FINE-GRAINED TRAFFIC MANAGEMENT IN COMPUTER NETWORKS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Ning Wu

December 2018

FINE-GRAINED TRAFFIC MANAGEMENT IN COMPUTER NETWORKS

Ning Wu, Ph.D.

Cornell University 2018

This dissertation investigates fine-grained traffic management in computer networks. Traffic management can effectively help networks to achieve higher Internet reliability, efficiency, and performance. As today't Internet traffic is growing rapidly and more dynamically, traffic management technologies that perform in fine grained become increasingly appealing because they manage and control network resources dynamically and adapt to the real-time traffic and condition changes. In this dissertation, we investigate fine-grained traffic management in three different dimensions: time, space, and application.

In the time dimension, we explore high-frequency traffic engineering. There exists an intrinsic performance tradeoff between responsiveness and stability for adaptive traffic engineering that performs in high frequency. We analyze it from a feedback control perspective, and derive a model that characterizes the system parameters' effects on the performance of the dynamic routing system. This allows quantitative analysis of adaptive TE algorithms and their design parameter choices. We then specialize the general framework in two representative network topologies and derive the stability conditions for their dynamic routing systems. Together they provide systematic insights on the relations among several network factors and the intrinsic tradeoff among different network control objectives.

In the space dimension, we investigate fine-granularity traffic split. For given traffic split ratios calculated mathematically by routing algorithms in the

routing engine, the routing realization mechanisms in the data plane implement such splits without breaking flows. Treating all flows equally, the state-of-the-art approaches deployed in switches do not provide enough accuracy especially when facing non-uniform flow size distribution. To accurately realize given traffic split ratios in switches with small performance degradation, we instead propose a dynamic load distribution scheme based on the collected load sharing statistics and incorporate such tradeoff in a weighted sum optimization. It finds the most accurate traffic splits with minimum route changes.

In the application dimension, we focus on per-application end-to-end path selection. The standard way to obtain end-to-end SLAs by creating private networks through business contracts is costly and takes lots of time to realize. We propose a platform that selects end-to-end path based on application specific performance need in real time through overlay networks. With the knowledge the network topology and conditions, it strives to achieve the optimal end-to-end performance by exploring the last-mile diversity. It allows the flexible and responsive per-application or per-end-user selection of the edge node for the overlay networks, and thus can fast recover from network failures and performance degradation. We present our design of the end-to-end throughput optimization system with detailed discussion of each component including dynamic routing engine, performance monitor and information exchange.

BIOGRAPHICAL SKETCH

Ning Wu earned her Bachelor of Science degree in Microelectronics from Fudan University in China in 2012. She received her Master of Engineering degree in System Engineering from Cornell University in 2013, then joined the doctoral degree program in Electrical and Computer Engineering in Cornell University in 2014. She is a member of the Networks Group, led by Prof. A. Kevin Tang. She was a recipient of Jacobs Fellowship and Holbrook Fellowship. Her research interests include control, optimization and management in computer networks.

To Zhengda, Zeyan and my parents.

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. A. Kevin Tang for his endless support and generous encouragement throughout my Ph.D. study. He has provided great guidance as well as much freedom in my research. Under his patient and effective supervision, I have developed independent thinking and learned how to conduct scientific research comprehensively. I am also grateful to the other members of my graduate committee, Prof. Ken Birman and Prof. Hakim Weatherspoon, for their invaluable advice that shaped my final dissertation and for their motivating words to me during my Ph.D study.

I am sincerely thankful to all my collaborators. I thank Nithin Michael, who offered me the opportunity to explore my interest in computer networks the first time. He always believed in me in every piece of work and inspired me at any time. I thank Nikolai Matni for his significant contributions to the collaboration of my first research project. I very much appreciated his efforts in iterating on the paper with me and his perseverance in research. To Chiun Lin Lim, Shih-Hao Tseng and Yingjie Bi, thanks for all the help and fruitful discussion on many projects. I am particular indebted to Ki Suh Lee and Andrey Gushchin for always being there. They provided me continued help, support and encouragement in every aspect of my study, work and life. To Han Wang and Archit Baweijia, thanks for enlightening me on computer systems and guiding me to analyze and solve problems from system perspective.

I would also like to thank the many friends, Meng Huang, Zihao Wang, Yin Feng, Cheng Fu, Shuang Liu and Steve Dai, who made my time at Cornell enjoyable and enriched. I would not forget all the fun times we spent together, the precious memories we had at Cornell, and the memorable trips we enjoyed throughout my Ph.D. years.

I cannot thank enough for my family for all their love. For my parents who raised me with enthusiasm of learning and supported me in all my pursuits. For my lovely son, Zeyan Lu, who was born in the last year of my Ph.D. study and brought us a great joy and motivation. And most of all for my loving husband, Zhengda Lu, whose endless love, faithful support and continued patience throughout the past ten years of my life made me the happiest and luckiest woman in the world. His always being with me has convinced me that companionship is the most everlasting confession of love. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Computer networks such as the Internet have been undergoing significant changes over the past decades. As the computer network technologies in hardware and software continuously evolve, starting with packet switching [56] and the ARPANET [40, 55], the horizons of the infrastructure have been expanded along several dimensions including performance, scale, and functionality.

At the same time, traffic over the Internet keeps growing rapidly in volume, pattern and diversity. Increasing nearly nine times from 400 million in 2000 to 3.5 billion users in 2017 [2], the annual global IP traffic will reach 3.3 ZB [1] by 2021 at a Compound Annual Growth Rate (CAGR) of 24 percent starting from 2016 according to a recent white paper by Cisco [8]. Aside from the increase in volume, network traffic is varying more dynamically, while it is known to cycle through periods of both high and low demand. Busy-hour Internet traffic is increasing 46 percent each year, more rapidly than the 32-percent growth in average Internet traffic [8]. Additionally, network traffic is fast growing in diversity. Today's Internet is so much more than just the home of email, static webpages and discussion boards. Video currently accounts for more than two-thirds of all Internet traffic in the world, and people accessing the Internet via a mobile device now outnumber those connecting from a computer [2]. Moreover, various applications come with their specific Quality of Service (QoS) metrics and user perception requirements. For example, web browsing applications require short response time, file transfer service expects high bandwidth, and video conferencing desires the real-time stream to have low latency and low jitter, etc.

[1] 1 ZB = 1000 Exabytes (EB)

The rapidly growing and more dynamic Internet traffic makes it more and more challenging for the Internet Service Providers (ISPs) to meet various Service Level Agreements (SLAs) that are offered to customers and business such as restrictions on latency, jitter, loss, etc. In the face of variable and unpredictable traffic demand, networks are often over-provisioned in order to accommodate a wide range of possibilities that include both normal and worst-case conditions. Naturally, this conservative approach minimizes the work of reactively managing the traffic and network resource. However, it results in low average utilization with many network links typically running at around 30% capacity [41], which is not only costly but also unsustainable. As network operators strive to lower cost and stay competitive, they are passionately seeking for more efficient approaches that can be responsive to the fast growth and dynamic of the traffic. With the introduction of Software Defined Networking (SDN) and its related technologies that enable the programmability of network devices and flexibility of technique deployment, fine-grained traffic management and network resource control solutions which adapt to the real-time traffic and network changes become increasingly appealing [48, 42, 41, 60, 69, 70]. The scope of this dissertation is to investigate the following research questions: to which extent can we exploit traffic management in fine-grained manner, and how much potential benefit can we obtain from that?

In this dissertation, we focus on fine-grained traffic management in three different dimensions: time, space, and application. More explicitly, we explore how to manage traffic routing in high frequency, how to manage traffic split in fine granularity, and how to manage end-to-end path selection per-application. In each dimension, we quantitatively study the benefit and cost of performing fine-grained traffic management, develop algorithms and architectures that re-

alize the vision, and implement the solutions to emulators or real networks for evaluation.

1.1 Traffic Management: A Control System

In today's computer network, ISPs strive to achieve a variety of goals in network provisioning, operation and management: the network is expected to be available at all times, to be efficient in resource usage, to be able to expand to large scale, to maintain low cost, to ensure different SLAs that are provided to customers and business, etc. However, as the Internet traffic is growing rapidly and more dynamically, it is nontrivial for the ISPs to meet these and other requirements. Besides, the known problems of today's Internet including sudden outages and unpredictable peering relationship changes [16, 12, 5, 1] make the tasks even more challenging without dynamically managing the traffic.

Traffic management technologies could potentially help networks to achieve the goals of offering higher Internet reliability, efficiency, flexibility and performance. They are widely used to monitor, test, analyze and control the network and element resources, in the deployment, integration, and coordination of the hardware, software and human elements. Despite varying in functionalities and realization methods, traffic management in general controls the network resources reactively by making adjustments in response to the changes within the network. They target at meeting some specific requirements at a reasonable cost.

Generally speaking, any real-time traffic management can be viewed as a feedback control system. Therefore, fundamental knowledge and rich research

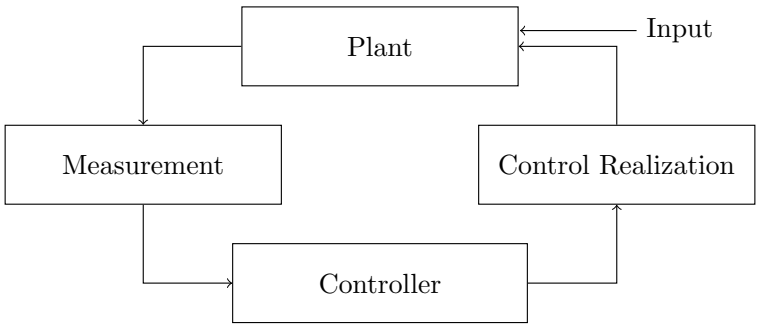


Figure 1.1: Block diagram of a closed-loop control system.

work in dynamic systems and control theory would facilitate our in-depth understanding, systematic design and performance analysis of traffic management. Figure 1.1 shows the block diagram of a classic closed-loop control system, as known as a feedback control system. The control loops that make up control systems are made themselves of measurement, controller, and control realization. In a feedback control system, the controller is responsible for making the plant behave in a desired manner by controlling given variables that would effectively affect the plant. The output of the plant is measured by some measurement mechanism, and the measured information is fed back to the controller. The controller computes the desired values for control variables based on specific algorithm or logic, and forwards them to the control realization component to actually adjust the controlled variables which influence the plant's behavior.

Real-time traffic management is essentially a feedback control system (as is illustrated in Figure 1.2): the network itself is the plant which evolves with varying traffic demand input, as well as the control variable input being adjusted by the control realization mechanism; periodically some specific network states are measured and fed back to the controller; the traffic management engine as the

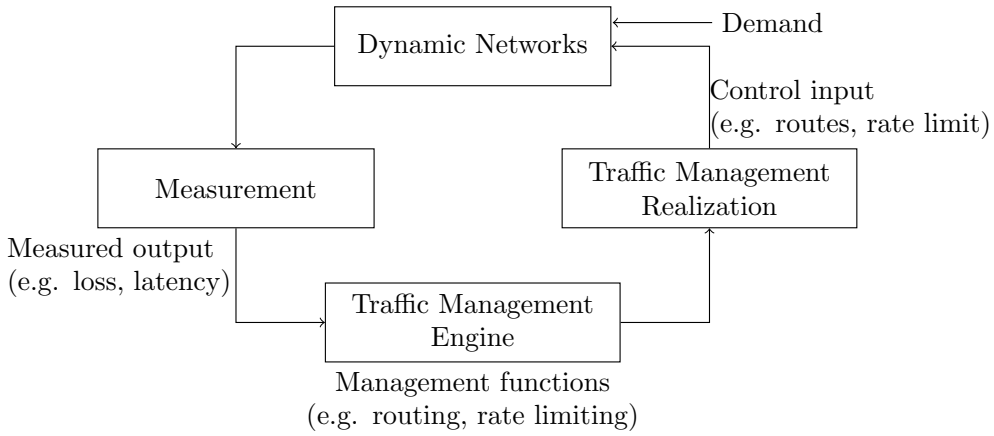


Figure 1.2: Block diagram of traffic management for computer networks.

controller is usually designed for particular traffic management functions (e.g. routing, congestion control, rate limiting, etc.) and it computes the desired values of the control variables in either distributed or centralized fashion; based on the computed values, the traffic management realization mechanism adjusts the actual variables being controlled to affect the network. Taking traffic engineering (TE), one of the main traffic management technologies, as an example. The measurement output could be any network state of TE's interest such as packet loss or latency. With the measured information obtained, the traffic management engine computes the TE routing solutions based on the TE objectives. Then the new routing strategies are configured by the TE realization component to control the routing in the network. In this way, the feedback control loop is formed and the system keeps repeating the process at a certain rate to make the network behave in a desired manner.

