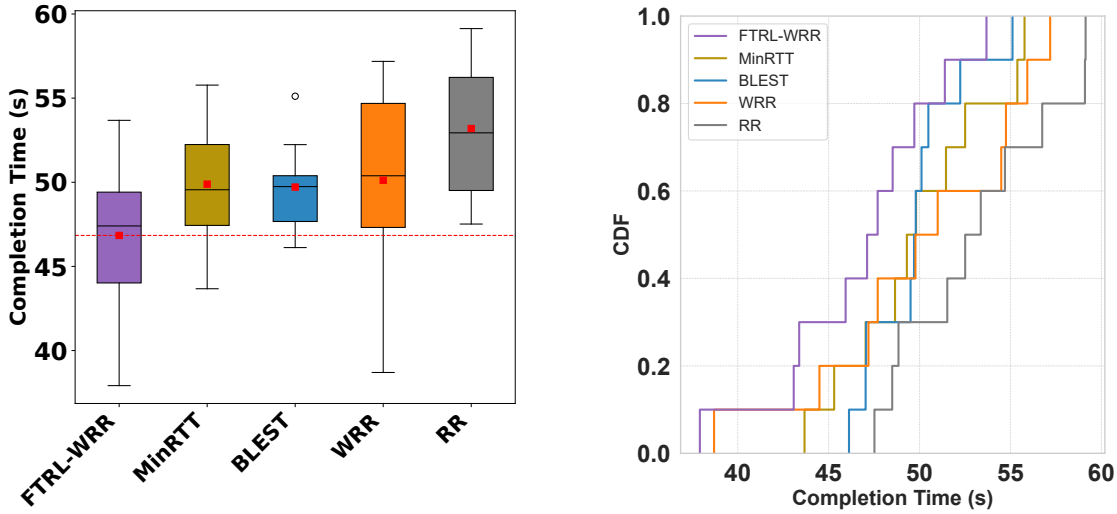
$$w_0 : w_1 = \frac{bw_{e0}}{bw_{e0} + bw_{e1}} : \frac{bw_{e1}}{bw_{e0} + bw_{e1}}, \tag{4.1}$$

which corresponds to the weights used in the original WRR in Picoquic. At timestep t, FTRL returns a weight X_t for Path 0. In the pacing rate calculation, if X_t is smaller than the weight w_0 derived from the probed bottleneck bandwidth, we can infer that the total bandwidth is $\frac{bw_{e0}}{X_t}$. Consequently, the bandwidth estimate for Path 1 can be increased according to the proportion generated by FTRL, expressed as $(1 - X_t)\frac{bw_{e0}}{X_t}$. Similarly, if $1 - X_t$ is smaller than the weight w_1, we can update the bandwidth estimate for Path 0 as $X_t \frac{bw_{e0}}{1-X_t}$. After these adjustments, the pacing rate calculated from the updated bandwidth estimates will reflect the weights produced by the FTRL model. It's important to note that these adjustments are only applied to the pacing rate, not the bandwidth estimates themselves, ensuring that congestion control still relies on the actual bandwidth estimates.

4.8.2 Evaluation on Real Starlink + Fiber

To evaluate our algorithm in a real testbed, we established a two-path environment. The server was deployed on the Google Cloud Platform (GCP), while the client,

running Ubuntu 22.04.2 LTS, was located at the University of Victoria (UVic). One path was configured through an Ethernet connection to Starlink using a user terminal placed at UVic, while the other path connected via fiber, routing through UVic, BCNET, and CANARIE to the GCP. We conducted tests by transferring a 1 GB file from the server to the client using Picoquic with multipath enabled, and evaluated different scheduling algorithms. Each algorithm was tested over 10 trials, with the results presented in Figure 4.12.



(a) Flow Completion Time

(b) CDF Plot

Figure 4.12: Results from Starlink + Fiber.

The results demonstrate that FTRL-WRR consistently reduces flow completion time in most trials. Specifically, FTRL-WRR reduces the mean flow completion time by 5.78% compared to BLEST, 6.11% compared to MinRTT, 6.54% compared to WRR, and 11.94% compared to the RR. These improvements can be attributed to the FTRL model's ability to optimize bandwidth utilization, along with the newly adjusted pacing mechanism that more efficiently handles bandwidth. These findings highlight the effectiveness and applicability of FTRL-WRR in real-world scenarios.