The performance of a feedback control system is largely affected by various factors from different dimensions. For example, the frequency that feedback loop takes place determines how fast the plant is adjusted and how long it takes

for the system to stabilize. The granularity of the control variables defined in the control system influences how much the plant is controlled and to which level the plant can be actuated. Given the desired value of the control variables computed by the controller, the accuracy that the actual variables can be adjusted by the control realization mechanism affects how far the plant deviates from the ideal behavior. In this dissertation, we investigate traffic management in fine-grained manner from three different dimensions. In temporal dimension, we focus on the high frequency of routing update that performs in the traffic management engine block. In spatial dimension, we explore the fine granularity of traffic split that operates in the traffic management realization block. In application dimension, we define per-application control variables and develop end-to-end path selection solutions based on application-specific performance metrics.

1.2 Challenges for Fine-grained Traffic Management

The challenges for fine-grained traffic management come from either the fundamental limitation of critical practical factors or the tradeoff between different requirements. In this section, we describe the challenges for managing traffic routing in high frequency, managing traffic split in fine granularity, and managing end-to-end path selection based on application-specific needs.

1.2.1 Temporal Dimension: Traffic Engineering in High Frequency

Traffic Engineering (TE) is of great importance for network management and optimization and has attracted much attention over the years, see [35, 34, 73, 67, 68, 22, 31, 21, 49] for a non-exhaustive example list. TE methods effectively map the traffic demand onto the network topology in order to optimize the use of network resources in computer networks. Unlike the static TE approach which computes and configures the optimal routing at one time, adaptive TE solutions dynamically adjust the routing decisions to gradually change the proportion of traffic on each path according to the network loads and conditions. In recent years, the rise of SDN [54] makes it more feasible to change routing frequently, and thus motivates the adaptive TE that operates at short timescale [17, 18]. Fast adaptive TE solutions [31, 44, 33, 42, 19, 57] become increasingly attractive because of their potential ability to react fast to the time-varying traffic demand as well as network failures and better utilize the network resources.

Adaptive TE techniques introduce an intrinsic performance tradeoff between responsiveness and stability when performing in high frequency. On one hand, adaptive TE is desired to update routing solutions as quickly as possible so that it can react to any network changes and traffic fluctuations in real-time; on the other hand, we wish to prevent routing oscillation and guarantee the stability of the network all the time. Intuitively, being responsive benefits from frequent update, whereas maintaining stability requires cautious steps. Too fast routing update makes the system too sensitive to the noise and disturbance which may result in over-reacting and routing oscillation. Update at low frequency, however, slows down the convergence speed in response to demand

changes, especially when a demand peak suddenly occurs which may potentially lead to packet loss if the routing update is not able to react in time.

Therefore, the challenge for adaptive TE in high frequency is mainly about how to strike the tradeoff between stability and responsiveness. In the real networks, the stability and responsiveness are affected by multiple factors, from design parameters in the control system to engineering facts such as propagation delay, measurement noise. Moreover, a systematic understanding of how the factors interact with each other is nontrivial. As a result, it is challenging to quantitatively analyze the impacts of the system parameters on the performance metrics for a given adaptive TE solution, let alone to best strike the performance tradeoff. These days, lacking of the comprehensive insights, the network designers and operators choose the tuning parameters involved empirically based on massive experiments [44, 19].

In this dissertation, we derive a framework for the analysis of adaptive TE solutions. We form a feedback control model that characterizes critical parameters and engineering factors. By properly modeling the feedback system, we are able to analyze the stability constraints of the steady state, as well as the dynamic response of transient behavior. We investigate the intrinsic interactions of the critical parameters and provide insightful observations of their effects on the network performance, which can be used to seek for the best parameter choices so as to optimize the performance of responsiveness while guaranteeing the stability of the system.

1.2.2 Spatial Dimension: Traffic Split in Fine Granularity

Traffic management that aims at managing routing typically works in the following fashion, as we have shown in Figure 1.2. Based on demand information, measurement outputs of the network state and other operational constraints, routes are computed periodically by the routing engine in either centralized or distributed manner according to certain routing algorithms. The solutions are then realized in the management realization component, usually on the data plane by properly updating routing tables in routers and switches. Among all kinds of routing techniques, multipath routing strategies that allocate traffic among multiple paths are widely used to provide better load balancing across the network. The multipath routing solution usually defines how to split the traffic among the available paths in the form of a set of split ratios, each of which is associated with one path or outgoing interface. Once the optimal solution is determined, the routing tables are updated to realize the corresponding desired split ratios.

Multipath routing algorithms normally solve the mathematical optimization problems assuming traffic can be split arbitrarily. However, in practice, individual flows are not split over multiple paths in order to avoid potential out-of-order arrivals which can cause significant performance degradation such as TCP goodput drop. Therefore, an important question raises in the above described picture: how well can a router realize the target traffic split ratios? Treating all flows equally, the state-of-the-art approaches deployed in routers do not provide enough accuracy especially when facing non-uniform flow size distribution. As traffic demand rapidly grows in both volume and variation, such inaccurate load distribution deviating from desired routing solutions can result

in negative effects such as network performance degradation due to imbalanced resource utilization or even link capacity violation.

To cope with the inaccurate traffic split problem, load-aware schemes have been proposed [45, 27, 52] that collect the traffic load information and dynamically adjust the existing paths assignment in real time to make the actual split ratios close to the target ones. Such schemes require extreme caution because path adjustment requires traffic migration to a different path, which could potentially lead to packet reordering when those two paths have different latency. In fact, in the design of a load distribution mechanism, the requirements of realizing accurate split ratios and producing minimum flow path update are of competing nature since updating to new traffic splits more precisely usually needs to shuffle more flows.

Therefore, we investigate the following research problem in this dissertation: how to achieve high accuracy of split ratios realization while the existing path assignment remains unchanged as much as possible? We incorporate such tradeoff in a weighted sum optimization. We develop a dynamic load distribution scheme based on the collected load sharing statistics. It finds the most accurate traffic splits with minimum route changes.

1.2.3 Application Dimension: End-to-end Path Selection in Fine Application

Although ISPs offer various SLAs to guarantee the average Internet performance in long term, most individual users and enterprises still suffer from

performance degradation from time to time such as low speed, large delay and high packet loss rate. In fact, managing the end-to-end user experience has been a non-trivial and highly complex issue in the Internet, for two reasons. First, the current Internet is an aggregation of a large number of networks owned by many ISPs with different economic interests [28]. The standard way to obtain end-to-end SLAs is to create private networks through business contracts among them. This obviously is costly and takes lots of time to realize. Second, the management and configuration complexity of Internet hardware routers produced by major vendors is substantial. The flexibility, availability and cost efficiency is increasingly limited by those hardware routers. The prevailing routing protocols being used in today's Internet such as the Border Gateway Protocol (BGP) [62] are known to have limitations in realizing any performance-aware dynamic routing solution. BGP routing neither incorporates the information of path performance or link capacity, nor allows fine-grained overriding of BGP-specified forwarding behavior [24]. With these two severe constraints, network providers and operators face great difficulties in realizing reliable, high-performance end-to-end path management, let alone providing good per-application user experience.

In face of the above mentioned challenges in the Internet infrastructure, overlay network architecture, which has been designed and developed for different purposes since a few decades ago [20, 39, 32], can be adopted as an enabler of achieving flexible control of routing. It places a virtual network over the physical infrastructure, thus leveraging the dynamic control of network resources on an abstraction layer. Although there exist various overlay-based techniques of application-aware routing on fast timescale [74, 63], they focus on controlling the routes inside the overlay network whereas the performance

of the upstream and downstream path in the last mile for the end users is beyond their control. In the case where the communication between the end user and the edge node is through the public Internet, the end-to-end performance can still be largely affected by any unpredicted delay and congestion over the Internet. Conceptually, if there exists diversity and flexibility in selecting the edge node, it is possible to achieve more reliable end-to-end performance.

Therefore, we investigate the following research question in the dissertation: can we explore the diversity of the last mile and flexibly manage per-application end-to-end path? In this dissertation, we take an alternative approach and propose the design of a platform that allows the exploration of the geographical and physical diversity in the last-mile. We develop a path selection scheme which makes the end-to-end path decisions dynamically for the specific applications with consideration of the jointly real-time traffic conditions in both the core network and the last-miles.

1.3 Contributions

We have introduced the challenges for achieving fine-grained traffic management in different aspects. In this dissertation, we explore fine-grained traffic management in time, space and application dimensions. In each dimension, we analyze the benefit and cost of performing fine-grained traffic management, and develop algorithms and architectures that realize the vision. In particular, we present the following three contributions: high-frequency traffic engineering, fine-granularity traffic split and per-application end-to-end path selection.

High-frequency traffic engineering

There exists an intrinsic performance tradeoff between responsiveness and stability for adaptive TE that performs in high frequency. We analyze it from a feedback control perspective, and derive a model that characterizes the system parameters' effects on the performance of the dynamic routing system. This allows quantitative analysis of adaptive TE algorithms and their design parameter choices. We then specialize the general framework in two representative network topologies and derive the stability conditions for their dynamic routing systems. Together they provide systematic insights on the relations among several network factors and the intrinsic tradeoff among different network control objectives.

Fine-granularity traffic split

For given traffic split ratios calculated mathematically by routing algorithms in the routing engine, the routing realization mechanisms in the data plane implement such splits without breaking flows. Treating all flows equally, the state-of-the-art approaches deployed in switches do not provide enough accuracy especially when facing non-uniform flow size distribution. To accurately realize given traffic split ratios in switches with small performance degradation, we instead propose a dynamic load distribution scheme based on the collected load sharing statistics and incorporate such tradeoff in a weighted sum optimization. It finds the most accurate traffic splits with minimum route changes.

Per-application end-to-end path selection

The standard way to obtain end-to-end SLAs for enterprises branch-to-branch Internet service by creating private networks through business contracts is costly and takes lots of time to realize. We propose a platform that selects

end-to-end path based on application specific performance need in real time through overlay networks. With the knowledge the network topology and conditions, it strives to achieve the optimal end-to-end performance by exploring the last-mile diversity. It allows the flexible and responsive per-application or per-end-user selection of the edge node for the overlay networks, and thus can fast recover from network failures and performance degradation. We present our design of the end-to-end performance optimization system with detailed discussion of each component including dynamic routing engine, performance monitor and information exchange.

CHAPTER 2

TEMPORAL MANAGEMENT: HIGH-FREQUENCY ADAPTIVE TRAFFIC ENGINEERING

2.1 Introduction

Traffic Engineering (TE) methods effectively map the traffic demand onto the network topology in order to optimize the use of network resources in IP networks. Static TE approaches compute the optimal routing solutions off-line for a given traffic demand matrix. They fail to react to unpredicted traffic and network condition changes if operating on a large timescale, but updating too frequently will lead to routing oscillations. To overcome these challenges, adaptive TE solutions have been proposed. Instead of computing and configuring the optimal routing at one time, they dynamically adjust the routing decisions to gradually change the proportion of traffic on each path according to the network loads and conditions [31, 44, 33, 42, 19, 57]. In recent years, SDN [54] makes it more feasible to change routing frequently, and thus motivates the adaptive TE that operates at short timescale [17, 18].

Any fast adaptive TE technique introduces more than one system parameters into their design. This is mainly due to the consideration of the intrinsic performance tradeoff between stability and responsiveness. On one hand, adaptive TE is expected to react to any network changes and traffic fluctuations as quickly as possible; on the other hand, it is supposed to keep the system stabilized in the steady state to prevent routing oscillation. In the real networks, the stability and responsiveness are affected by multiple factors, from design parameters in the control laws to engineering factors such as propagation delay, measurement

noise. Moreover, a systematic understanding of how the factors interact with each other is nontrivial. Therefore, it is challenging to quantitatively analyze the impacts of the system parameters on the performance metrics for a given adaptive TE solution, let alone to best strike the performance tradeoff.

In this chapter we provide a framework for analyzing the performance of any given adaptive TE solutions. We take a comprehensive approach and form a model that characterizes three critical parameters: (i) the step size in the TE control law which determines how much to move along the descending direction at each iteration; (ii) the update interval of the routing computation which defines how frequently the routing strategy changes; and (iii) the physical propagation delays on the network. Being reactive benefits from large step size and frequent update, whereas maintaining stability requires cautious steps and small system noise which is sensitive at high frequency. We analyze the intrinsic interactions of these parameters in two representative case studies. Experiments from Mininet [50] emulations are carried out to further provide insightful observations of their effects on the network performance.

2.2 Model

In this section, we propose the model that is used to analyze the fast adaptive TE system. In general, any real-time adaptive TE solutions can be viewed as a feedback control system (as is illustrated in Figure 2.1): the network itself is the plant which evolves with varying traffic demand input $d(t)$ as well as the routing strategy $u(t)$ being updated by some routing engine as the controller; periodically the metrics of the network state $x(t)$ is measured and fed back to

the routing engine; the routing engine then computes the TE routing solutions based on the state information obtained and configures the new routing strategy to the network.

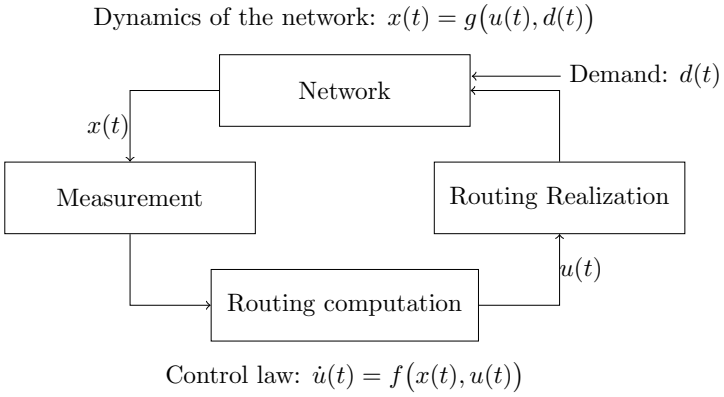


Figure 2.1: Adaptive TE control diagram.

Suppose a network consists of a set of users (pairs of ingress-egress nodes) $\mathcal{I}$, and a set of links $\mathcal{L}$. Each link l has a capacity c_l. We assume the packets are routed based on source routing, meaning that the ingress router of a packet determines the specific path for the packet among all pre-defined paths. Each user $i \in \mathcal{I}$ has demand d_i to send from source to destination via a set of paths $\mathcal{P}$ where a subset $\mathcal{P}_i \subseteq \mathcal{P}$ consists of all feasible paths that belong to user i. Each path $k \in \mathcal{P}$ is composed by a set of directed links, denoted as $\mathcal{L}_k$. Suppose the number of paths and links in the network is K and L respectively. We further denote an $L \times K$ matrix $R_0 = (r_{lk})$ to represent the link composition of each path such that

$$r_{lk} = \begin{cases} 1, & \text{if } l \in \mathcal{L}_k, \\ 0, & \text{otherwise.} \end{cases}$$

For the traffic belonging to user i, the routing engine deployed in its ingress router decides on how much percentage of d_i is routed to each path $k \in \mathcal{P}_i$. The routing decisions are made according to the path metrics and the control law

defined by the adaptive TE algorithm. Let the state vector be $x \in \mathbb{R}^{K\times 1}$ where the component x_k represents the path metric for path k. The control vector is denoted as $u \in \mathbb{R}^{K\times 1}$ where the component u_k is the split ratio for path k. The split ratios of paths for user i satisfy that $\sum_k u_k(t) = 1, \forall k \in \mathcal{P}_i$. The control law for adaptive TE follows some function $f(\cdot)$ such that

$$\dot{u}(t) = f\big(x(t), u(t)\big).$$

The path metrics can be defined as, for example, the aggregated utilization of all links on the path, the maximum link utilization on the path, or the total latency along the path, etc, depending on the performance objectives of TE. The performance of a link is affected by the aggregated traffic from different user's paths that share the same link, which in turn determines the path metrics. Thus the path metrics can be expressed as a function $g(\cdot)$ of the split ratios for each path and the current demand denoted as $d(t)$:

$$x(t) = g\big(u(t), d(t)\big).$$

We consider the path metric to be the maximum utilization of all the links along a path, which is a widely used metric in TE algorithms with the objectives of load balancing or avoiding congestion. In the delay-free case, the utilization of link l at time t, denoted as $w_l(t)$, is given by

$$w_l(t) = \frac{\sum_{j:l\in\mathcal{L}_j} u_j(t) d_i(t)}{c_l}, \quad i : j \in \mathcal{P}_i,$$

where user i is the owner of path j. So the metric for path k is given by

$$x_k(t) = \max_{l\in\mathcal{L}_k} \frac{w_l(t)}{c_l}.$$

We further consider a specific control law as follows

$$\dot{u}_k(t) = \alpha\bigg(\frac{\sum_{j\in\mathcal{P}_i} x_j(t)}{K_i} - x_k(t)\bigg), \forall k \in \mathcal{P}_i, \tag{2.1}$$

where α is the step size as a tunable parameter, and K_i is the number of paths that belong to user i. It is easy to justify that Eq 2.1 guarantees the split ratios of paths for an user add up to 1 with the satisfied initial set of split ratios:

$$\sum_{k \in \mathcal{P}_i} \dot{u}_k(t) = \alpha \sum_{j \in \mathcal{P}_i} x_j(t) - \alpha \sum_{k \in \mathcal{P}_i} x_k(t) = 0.$$

To express the system in matrix form, let $P_0(t) = (p_{k,l}(t))$ be the $K \times L$ matrix of the path metric function, with $p_{k,l}(t) = 1$ if utilization of link l is the maximum along path k, and 0 otherwise. Furthermore, let $D_i(t) \in \mathbb{R}^{K_i \times K_i}$ be the $K_i \times K_i$ diagonal matrix of demand for user i's paths, i.e. $D_i(t) := diag(d_i(t))$, and $D(t) \in \mathbb{R}^{K \times K}$ be the $K \times K$ diagonal matrix of demand for all users' paths, i.e. $D(t) := diag(D_i(t))$. Let $M_i = (m^i_{j,k})$ denote the $K_i \times K_i$ symmetric matrix for user i such that

$$m^i_{j,k} = \begin{cases} 1 - \frac{1}{K_i}, & \text{if } j = k, \\ -\frac{1}{K_i}, & \text{otherwise.} \end{cases}$$

Finally, let $M := diag(M_i) \in \mathbb{R}^{K \times K}$, $C := diag(\frac{1}{c_l}) \in \mathbb{R}^{L \times L}$. Then the state space equations are in the following form:

$$\dot{u}(t) = -\alpha M x(t),$$

$$x(t) = P_0(t) C R_0 D(t) u(t).$$

To compute the path metrics, the ingress node of each user need to collect the information of link states by periodically sending the probing packets through all its paths. When the probing packet travels through each node along the path, the router will push the link states (the link utilization in our case) information in the probing packet. The destination router of the path then sends it back to the source router once receiving the probing packet. In real network, it takes a round-trip time (RTT) for the probing packet to come back, and thus we here

incorporate delay into the model. Let δ_k^l denote the propagation delay from the source of path k to link l, and let β_k^l denote the backward delay from link l back to the source of path k for probing packets. The utilization of link l at time t is

$$w_l(t) = \frac{\sum_{j:l\in\mathcal{L}_j} u_j(t-\delta_j^l)d_i(t-\delta_j^l)}{c_l}. \tag{2.2}$$

The path metric for any path $k \in \mathcal{P}$ at time t is determined by the feedback link utilization collected from each link, i.e.

$$x_k(t) = \max_{l\in\mathcal{L}_k} w_l(t-\beta_k^l), \quad \forall k \in \mathcal{P}. \tag{2.3}$$

2.3 Analysis

In this section, we analyze the stability of the model, and specialize the general methodology in two representative network topologies [31], single bottleneck link and shared links.

2.3.1 Stability analysis

One common feature for any adaptive TE method is that it can only be implemented in discrete time. This fact imposes an important factor into the practical system that the continuous-time model does not capture: the update interval of routing computation. In the following, we discretize the system and study the effect of step size, update interval and system delay, as well as their internal relations.

Let τ be the update interval of the routing computation that defines how frequently the routing strategy changes. We assume the link utilization values be-

ing collected at each router is measured during the period of τ. In Section 2.3.2, we deeply investigate the internal relations of update interval τ and round-trip time T, and derive the stability condition for arbitrary value of τ/T in the case of single user with single bottleneck link. However, for the general network, we simplify the discrete-time model by assuming that $\beta_k^l = n_{\beta,k}^l \tau$ and $\delta_j^l = n_{\delta,j}^l \tau$, where $n_{\beta,k}^l$ and $n_{\delta,j}^l$ are integers. We assume the synchronization of all ingress nodes for each update at each time stamp $n\tau, n = 0, 1, 2, \cdots$. Furthermore, because we are interested in the question whether a given fast adaptive TE can stabilize and how fast it converges in the presence of any level of demand, we focus on the stability analysis in the steady state such that the demand matrix $D(t)$ is not changed within the timescale of our interest, i.e. $d_i(t) = d_i, \forall t$.

Discretizing Eq. 2.2 and Eq. 2.3, we have

$$w_l(n) = \frac{\sum_{j:l\in\mathcal{L}_j} u_j(n - n_{\delta,j}^l)d_i}{c_l}, \tag{2.4}$$

$$x_k(n+1) = \max_{l\in\mathcal{L}_k} w_l(n - n_{\beta,k}^l), \quad \forall k \in \mathcal{P}. \tag{2.5}$$

Taking the z transform of Eq. 2.4 and Eq. 2.5 yields

$$x_k(z) = \max_{l\in\mathcal{L}_k} \sum_{j:l\in\mathcal{L}_j} z^{-(n_{\delta,j}^l + n_{\beta,k}^l)} u_j(z)\frac{d_i}{c_l}, \quad i : j \in \mathcal{P}_i.$$

The control law follows from discretizing Eq. 2.1:

$$u_k(n) = u_k(n-1) + \alpha\left(\frac{\sum_{j\in\mathcal{P}_i} x_j(n)}{K_i} - x_k(n)\right), \forall k \in \mathcal{P}_i.$$

Thus the discrete-time model with feedback delays in frequency domain is given by

$$u(z) = z^{-1}u(z) - \alpha M x(z), \tag{2.6}$$

$$zx(z) - zx(0) - x(1) = P(z)CR(z)Du(z), \tag{2.7}$$

where the matrices $R(z)$ and $P(z)$ in frequency domain are written as:

$$(R(z))_{jl} := \begin{cases} z^{-n^l_{\delta,j}}, & \text{if the path } j \text{ uses link } l, \\ 0, & \text{otherwise,} \end{cases}$$

$$(P(z))_{kl} := \begin{cases} z^{-n^l_{\beta,k}}, & \text{if } l \text{ is the bottleneck link for path } k, \\ 0, & \text{otherwise.} \end{cases}$$

The stability condition can be developed based on basic control theory. The system is stable if the poles of the closed-loop transfer function have magnitude less than 1, i.e. the roots of $det((z-1)(I+G(z))) = 0$ have magnitude less than 1, where

$$G(z) = \frac{\alpha M P(z) C R(z) D}{z-1}.$$

Intuitively, the larger step size α and the smaller update interval τ make the system more responsive. However, the stability condition is also dependent of the value of α, as well as the relation of τ and delays. As a result, it is critical to determine the region of the parameter choices under the constraint of the stability condition so that the stability is guaranteed when seeking for the best solutions of achieving responsiveness.

2.3.2 Case study I: single bottleneck link

In the first case study, we consider a topology consisting of a single pair of ingress-egress nodes with two paths as shown in Figure 4.3. The demand from ingress node $I0$ to egress node $E0$ is D. We assume all links are identical with the same capacity C and the same forwarding and backward delays. Suppose

the link between the source host and ingress node $I0$ is also restricted by capacity C, then we have $D \in [0, C]$. We further assume initially all traffic are routed through the upper path $l_1 \to l_2$, which we denote as path 1 and thus path 2 is the lower path. Either l_1 or l_2 can be viewed as the bottleneck link. In our following discussion, l_2 is assumed as the bottleneck link before the load on the two paths are balanced.

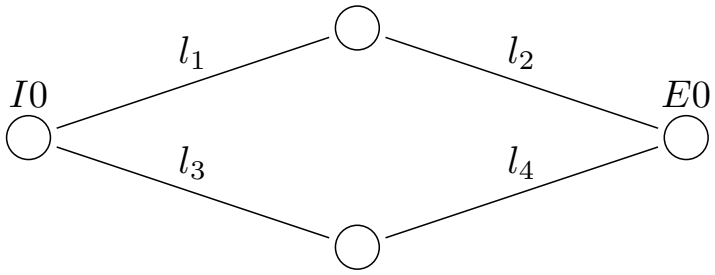


Figure 2.2: Single user with two paths.