Chapter 5

Future Work and Conclusions

5.1 Limitations and Future Work

We present the results from both emulated environments and a real testbed, discussing the impact of different congestion control algorithms. The findings highlight the effectiveness of FTRL-WRR, particularly in dynamic environments. However, there are some limitations.

Currently, FTRL-WRR only handles a two-path scenario, where the problem is simplified into a 1-dimensional optimization. Extending this to an n-path scenario introduces an $(n-1)$-dimensional optimization problem, as the sum of traffic proportions over all paths must equal 1. While extending the framework to n-path scenarios is feasible, each additional path requires algorithmic adjustments. Let w_i represent the weight for path i, the proportion of data assigned to path i. In an $(n+1)$-path scenario, the optimization becomes n-dimensional, and the feasible set is defined as $K = \{w \in \mathbb{R}^n : 0 < w_i < 1 \text{ for } i = 1, \ldots, n; \sum_{i=1}^{n} w_i < 1\}$. The self-concordant barrier $R(w)$ is given by $R(w) = -\sum_{i=1}^{n} \ln(w_i) - \ln\left(1 - \sum_{i=1}^{n} w_i\right)$ [15], and the second derivative, $R''(w)$, forms an n-dimensional Hessian matrix. Additionally, instead of sampling ξ from two points, -1 or 1, it must now be sampled from the surface of an n-dimensional unit ball $\mathbb{S}_1^{n-1}$. After playing an action, the loss value Y_t is observed, and the estimated gradient g_t, an n-dimensional vector, is calculated. Once the weights $(w_1, w_2, ..., w_n)$ are generated, they are applied to WRR as path weights. Although this framework can be extended to multipath scenarios, the matrix calculations involved result in significantly higher computational overhead. Additionally, unlike in the 1-dimensional case where exploration is efficient due to only two directions (pos-

itive or negative) being considered, in the multi-dimensional case, exploration spans infinitely many directions. This could lead to a substantial increase in exploration time. One possible approach to mitigate unnecessary probing is to use the bottleneck bandwidth estimate for each path as an initial weight and then restrict exploration to a neighboring region. Future work can explore extending this approach to multipath scenarios and analyze its convergence behavior in such cases.

Moreover, as demonstrated in Section 4.3.2, deviations introduced by CCAs can be amplified by FTRL-WRR, potentially leading to degraded performance. Therefore, selecting a suitable CCA for the given environment is critical. In our evaluation, we chose BBRv1 for its robustness against packet loss. Future research can explore testing FTRL-WRR with other robust CCAs that are effective in lossy environments. Additionally, as discussed in Section 4.8, the weights generated by FTRL-WRR can be incorporated into bottleneck bandwidth estimation to assist in pacing. This integration could also be extended to model-based CCAs, enabling smoother coordination between the scheduler and CCAs. Future work can focus on developing CCAs that are specifically designed to collaborate with FTRL-WRR.

FTRL-WRR has been implemented in Picoquic, but its integration is not limited to this system. Many platforms supporting the multipath extension of QUIC could also incorporate FTRL-WRR. The first MPQUIC prototype was developed by De Coninck et al. in QUIC-GO [7]. Other implementations, such as QUICHE [6], which follows IETF standards and offers a low-level API for QUIC packet processing and connection management, also support multipath QUIC. Similarly, XQUIC[2], developed by Alibaba, provides multipath QUIC support. Furthermore, Shu et al. [25] have implemented MPQUIC in the ns-3 network simulator. Future work may involve integrating FTRL-WRR into these platforms to enable more extensive evaluations.

Finally, as noted in Section 4.3.1, FTRL-WRR's performance may be impacted in low-dynamic environments due to continuous exploration. While exploration is a fundamental aspect of most online learning algorithms, the overhead it introduces can potentially be reduced. Future work could explore ways to minimize the exploration overhead in the FTRL algorithm.

5.2 Conclusions

In this thesis, we focused on a two-path transmission scenario and proposed FTRL-WRR, a learning-based algorithm designed to optimize traffic distribution. We began

by analyzing existing MPQUIC schedulers, evaluating their strengths and limitations. Building on this, we developed a two-path transmission model and observed that the traffic distribution-throughput relationship generally follows a quasi-concave function. This insight motivated us to apply bandit convex optimization to address the traffic distribution challenge. After thoroughly examining bandit convex optimization, we justified our choice of FTRL as the underlying framework for our solution.

We provided a detailed explanation of the FTRL-WRR algorithm, including its three main components: the FTRL-based learner, the ADWIN2-based distribution change detector, and the WRR-based scheduler. A complexity analysis was also presented. To assess the behavior and performance of FTRL-WRR, we first evaluated it in a customized test environment, demonstrating its efficacy in high dynamic scenarios. We further explored its interaction with different CCAs.

Next, we developed a Starlink emulator using Mininet and `tc-netem`, incorporating real-world traces. We compared FTRL-WRR with multiple schedulers across various emulated environments, including Starlink "Bent-Pipe" + 5G, Starlink ISL + 5G, Starlink ISL + Starlink ISL, and Starlink ISL + OneWeb. The results consistently showed that FTRL-WRR achieves lower flow completion time, confirming its effectiveness in LEO network scenarios. We also conducted experiments on a real-world testbed with Starlink and fiber.

To address the challenges of applying FTRL-WRR in real-world testbeds, we designed and implemented a pacing mechanism. This mechanism ensures that the weights generated by the FTRL module are utilized efficiently while preventing burst traffic from impacting performance. In our tests with the Starlink + fiber testbed, we confirmed that FTRL-WRR effectively reduces flow completion time in real-world scenarios.

After a thorough evaluation, we summarized the current limitations and suggested directions for future research, including extending the current two-path scheduler to support multipath scenarios, enhancing CCAs to work effectively with FTRL-WRR, implementing FTRL-WRR across different platforms, and improving the exploration efficiency of the FTRL algorithm.

Overall, this thesis presents a step forward in improving two-path scheduling for dynamic network environments, especially in LEO satellite and heterogeneous network scenarios. The development and evaluation of FTRL-WRR demonstrate its ability to enhance network performance by effectively distributing traffic in different network conditions. Through both emulated and real-world tests, FTRL-WRR has

shown its ability to reduce flow completion times and maintain robust performance in high dynamic environments. Moving forward, we hope the proposed methodologies contribute to further progress in multipath scheduling and adaptive traffic management across emerging network technologies.

Bibliography

[1] Satellite Statistics, 2024. `https://planet4589.org/space/con/star/stats.html`.

[2] Alibaba. XQUIC, 2024. `https://github.com/alibaba/xquic`.

[3] Austria, Phillipe, Park, Chol Hyun, Jo, Ju-Yeon, Kim, Yoohwan, Sundaresan, Rahul, and Pham, Khanh. BBR Congestion Control Analysis with Multipath TCP (MPTCP) and Asymmetrical Latency Subflow. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 1065–1069. IEEE, 2022.

[4] Bifet, Albert and Gavalda, Ricard. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 443–448. SIAM, 2007.

[5] Bonaventure, Olivier, Paasch, Christoph, and Barré, Sébastien. The Multipath TCP Project. `https://www.multipath-tcp.org/`, 2024.

[6] Cloudflare. QUICHE, 2024. `https://github.com/cloudflare/quiche`.

[7] De Coninck, Quentin and Bonaventure, Olivier. Multipath QUIC: Design and Evaluation. In *Proceedings of the 13th international conference on emerging networking experiments and technologies*, pages 160–166, 2017.

[8] Deutschmann, Jörg, Jahandar, Saeid, Hielscher, Kai-Steffen, and German, Reinhard. Internet via Satellite: GEO vs. LEO, OpenVPN vs. Wireguard, and CUBIC vs. BBR. In *Proceedings of the 1st ACM MobiCom Workshop on Satellite Networking and Computing*, pages 19–24, 2023.

[9] Ferlin, Simone, Alay, Özgü, Mehani, Olivier, and Boreli, Roksana. BLEST: Blocking Estimation-Based MPTCP Scheduler for Heterogeneous Networks. In

2016 IFIP networking conference (IFIP networking) and workshops, pages 431–439. IEEE, 2016.

[10] Henri, Yvon. The OneWeb Satellite System. In *Handbook of Small Satellites: Technology, Design, Manufacture, Applications, Economics and Regulation*, pages 1091–1100. Springer, 2020.

[11] Private Octopus Inc. Picoquic, 2024. `https://github.com/private-octopus/picoquic`.