Delay-free

In the ideal case, the propagation delay and backward delay are negligible. In the discrete-time model, we still use α for the step size, while it is worthwhile noting that the step size α in continuous-time model is equivalent to $\alpha\tau$ when we discretize the system. Considering the fact that $u_1(n) + u_2(n) = 1$, the system can be simplified as

$$x(n+1) = \frac{D(n)u(n)}{C},$$
$$u(n) = u(n-1) + \alpha(\frac{D(n)}{2C} - x(n)),$$

where we let $x(n)$ and $u(n)$ be the state and control variable for path 1. For a given demand $D \in [0, C]$, the state evolution is given by

$$x(n+1) = (1 - \alpha\frac{D}{C})x(n) + \frac{\alpha}{2}(\frac{D}{C})^2.$$

To ensure that the closed-loop system is stable, the following condition need to hold:

$$|1 - \alpha\frac{D}{C}| < 1, \forall\frac{D}{C} \in [0, 1].$$

When $0 < \alpha < 2$, the system can be stablized and will converge to the state $\bar{x} = D/2C, \quad \bar{u} = 1/2$. In the single bottleneck link case, when the delay is negligible, the stability constraint for the system is independent of any parameters other than the step size.

With delay

We now consider the delay of the closed-loop system. When the link propagation delay is not negligible, one significant variation from the ideal model is the fact that once the control decisions (i.e. the new split ratios) are updated at the source node, the state variables (the link utilization) in the downstream nodes do not change with the new update instantaneously. In other words, the link utilization measured by the nodes at each step within the measurement period τ may be affected by both the current and historical control variables. This fact is not reflected by the previous analysis of delay-free case. Furthermore, since any adaptive TE solution is implemented in discrete time, the effect of propagation delay on the system also depends on the update interval.

Let δ be the identical propagation delay from ingress node to link l_2. Let β be the identical backward delay that takes for the ingress node to receive the latest state information from link l_2. The instantaneous link utilization at time t for

link l_1 and l_2 are given by

$$w_1(t) = \frac{u(t)D(t)}{C},$$
$$w_2(t) = w_1(t-\delta) = \frac{u(t-\delta)D(t-\delta)}{C}.$$

Since we assume the measurement time interval for the link utilization is τ, the actual link utilization of l_2 that is fed back to the ingress node is the averaged value $\bar{w}_2(t)$ over the past time period τ, i.e.

$$\bar{w}_2(t) = \frac{1}{\tau}\int_{t-\tau}^{t} w_2(\eta)d\eta = \frac{1}{\tau}\int_{t-\tau-\delta}^{t-\delta} u(\eta)\frac{D(\eta)}{C}d\eta.$$

To implement the control law in discrete time, we sample the state variable and compute the control law every τ, starting from $t = 0$. Once the control law is updated, the control variable $u(t)$ keeps unchanged within time interval τ until the next update is triggered, i.e.

$$u(t) = u(n\tau), \quad n = \lfloor\frac{t}{\tau}\rfloor.$$

As the bottleneck link, l_2's actual measured link utilization $\bar{w}_2$ will be the path metric. Since it takes β for the feedback message to travel back to the routing engine, the path metric at time $n\tau$ is $\bar{w}_2(n\tau-\beta)$, $n = 0, 1, 2, \cdots$, is given by

$$\begin{aligned}
x(n\tau) &= \bar{w}_2(n\tau-\beta) \\
&= \frac{1}{\tau}\int_{(n-1)\tau-\delta-\beta}^{n\tau-\delta-\beta} u(\eta)\frac{D(\eta)}{C}d\eta \\
&= \frac{1}{\tau}u((n-2-\lfloor\frac{T}{\tau}\rfloor)\tau)\int_{(n-1)\tau-T}^{(n-1)\tau-\lfloor\frac{T}{\tau}\rfloor\tau}\frac{D(\eta)}{C}d\eta \\
&+ \frac{1}{\tau}u((n-1-\lfloor\frac{T}{\tau}\rfloor)\tau)\int_{(n-1)\tau-\lfloor\frac{T}{\tau}\rfloor\tau}^{n\tau-T}\frac{D(\eta)}{C}d\eta,
\end{aligned}$$

where $T = \delta + \beta$ is the RTT of the path.

In the steady state where $D(t)$ is a constant value D, we derive the discrete time state space equations with feedback delays and update interval:

$$x(n) = \frac{D}{C}[(k - \lfloor k \rfloor)u(n - 2 - \lfloor k \rfloor) + (1 + \lfloor k \rfloor - k)u(n - 1 - \lfloor k \rfloor)], \tag{2.8}$$

$$u(n) = u(n-1) + \alpha(\frac{D}{2C} - x(n)), \quad k = \frac{T}{\tau}. \tag{2.9}$$

Figure 2.3a shows the block diagram, where we define

$$H(z) = \frac{\alpha z}{z-1},$$
$$G(z) = \frac{D}{C}\left(\frac{k - \lfloor k \rfloor}{z^{2+\lfloor k \rfloor}} + \frac{1 + \lfloor k \rfloor - k}{z^{1+\lfloor k \rfloor}}\right), R = \frac{D}{2C}.$$

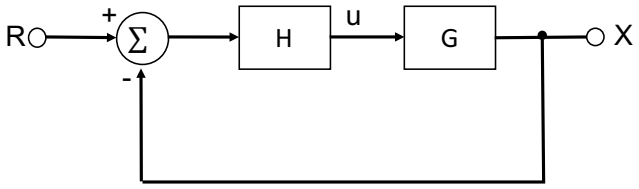
The transfer function of the closed-loop system follows that

$$Q(z) = \frac{H(z)G(z)}{1 + H(z)G(z)}.$$

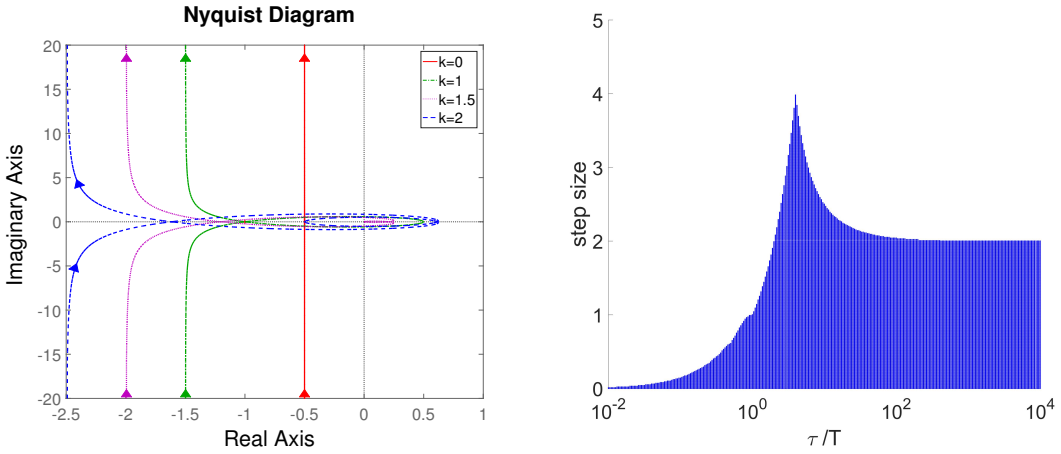
To ensure that the system is stable, all the roots of the following equation should have magnitude less than 1:

$$z^{2+\lfloor k \rfloor} - z^{1+\lfloor k \rfloor} + \alpha\frac{D}{C}(1 + \lfloor k \rfloor - k)z + \alpha\frac{D}{C}(k - \lfloor k \rfloor) = 0.$$

Figure 2.3b shows the Nyquist plot of $G(z)H(z)$ for some values of τ/T. Figure 2.3c shows the range of α as the value of τ/T varies when $D/C \in [0, 1]$. One way to understand why the stability region of step size changes in this form is to consider the effect of averaging state variables with different levels of historical states. When τ is much larger than T meaning that propagation delay is negligible, $x(n)$ depends on $u(n-1)$ only, so the region of α is independent of τ. This is also consistent with the result we have derived in the ideal delay-free case. With a certain level of averaging, the state variables are smoothed, thus enlarging the range of α. When $\tau = T$, the state variables obtained are out dated relying completely on historical data $u(n-2)$. As T becomes much larger than τ, i.e. τ/T becomes much smaller than 1, the system relies on older data. Therefore, it is more likely to oscillate and the stability region for α is narrower.



(a) Feedback control block diagram



(b) Nyquist plot of $G(z)H(z)$ for different τ/T. (c) Stability region of step size when τ/T varies.

Figure 2.3: Stability analysis.

2.3.3 Case study II: shared links

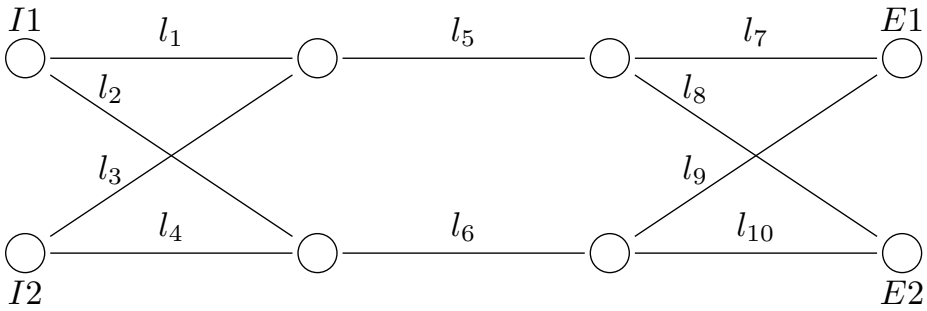


Figure 2.4: Multiple users with shared links.

In the following section, we study the case in which links are shared by multiple users. Consider the network topology shown in Figure 2.4, there are 4 users, namely $I1-E1$, $I1-E2$, $I2-E1$, $I2-E2$. Each user has two paths. The

bottleneck links are l_5 and l_6 which are the common links for four paths. $x(n)$ and $u(n)$ are 8×1 vectors. The state space equations of the system are described in Eq. 2.6 and Eq. 2.7, where the matrix M and $P(z)$ are specifically in the following forms:

$$M = \frac{1}{2}\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix},$$

$$P(z) = \begin{bmatrix} 0 & 0 & 0 & 0 & z^{-n^5_{\beta,1}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & z^{-n^6_{\beta,2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & z^{-n^5_{\beta,3}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{-n^6_{\beta,4}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & z^{-n^5_{\beta,5}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & e^{-n^6_{\beta,6}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & z^{-n^5_{\beta,7}} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & z^{-n^6_{\beta,8}} & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The 1th to 4th column of $R(z)$, as an example due to the space limitation, is

$$R_{j,1-4}(z) = \begin{bmatrix} z^{-n^1_{\delta,1}} & 0 & z^{-n^1_{\delta,3}} & 0 \\ 0 & z^{-n^2_{\delta,2}} & 0 & z^{-n^2_{\delta,4}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ z^{-n^5_{\delta,1}} & 0 & z^{-n^5_{\delta,3}} & 0 \\ 0 & z^{-n^6_{\delta,2}} & 0 & z^{-n^6_{\delta,4}} \\ z^{-n^7_{\delta,1}} & 0 & 0 & 0 \\ 0 & 0 & z^{-n^8_{\delta,3}} & 0 \\ 0 & z^{-n^9_{\delta,2}} & 0 & 0 \\ 0 & 0 & 0 & z^{-n^{10}_{\delta,4}} \end{bmatrix}.$$

According to the multivariable Nyquist criterion [30], the stability condition is equivalent to the following statement: the eigenvalues of $G(e^{j\omega})$, for ω from 0 to 2π, should not encircle the point -1. Let $\lambda(G(e^{j\omega}))$ denote any eigenvalue of $G(e^{j\omega})$. Suppose the RTT for all paths are identical so that $n^l_{\delta,j} + n^l_{\beta,k} = n_T$ for all users sharing link l, then the eigenvalues of $G(e^{j\omega})$ is given by

$$\begin{aligned} \lambda(G(e^{j\omega})) &= \lambda(\alpha M P_0 C R_0 D)\frac{e^{-j\omega n_T}}{e^{j\omega}-1} \\ &= \alpha\lambda(H)\frac{e^{-j\omega n_T}}{e^{j\omega}-1}, \end{aligned}$$

where $H = MP_0CR_0D$. H has all eigenvalues of real number. Let $\|A\|_\infty$ denote the matrix ∞- norm $\|A\|_\infty = \max_i \sum_j |A_{ij}|$ which is the maximum row sum. The magnitude of any eigenvalue of H is upper-bounded by

$$|\lambda(H)| \le \|H\|_\infty \le \|M\|_\infty \|P_0 C R_0\|_\infty \|D\|_\infty. \tag{2.10}$$

Note that

$$\|M\|_\infty = 2 - \frac{2}{K^{\max}}, \quad \|D\|_\infty = d^{\max},$$

$$\|P_0 C R_0\|_\infty \leq \frac{N^{\max}}{c^{\min}},$$

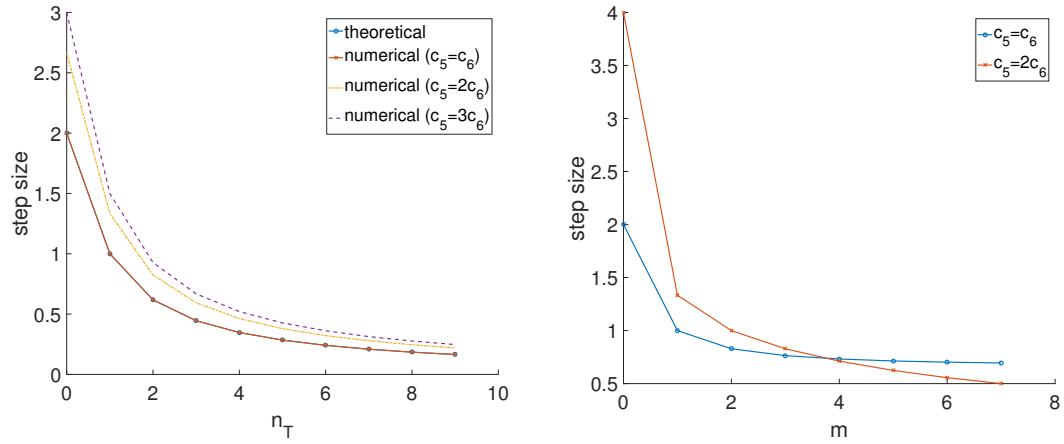
where $K^{\max} \geq 2$ and $d^{\max}$ are the maximum values of K_i and d_i among all users i, $N^{\max} \geq 1$ is the maximum number of users sharing a link, and $c^{\min}$ is the minimum link capacity.

Furthermore, when $Im\{\frac{e^{-j\omega n_T}}{e^{j\omega}-1}\} = 0$, $Re\{\frac{e^{-j\omega n_T}}{e^{j\omega}-1}\}$ is lower bounded by

$$\begin{aligned} Re\{\frac{e^{-j\omega n_T}}{e^{j\omega}-1}\} &\geq \left(\frac{e^{-j\omega n_T}}{e^{j\omega}-1}\right)_{\omega=\pi/(2n_T+1)} \\ &= \frac{\sin\frac{\omega}{2}}{\cos\omega - 1} = -\frac{1}{2\sin\frac{\omega}{2}} \\ &= -\frac{1}{2\sin\frac{\pi}{4n_T+2}}. \end{aligned}$$

Therefore the sufficient condition for the closed-loop system stability with homogeneous RTT is given by

$$\alpha < \frac{\sin\frac{\pi}{4n_T+2} K^{\max} c^{\min}}{(K^{\max}-1) d^{\max} N^{\max}}.$$



(a) The upper bound of step size for homogeneous RTT as n_T varies.

(b) The upper bound of step size for heterogeneous RTT. $m = \frac{n_{\delta,5}}{n_{\delta,6}}$.

Figure 2.5: The upper bound of step size.

Figure 2.5a shows the theoretical upper bound of α derived from the sufficient condition above in homogeneous RTT case as n_T varies from 0 to 9, and the

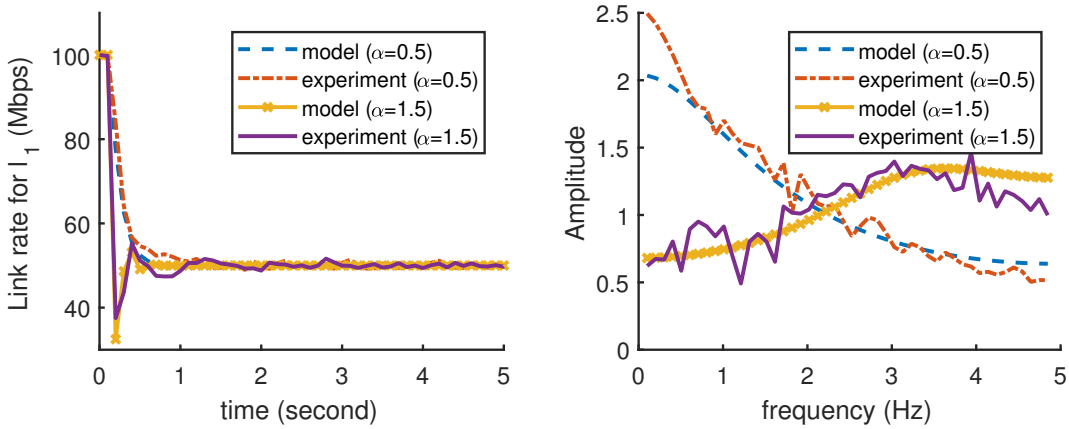
numerical results computed for the necessary condition of stability. In the theoretical computation, we set $c^{\min} = 20Mbps$, $d^{\max} = 5Mbps$, $N^{\max} = 4$ and $K^{\max} = 2$. In the numerical computation of homogeneous RTT case, we set the demand for all users to be $5Mbps$, all links capacity to be $20Mpbs$ except c_5. When $c_5 = c_6$, the numerical results are exactly the same as our theoretical derivation because $|\lambda(H)| = \frac{2(K^{\max}-1)d^{\max}N^{\max}}{K^{\max}c^{\min}}$. When $c_5 > c_6 = 20Mbps$, as is shown in the case of $c_5 = 2c_6$ and $c_5 = 3c_6$, $c^{\min} = 20Mbps$ still holds, but $|\lambda(H)|$ becomes strictly less than its upper bound. So in these two cases we can see α's upper bound derived from sufficient condition is less than the numerical computation for necessary condition.

We further show the step size upper bound for some heterogeneous RTT cases in Figure 2.5b. In the computation, we ignore all the backward delay and make the forwarding propagation delay identical for all the links from any path using this link, i.e. $n^p_{\delta,l} = n_{\delta,l}, \forall p \in \mathcal{L}_l$. We set $n_{\delta,6} = 1$ and vary $n_{\delta,5}$ to obtain the upper bound of α as the propagation delay difference between two bottleneck links changes. Comparing with Figure 2.5a, the theoretical result for sufficient condition will also hold for heterogeneous RTT cases with n_T being the maximum n_T, while the sufficient condition of stability for heterogeneous RTT remains to be further studied.

2.4 Evaluation

In this section, we describe our emulation setup, and demonstrate the validation results for our model and analysis.

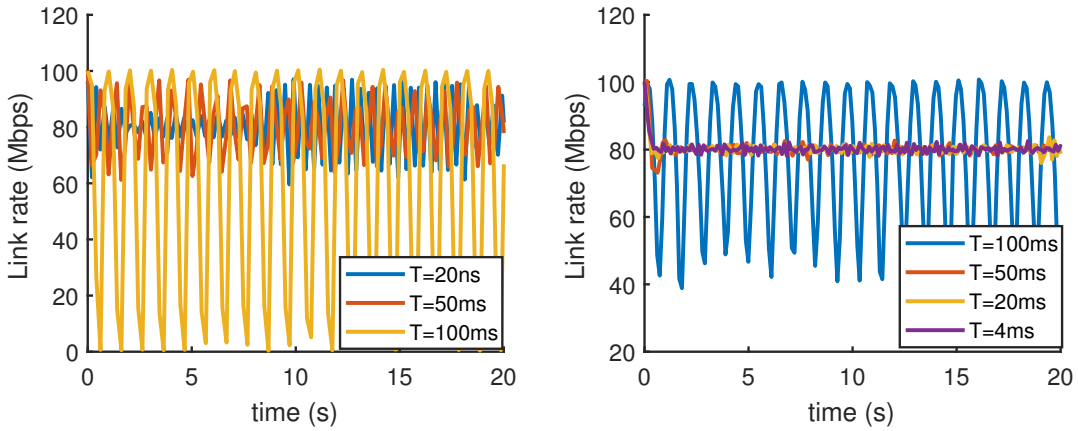
2.4.1 Validating the model



(a) Link rate for l_1 in time domain.

(b) Amplitude spectrum of link rate for l_1. $\tau = 100ms, T = 10ms$.

Figure 2.6: Model validation.



(a) $\alpha = 1.3, \tau = 200ms$.

(b) $\alpha = 1, \tau = 100ms$.

Figure 2.7: Validation of stability region for single-link.

A four-node topology, as is shown in Figure 4.3, is set up on Mininet for model validation. Each node is associated with an Open vSwitch [11]. The two paths are pre-configured in the network so that the switches forward the packets accordingly. A customized controller with our control law 2.1 is deployed on the ingress node. The routing computation is triggered every τ and then the

updated routing rules will be installed to the ingress switch. The ingress switch uses group table to realize the multi-path routing with unequal weights. The switch in each node measures the link throughput within a measurement period τ. A probing packet is created and sent along each path to collect the link states every τ. We set $C = 100Mbps$ for all links.

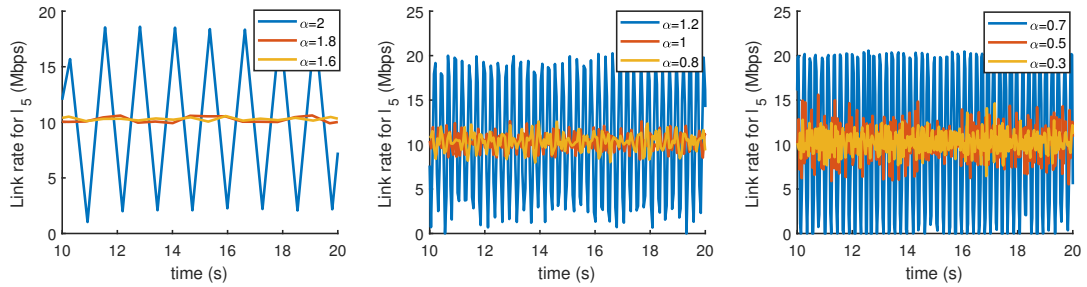
In the model validation experiments, 100 Mbps traffic destined to E_0 are generated at I_0 via iPerf. The initial routing strategy is configured to be forwarding all traffic through the upper path. Adaptive TE implemented in the controller collects the statistics of path metrics and dynamically adjusts the split ratio for each path. The actual evolution of transmission rate on link l_1 shown in Figure 2.6 validates the predictions from our model. With different values of the step size,the experimental curves for the measured link rate of l_1 in both time domain and frequency domain show good agreement with the ideal curve predicted by the model. On the experimental curve, the noise effect produced by the measurement can also be observed.

2.4.2 Validating the analysis

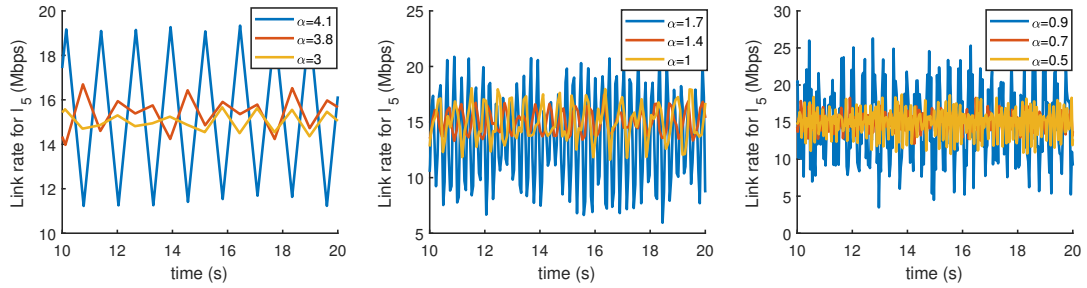
Our analysis is validated with respect to single bottleneck link and shared links topologies. Two representative adaptive TE control laws are considered in the experiments: the general form specified in Eq. 2.1 and a particular routing strategy proposed by TeXCP [44].

On the testbed with four-node topology, we introduce $60Mbps$ background traffic on link l_4, and generate 100 Mbps traffic for user $I_0 - E_0$. Therefore, the split ratios for user $I_0 - E_0$ are expected to be stabilized around 0.8 and 0.2 for

the upper path and the lower path respectively. To judge whether the system is stabilized, we record the measured transmission rate of link l_1. Since our analysis in Section 2.3.2 indicates that the stability is dependent of the step size, feedback delay and update interval, experiments are carried out with different values of the three parameters. The results in Figure 2.7a and Figure 2.7b show the agreement on the stability region of α with the theoretical results for single-link in Figure 2.3c. When $\alpha = 1.3$, the throughput of l_1 is stable in the case of $T = 20ns$. The oscillation is larger for $\tau/T = 2$ than $\tau/T = 4$. In Figure 2.7b, $\alpha = 1$ is not within the stability region of $\tau/T = 1$ ($T = 100$ ms). In both of the experiments, we fix the value of τ which is large enough to maintain the effect measurement noise input small and identical when T varies.