[12] Iyengar, Jana and Thomson, Martin. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[13] Langley, Adam, Riddoch, Alistair, Wilk, Alyssa, Vicente, Antonio, Krasic, Charles, Zhang, Dan, Yang, Fan, Kouranov, Fedor, Swett, Ian, Iyengar, Janardhan, Bailey, Jeff, Dorfman, Jeremy, Roskind, Jim, Kulik, Joanna, Westin, Patrik, Tenneti, Raman, Shade, Robbie, Hamilton, Ryan, Vasiliev, Victor, Chang, Wan-Teh, and Shi, Zhongyi. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the conference of the ACM special interest group on data communication*, pages 183–196, 2017.

[14] Lantz, Bob, Heller, Brandon, and McKeown, Nick. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[15] Lattimore, Tor. Bandit Convex Optimisation. *arXiv preprint arXiv:2402.06535*, 2024.

[16] Lim, Yeon-sup, Nahum, Erich M, Towsley, Don, and Gibbens, Richard J. ECF: An MPTCP path scheduler to manage heterogeneous paths. In *Proceedings of the 13th international conference on emerging networking experiments and technologies*, pages 147–159, 2017.

[17] Linux. *tc-netem(8) — Linux manual page*, 2011. `https://man7.org/linux/man-pages/man8/tc-netem.8.html`.

[18] Liu, Sijia, Chen, Pin-Yu, Kailkhura, Bhavya, Zhang, Gaoyuan, Hero, Alfred O. III, and Varshney, Pramod K. A Primer on Zeroth-Order Optimization in Signal Processing and Machine Learning: Principals, Recent Advances, and Applications. *IEEE Signal Processing Magazine*, 37(5):43–54, 2020.

[19] McDowell, J.C. The Low Earth Orbit Satellite Population and Impacts of The SpaceX Starlink Constellation. *The Astrophysical Journal Letters*, 892(2):L36, 2020.

[20] Mdounin. Upstream: smooth weighted round-robin balancing., 2012. `https://github.com/phusion/nginx/commit/27e94984486058d73157038f7950a0a36ecc6e35`.

[21] Mohan, Nitinder, Ferguson, Andrew, Cech, Hendrik, Renatin, Prakita Rayyan, Bose, Rohan, Marina, Mahesh, and Ott, Jörg. A Multifaceted Look at Starlink Performance. *arXiv preprint arXiv:2310.09242*, 2023.

[22] Nemirovski, Arkadi. Interior Point Polynomial Time Methods in Convex Programming. *Lecture notes*, 42(16):3215–3224, 2004.

[23] OneWeb. Business aviation, 2024. `https://oneweb.net/solutions/aviation/business-aviation`.

[24] Raca, Darijo, Leahy, Dylan, Sreenan, Cormac J, and Quinlan, Jason J. Beyond Throughput, the Next Generation: A 5G Dataset with Channel and Context Metrics. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 303–308, 2020.

[25] Shu, Shengjie, Yang, Wenjun, Pan, Jianping, and Cai, Lin. A Multipath Extension to The QUIC Module for ns-3. In *Proceedings of the 2023 Workshop on ns-3*, pages 86–93, 2023.

[26] Thomson, Martin and Turner, Sean. Using TLS to secure QUIC. *RFC 9001*, 2021.

[27] Wischik, Damon, Handley, Mark, and Braun, Marcelo Bagnulo. The Resource Pooling Principle. *ACM SIGCOMM Computer Communication Review*, 38(5):47–52, 2008.

[28] Wu, Hongjia, Alay, Özgü, Brunstrom, Anna, Ferlin, Simone, and Caso, Giuseppe. Peekaboo: Learning-based Multipath Scheduling for Dynamic Heterogeneous Environments. *IEEE Journal on Selected Areas in Communications*, 38(10):2295–2310, 2020.

[29] Yang, Siyu, Li, Hewu, and Wu, Qian. Performance Analysis of QUIC Protocol in Integrated Satellites and Terrestrial Networks. In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1425–1430. IEEE, 2018.

[30] Zhao, Jinwei and Pan, Jianping. LENS: A LEO Satellite Network Measurement Dataset. In *Proceedings of the 15th ACM Multimedia Systems Conference*, pages 278–284, 2024.