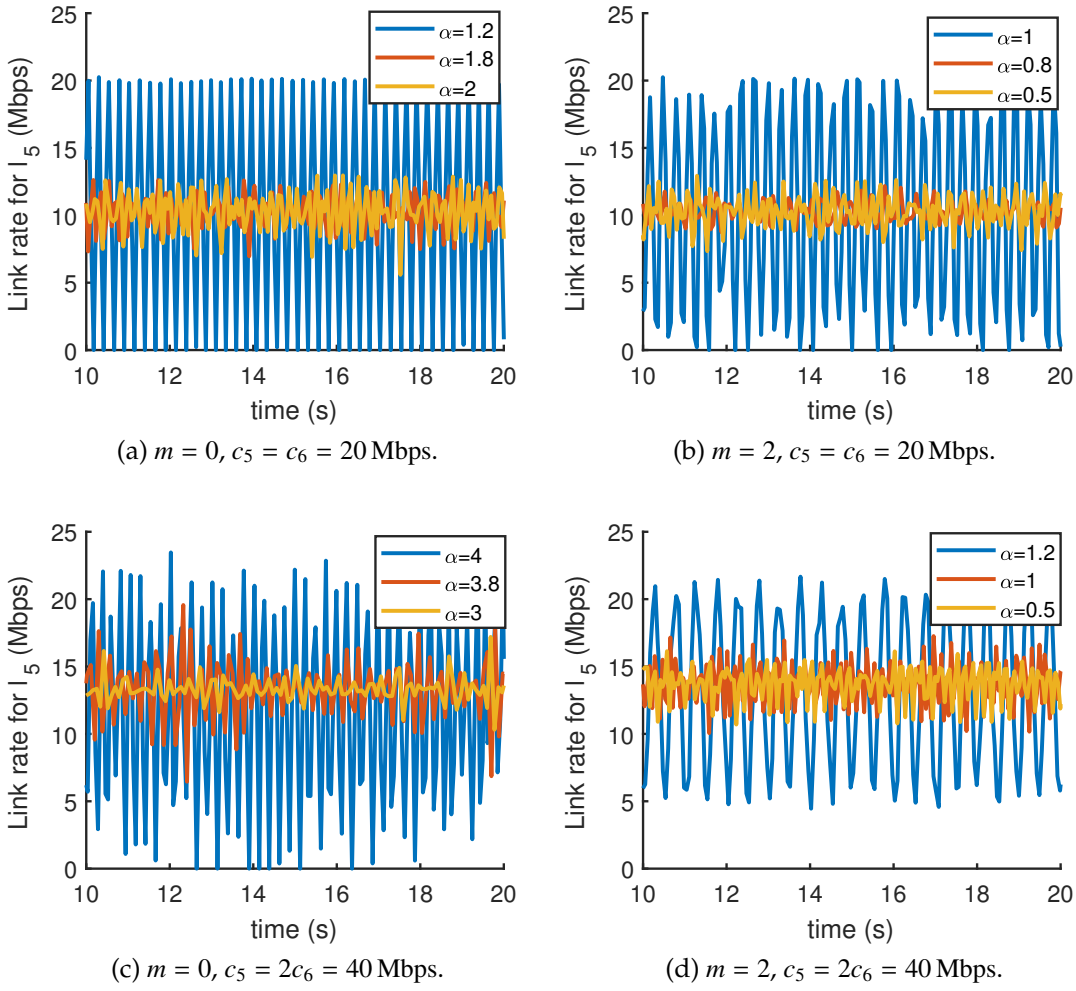
(a) $n_T = 0.1$, $c_5 = c_6 = 20$ Mbps. (b) $n_T = 1$, $c_5 = c_6 = 20$ Mbps. (c) $n_T = 2$, $c_5 = c_6 = 20$ Mbps.



(d) $n_T = 0.1$, $c_5 = 3c_6 = 60$ Mbps. (e) $n_T = 1$, $c_5 = 3c_6 = 60$ Mbps. (f) $n_T = 2$, $c_5 = 3c_6 = 60$ Mbps.

Figure 2.8: Validation of stability region for shared-link with identical RTT = 60 ms.

We proceed to validate our analysis for shared-link case with multi-user (Section 2.3.3) by setting up an eight-node topology shown in Figure 2.4 on the



(a) $m = 0$, $c_5 = c_6 = 20$ Mbps.

(b) $m = 2$, $c_5 = c_6 = 20$ Mbps.

(c) $m = 0$, $c_5 = 2c_6 = 40$ Mbps.

(d) $m = 2$, $c_5 = 2c_6 = 40$ Mbps.

Figure 2.9: Validation of stability region for shared-link with heterogeneous RTT.

same testbed. A centralized controller is implemented which controls the routing update for all users. Each user starts a session of 5 Mbps traffic transfer to its destination asynchronously. The capacity for each link except l_5 is fixed at 20 Mbps. In the case of homogenous RTT, the RTT for each path is 60 ms. To justify the stability region in Figure 2.5a with different n_T, we consider three values of the update interval τ (600 ms, 60 ms and 30 ms) and two values of link capacity c_5 (20 Mbps and 60 Mbps). Figure 2.8 shows in each case, when the transmission rate of l_5 exhibits periodic oscillation, the corresponding value of the step size is

outside the stability region predicted in Figure 2.5a. For the experiments with heterogenous RTT, the paths using link l_6 have RTT = 60 ms, and the paths using link l_5 have RTT of either 0 or 120 ms. The update interval is configured to be $\tau = 60$ ms so that m is either 0 or 2 with $n_{\delta,6} = 1$. We consider two values of m (0 and 2) and two values of c_5 (20 Mbps and 40 Mbps). Figure 2.9 provides the validation for the stability region in Figure 2.5b.

We finally validate the stability region analysis of TeXCP [44] using our proposed methodology. TeXCP mainly consists of two layers of control functions: the outer layer is the adaptive TE update, and inside there is a feedback congestion control loop operating at least 5 times within one TE update. We simplify the process by ignoring the inner loop of congestion control and only consider the feedback loop of TE updates. Since the congestion control strategy guarantees the state variables $x(n)$ obtained have been stabilized to at least 95% of the latest control variables $u(n)$, we wait for long enough time to make sure the transmission rates have become stable. Therefore, we set $\tau = 5T$ in all of the following experiments. According to the method of routing computation described in TeXCP, its TE control law for path $k \in \mathcal{P}_i$ is in the following form

$$u_k(n) = u_k(n-1) + u_k(n-1)[\sum_{j \in \mathcal{P}_i} u_j(n-1)x_j(n) - x_k(n)].$$

In the single-link case, the routing rule can be written as

$$u(n) = u(n-1) + 2u(n-1)(1-u(n-1))(\frac{D}{2C} - x(n)).$$

Compared with Eq. 2.9, the step size of TeXCP follows that $\alpha = 2u(n)(1-u(n))$, which is a varying number in the range of $[0, 0.5]$ because $u(n) \in [0, 1]$. Based on the stability region we derived in Figure 2.3c, the range of step size $[0, 0.5]$ will guarantee the stability regardless of the feedback delay and update interval, as

is shown in Figure 2.10a. By imposing an additional parameter a on the TE control law such that

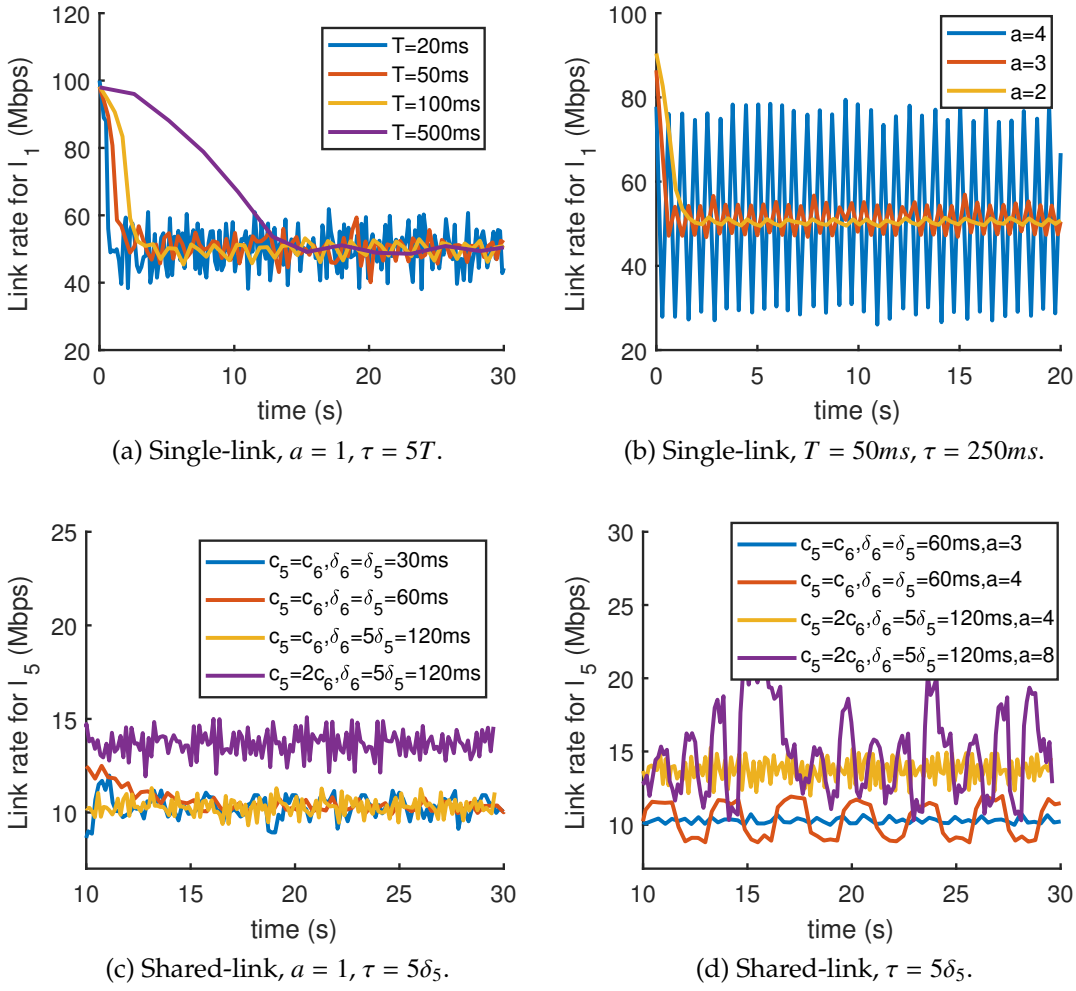
$$u(n) = u(n-1) + a \cdot 2u(n-1)(1-u(n-1))(\frac{D}{2C} - x(n)),$$

we can investigate the impact of step size on stability. In this case, the step size is written as $\alpha = a \cdot 2u(n)(1-u(n))$ and it is easy to compute its boundary $\alpha \in [0, 0.5a]$. Note that with the parameter a adding in the control law, the original design in TeXCP is a special case where $a = 1$. In Figure 2.10b, when $a = 4$ indicating that the step size could be as large as 2, the system becomes unstable. It is consistent with our analysis in Figure 2.3c that when $\tau/T = 5$, the upper bound of step size is less than 2.

In the shared-link case, the specific control law for the given network topology in Figure 2.4 follows that

$$u(n) = u(n-1) - 2U(n-1)(I - U(n-1))Mx(n),$$

where $U(n) := diag(u_k(n)) \in \mathbb{R}^{K \times K}$. Note that the step size in the control law is considered as a scalar in Eq. 2.6, whereas TeXCP attaches a specific step size $\alpha_k(n) = 2u_k(n)(1 - u_k(n))$ to each control variable u_k. We run a set of experiments on the testbed with 8-node topology, including the cases of homogeneous/heterogenous RTT and link capacities. δ_5 and δ_6 are defined as the RTT of the paths that use link l_5 and l_6 respectively. The update interval remains to be $\tau = 5\delta_5$, so that $n_T = 0.2$ holds in homogeneous RTT case, while in heterogenous RTT case $n_{\delta,5} = 0.2, n_{\delta,6} = 1$. Figure 2.10c demonstrates that the original TeXCP control law ensures the routing update to be stabilized in all cases. We again impose the parameter a on the original control law and experimentally seek for the new stability region by tuning the value of a. Figure 2.10d shows that with homogeneous RTT, $a = 4$ makes the system unstable, which validates the upper



(a) Single-link, $a = 1$, $\tau = 5T$.

(b) Single-link, $T = 50ms$, $\tau = 250ms$.

(c) Shared-link, $a = 1$, $\tau = 5\delta_5$.

(d) Shared-link, $\tau = 5\delta_5$.

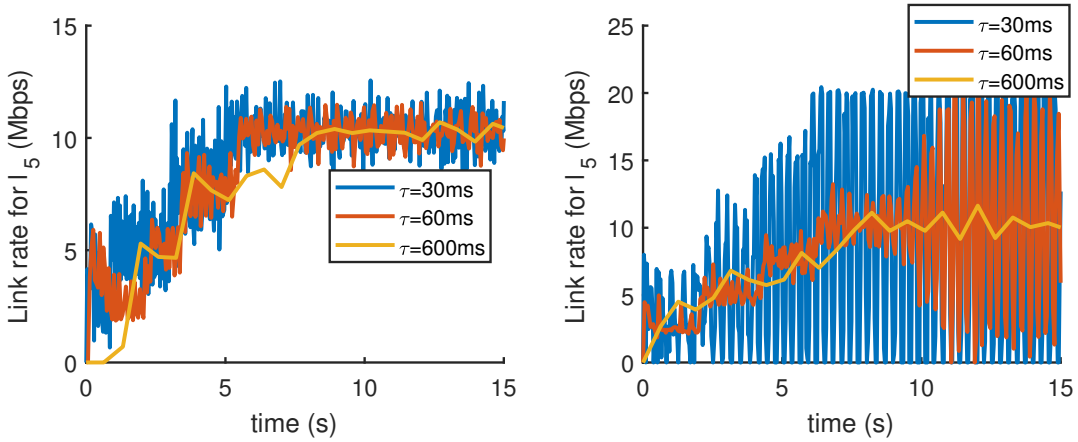
Figure 2.10: Validation of stability region for TeXCP.

bound of step size for $n_T = 0.2$ in Figure 2.5a. Furthermore, it also provides justification of the stability region for heterogeneous RTT in Figure 2.5b when $m = 0.2$ and $c_5 = 2c_6$.

2.4.3 Stability versus responsiveness

Our comprehensive experiments above have shown that the critical system parameters and their internal relations play significant roles on the system sta-

bility. Figure 2.10 for single-link case and Figure 2.11 for shared-link case are good examples of illustrating how the choices of step size α, update interval τ and feedback delay T affect the performance tradeoff between stability and responsiveness. Observation from Figure 2.10a indicates that the system is more responsive when the feedback delay is smaller. Figure 2.10b shows that a larger step size leads to faster convergence to the set point, however, too large a step size will cause the routing oscillation. In Figure 2.11a, more frequent update can make the system respond and converge more quickly, while it turns out in Figure 2.11b that smaller update interval may result in a smaller stability region for choices of the step size. Our quantitative analysis for the model helps to restrict the boundary of the parameters based on stability condition, and thus facilitates the systematic evaluation of parameters' effects on the performance.



(a) $\alpha = 0.5$. (b) $\alpha = 1.2$.

Figure 2.11: Tradeoff between stability and responsiveness.

2.5 Summary

In this chapter we proposed a control theoretical model for fast adaptive traffic engineering. The model incorporates several important factors including physical propagation delay, routing update interval and step size. In particular, it formalizes the intrinsic tradeoff between being reactive, hence tending to use large step size and frequent sampling, and maintaining stability, which benefits from adopting more cautious steps as well as larger measurement duration to better filter out the high frequency noise. We analyze the model to provide guiding insight for network operators to set those parameters. Furthermore, experiments are carried out to quantitatively verify the predictions from the model. Our model covers related existing work as special cases and therefore can serve as a basis for further combination with different traffic models.

CHAPTER 3

SPATIAL MANAGEMENT: FINE-GRAINED ACCURATE TRAFFIC SPLIT

3.1 Introduction

Multipath routing technologies that allocate traffic among multiple paths are widely used to provide better load balancing across the network. The multipath routing solution usually defines how to split the traffic among the available paths in the form of a set of split ratios, each of which is associated with one path or outgoing interface. Once the routing algorithm determines the optimal paths for each commodity flow [1], the corresponding split ratios associated with each outgoing interface at each switch are then set to split the traffic over multiple paths as the solution requires.

The general process of forwarding traffic over multiple paths is shown in Fig. 3.3, which mainly comprises of two key components: traffic division and path selection. The arrival packets are first classified into traffic units in the traffic division module. The outgoing interface for each traffic unit are then determined independently by the path selection module, while all packets from the same traffic unit follow the path identically.

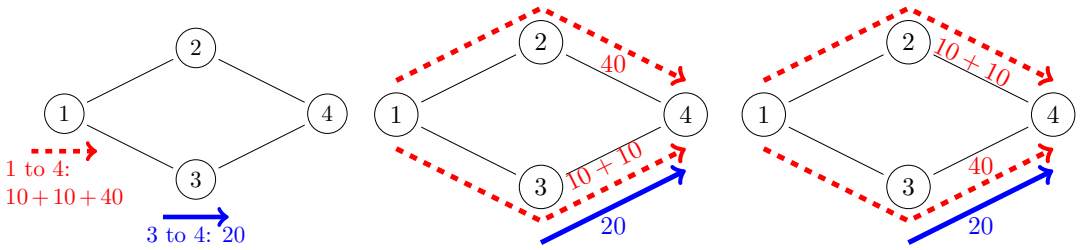
The aggregated traffic are divided into units at a certain level of granularity [61]. The smallest scale of division is a single packet. Since the path for each packet is determined independently, packet-level division achieves the finest-grained splitting but will cause the packet-reordering problems that severely

[1] The exact meaning of commodity depends on the granularity of the routing schemes. Usually each commodity corresponds to traffic with the same source-destination pair, or just with the same destination subnet.

degrades the throughput of transport layer protocols such as TCP. To preserve packet ordering, flow-level traffic division is widely used. In flow-level division, packets that share the same value of some fields in IP packet header are grouped together as a flow with a unique identifier. The granularity of flow unit depends on the fields to be matched in the packet header. Packets from a particular flow can be further grouped into subflows, referred to as flowlets, by considering the inter-arrival time of the packets [45].

The path selection module selects the path for each traffic unit independently. The path is determined based on the target split ratios with or without additional information of the traffic load. Being unaware of the traffic load information, the path selector either selects the path in a round robin manner for individual packets which is not desirable due to packet reordering issues, or performs the hash-based path selection for flows/subflows assuming a uniform flow size distribution. This type of path selection schemes in general has low computational complexity and overhead. It is, however, not able to control the actual split ratios when the flow size distribution is skewed. The other type of schemes collects the traffic load information and dynamically adjusts the existing paths assignment in real time to make the actual split ratios close to the target ones. In many today's commodity routers, equal-cost-multipath (ECMP) is implemented which splits traffic evenly over each path. Weight-equal-cost-multipath (WCMP) is offered by some type of routers with a coarse granularity of split ratios being supported. Those models being implemented nowadays do not collect the information on traffic or network condition in the path selection process, and therefore cannot provide accurate traffic splitting over multiple paths.

There are two critical problems involved in the above described picture. First, how well can a switch realize given traffic split ratios? TE algorithms usually take commodity traffic demand or aggregate link loads as input and then solve certain mathematical optimization problems assuming traffic can be split arbitrarily. However, in practice, individual flows are not split over multiple paths to avoid potential issues of packet out-of-order as mentioned above. This flow-level granularity constraint can well force the actual split ratios deviate from the ones that are demanded by TE. This is particularly true when flows have vastly different rates while current solutions treat all flows equally when it comes to decide which flows should be moved to another path.



(a) Demand (b) Optimal solutions (c) Actual results

Figure 3.1: Example of inaccurate splitting.

Such inaccurate load distribution deviating from target TE solutions can result in negative effects such as network performance degradation due to imbalanced resource utilization or even link capacity violation. We here illustrate such consequences with a simple example (Fig. 3.1). The demand matrix consists of two commodities of traffic: 60 units from node 1 to node 4 and 20 units from node 3 to node 4 (Fig. 3.1a). Considering the objective of TE as to minimize the maximum link load, one optimal routing scheme (Fig. 3.1b) is to forward 2/3 of the traffic from node 1 to node 4 through the upper path and 1/3 through the lower path. Suppose the traffic are divided into three individual flows based

on 5-tuple hash. Being unaware of the flow sizes, path selection module may assign 2 flows with size 10 to the upper path (Fig. 3.1c). The resulting link load imbalance deviates from the objective of TE and can violate the link capacity constraint if the capacity is no more than 60 units.

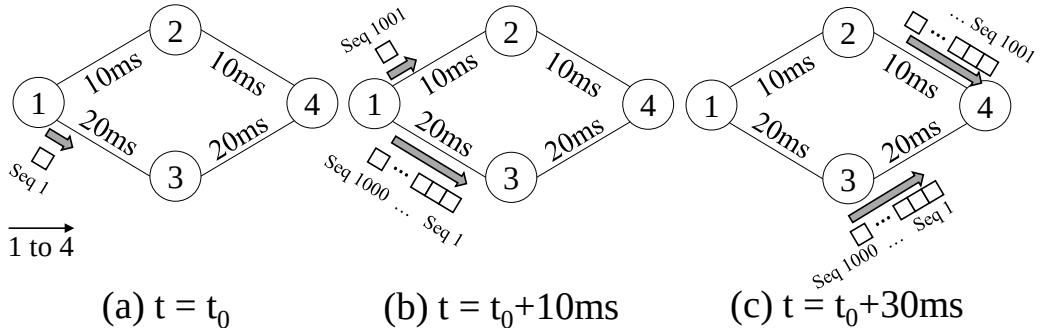


Figure 3.2: Example of out-of-order packets.

Second, how much performance cost does realizing traffic splits incur? Here we are not discussing the cost that are caused by the control plane, such as transient loops, but focusing on the cost that is associated with moving a flow from one path to another in the data plane, which can lead to out-of-order packet arrivals when those two paths have different latency. Fig. 3.2 demonstrates an example of such packets reordering. Initially at time $t = t_0$, packets starting with sequence number 1 were forwarded through the lower path from node 1 to node 4 (Fig. 3.2a). After 10 ms, its routing path was changed from the lower path to the upper one. Packets with sequence number 1 to 1000 were already sent out during the past 10 ms and were traveling on the lower path (Fig. 3.2b). It took around 20 ms for the first packet on the upper path with sequence number 1001 to reach the destination. At time $t = t_0 + 30$ ms, the packets with smaller sequence number were still on the way between node 2 to node 4 (Fig.3.2c), so out-of-order packets were encountered at the receiver.

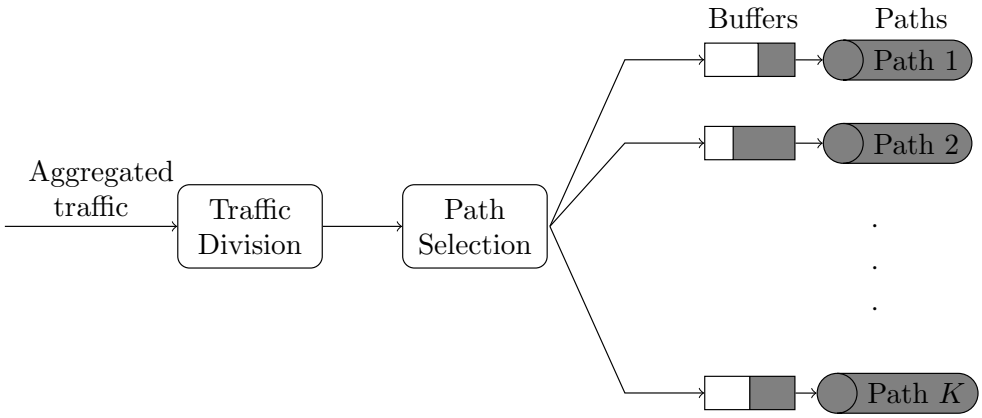


Figure 3.3: Multipath forwarding system

As traffic demand rapidly grows in both volume and variation, to meet with the Quality of Service (QoS) requirements at high data rates, TE techniques need to become finer, both in terms of adjustment precision as well as update frequency. These in term directly require better solutions in the data plane to deal with the above mentioned two issues, i.e., load balancing mechanisms that can produce traffic splits that are close to what TE algorithms demand with small update cost. Finally, it is worth noting that these two requirements are of competing nature since updating to new traffic splits more precisely usually needs to shuffle more flows. In this chapter, we propose a load distribution scheme that incorporates such tradeoff in a weighted sum optimization. It aims to find the most accurate traffic splits with minimum route changes based on the collected load sharing statistics.

3.2 Design

In this section, we discuss our design choices in detail. Those then become the constraints of our mathematical formulation. An algorithmic solution to the for-

mulation is provided and will be mapped back to the system in the next section. We focus on the flow-level path reassignment problem for one commodity and assume that there are multiple paths available with corresponding target split ratios provided by TE.

3.2.1 Key Features

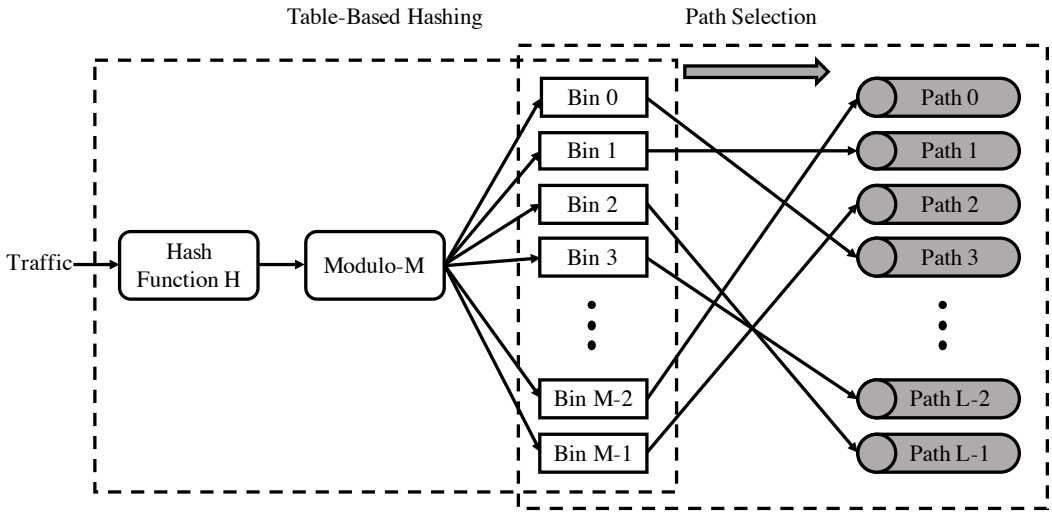


Figure 3.4: Design of traffic splitting.

Fig 3.4 illustrates our design of traffic splitting mechanism. In this subsection, we first present some important design decisions and key features.

Table-based hashing: Our flow-level traffic splitting approach bases on table-based hashing. A flow is first identified by five-tuple in the hash function H and then assigned to a bin among M bins according to its hash value. Each bin is mapped to one of the L paths. We choose table-based hashing for three main reasons: 1) It provides the flexibility of remapping between bins and paths; 2) Path reassignment is scalable because the number of bins is limited

and does not change as the number of flows increases; 3) Table-based hashing makes it possible to separate traffic with different routing rules required by different classes of service of priorities. For example, if the high priority traffic are required to be forwarded strictly through the single shortest path, then they need to be excluded from the multipath routing process. It can be realized by assigning this particular type of traffic to a dedicated set of bins which do not allow bin-path remapping.

Rate-aware dynamic path reassignment: The path selection process determines the remapping between bins and paths. Dynamic remapping is proposed to minimize the real-time error between the actual load distribution and the target split ratio for each path. We effectively reassign some of the bins to different paths with the knowledge of the current load sharing statistics.

Direct control of reassignment level: In order to quantitatively control the degree of packets reordering caused by path reassignment, we consider a limit for the amount of traffic that are shifted to a different path.

Valuable reassignment only: In addition to the limited level of reassignment, we also consider a weighting parameter to evaluate the tradeoff between splitting error and the reassignment level. A certain level of remapping is adopted only if it is valuable. We target at finding the best strategy to minimize the error of splitting with only necessary and worthwhile path reassignments.

3.2.2 Problem Formulation

We now formulate the problem of bin-path remapping. Suppose there is a set $\mathcal{M}$ of M bins being assigned to a set $\mathcal{L}$ of L different paths. Let r_i be the traffic arrival rate of bin i. Let α_l be the target split ratio of path l that is provided by TE solution. x_i^l denotes the mapping decision between bin i and path l, and we have $x_i^l = 1$ if bin i is assigned to path l and 0 otherwise. We further define the overflow fraction ρ_l to be the difference between actual load fraction and the target split ratio for path l, given by $\rho_l = \frac{\sum_{i \in \mathcal{M}} r_i x_i^l}{\sum_{i \in \mathcal{M}} r_i} - \alpha_l$. Therefore the maximum overflow fraction among all available paths is $\rho = \max_{l \in \mathcal{L}}\{\rho_l\}$. Assuming the current assignment strategy is determined by a set of $\{x_{i,0}^l\}$, we aim at finding the optimal $\{x_i^l\}$ that minimizes the weighted sum of the maximum overload fraction ρ and the degree of reassignment fraction. The weighting parameter is denoted as λ. We propose two options to quantify the degree of reassignment fraction. The first option is to consider the number of bins being remapped. We add an additional constraint on the number of bins that can be reassigned from the current path to a different one. If the number of reassigned bins is limited by K, then the problem formulation is given by

$$\min \ \rho + \lambda\Delta \tag{3.1}$$

$$\text{s.t.} \ \sum_{i\in\mathcal{M}} r_i x_i^l \le (\rho + \alpha_l) \sum_{i\in\mathcal{M}} r_i \quad \forall l \in \mathcal{L} \tag{3.1a}$$

$$\sum_{l\in\mathcal{L}} x_i^l = 1 \qquad \forall i \in \mathcal{M} \tag{3.1b}$$

$$x_i^l \in \{0,1\} \qquad \forall i \in \mathcal{M}, l \in \mathcal{L} \tag{3.1c}$$

$$M - \sum_{l\in\mathcal{L}} \sum_{i\in\mathcal{M}} x_i^l x_{i,0}^l = \Delta \tag{3.1d}$$

$$\Delta \le K \tag{3.1e}$$

In the objective function, the parameter λ determines the tradeoff between maximum overflow and the number of bins being reassigned. Condition 3.1a defines the maximum overflow fraction ρ such that the overflow fraction for each path should not exceed. Condition 3.1b and 3.1c require that one bin can only be assigned to one path. Condition 3.1d describes the number of bins that are selected to be remapped. Condition 3.1e restricts the number of bins to be remapped. One underlying rule based on the objective function is if the initial maximum overflow ρ_0 satisfies $\rho_0 \le \lambda$, then the optimal value is $opt^* = \rho_0$ with $\Delta^* = 0$, meaning that we prefer to not reassign any bins because any path change costs more price than benefits. Another intuitive rule following that is any optimal solution of Δ^* satisfies $\Delta^* \le \rho_0/\lambda$.

Our problem formulation differs from the canonical bin-packing problem in two main aspects. Firstly, we consider the price of the bin reassignment, and the solutions are constrained by the number of bins being reallocated. Furthermore, overflow is mostly possible in our reallocation strategy while it is not allowed in the original bin-packing problem.

The second option of quantifying the degree of reassignment is the fraction of traffic load that are shifted. If the fraction of traffic load that can be reassigned is limited by δ, then the problem is formulated in the following form:

$$\begin{aligned}
&\min \ \rho + \lambda\Delta^+ && && (3.2)\\
&\text{s.t.} \ \sum_{i:\in\mathcal{M}} r_i x_i^l \le (\rho + \alpha_l) \sum_{i:\in\mathcal{M}} r_i && \forall l \in \mathcal{L}\\
&\quad \sum_{l:\in\mathcal{L}} x_i^l = 1 && \forall i :\in \mathcal{M}\\
&\quad x_i^l \in \{0,1\} && \forall i \in \mathcal{M}, l \in \mathcal{L}\\
&\quad \sum_{i:\in\mathcal{M}} (1 - \sum_{l:\in\mathcal{L}} x_i^l \eta_i^l) r_i = \Delta^+ && && (3.2a)\\
&\quad \Delta^+ \le \delta \sum_{i:\in\mathcal{M}} r_i && && (3.2b)
\end{aligned}$$

Only condition 3.2a and condition 3.2b are different from problem 3.1, which states the limit of traffic load fraction that are reassigned.

3.2.3 Algorithm

In this subsection, we propose an efficient algorithm that computes the approximate reassignment solutions to the optimization problem. The idea behind the algorithm is to first estimate a lower bound of the maximum overflow fraction, to position the goal of accuracy at an achievable and reasonable level with the limit on the reassignment degree. Based on the lower bound, we then select the set of bins from a candidate pool and reassign them to new paths.

Lower bound of the maximum overflow

We obtain a lower bound of the maximum overflow fraction by finding the optimal solution to the following Linear Programming (LP) relaxation:

$$
\begin{aligned}
&\min \ \rho && && (3.3)\\
&\text{s.t.} \ \sum_{i\in\mathcal{M}} r_i x_i^l \le (\rho+\alpha_l)\sum_{i\in\mathcal{M}} r_i && \forall l\in\mathcal{L}\\
&\sum_{l\in\mathcal{L}} x_i^l = 1 && \forall i\in\mathcal{M}\\
&x_i^l \in [0,1] && \forall i\in\mathcal{M}, l\in\mathcal{L}\\
&\sum_{i\in\mathcal{M}}(1-\sum_{l\in\mathcal{L}} x_i^l x_{i,0}^l)r_i \le \delta\sum_{i\in\mathcal{M}} r_i
\end{aligned}
$$

δ in problem (3.3) is the limitation of load reassignment provided by problem (3.2), and we set $\delta = K/M$ if the reassignment is constrained by the limited number of reassigned bins K in problem (3.1). Let ρ_l^0 be the initial overflow fraction for path l, i.e., $\rho_l^0 = \frac{\sum_{i\in\mathcal{M}} r_i x_{i,0}^l}{\sum_{i\in\mathcal{M}} r_i} - \alpha_l$. Assuming the paths are sorted in the descending order of ρ_l^0, we then have the following

Theorem 1. *The optimal value ρ^* of the LP relaxation (3.3) is*

$$\rho^* = \max\{\frac{\sum_{l=1}^{\hat{l}}\rho_l^0 - \delta}{\hat{l}}, 0\}, \tag{3.4}$$

where $\hat{l}$ is defined as

$$\hat{l} = \min\{i\in\mathcal{L} : \frac{\sum_{l=1}^{i}\rho_l^0 - \delta}{i} \ge \rho_{l+1}^0\}. \tag{3.5}$$

Proof. First note that the sum of overflow fraction for all paths is zero because $\sum_{l\in\mathcal{L}}\rho_l = \frac{\sum_{l\in\mathcal{L}}\sum_{i\in\mathcal{M}} r_i x_i^l}{\sum_{i\in\mathcal{M}} r_i} - \sum_{l\in\mathcal{L}}\alpha_l = 0$. Hence by definition ρ^* is nonnegative. Let $\mathcal{L}^+$ be the set of paths being overloaded initially, i.e. $\rho_l^0 > 0, \forall l \in \mathcal{L}^+$, and assume $\mathcal{L}^+$ is a sorted set in the descending order of ρ_l^0. Similarly, let $\mathcal{L}^-$ be the sorted set of paths being underloaded initially.

a) If $\sum_{l\in\mathcal{L}^+}\rho_l^0 \le \delta$, then $\frac{\sum_{l=1}^{\hat{l}}\rho_l^0-\delta}{\hat{l}} \le 0$ for $\forall\hat{l}\in\mathcal{L}$. In this case, Eq. 3.4 results in $\rho^*=0$. This can be easily achieved with total reassignment fraction no larger than δ by moving exactly ρ_l^0 load fraction on each path $l\in\mathcal{L}^+$ to paths in $\mathcal{L}^-$.

b) If $\sum_{l\in\mathcal{L}^+}\rho_l^0 > \delta$, then by Eq. 3.5, $\hat{l}$ is no less than the last-indexed path in $\mathcal{L}^+$. So Eq. 3.4 results in $\rho^*=\frac{\sum_{l=1}^{\hat{l}}\rho_l^0-\delta}{\hat{l}}>0$. It can be achieved by shifting load from each path $l=1,\cdots,\hat{l}$ to paths in $\mathcal{L}^-$ with the amount of $\rho_l^0-\rho^*$. We prove its optimality by contradiction. Suppose there exists a feasible solution resulting in $\rho'<\rho^*$, then the reassigned load fraction is

$$\sum_{l=1}^{\hat{l}}\rho_l^0-\hat{l}\rho' > \sum_{l=1}^{\hat{l}}\rho_l^0-(\sum_{l=1}^{\hat{l}}\rho_l^0-\delta)=\delta.$$

This contradicts the constraint of reassigned load limit δ. Therefore, in this case $\rho^*=\frac{\sum_{l=1}^{\hat{l}}\rho_l^0-\delta}{\hat{l}}$ is the optimal value.

Overall, the optimal value is $\rho^*=\max\{\frac{\sum_{l=1}^{\hat{l}}\rho_l^0-\delta}{\hat{l}},0\}$.

□

We here present a procedure to construct a set of reassigned bins and show that it leads to an optimal solution to the LP relaxation (3.3).

Procedure 1:

(i) Starting from the first path l in $\mathcal{L}^+$, we define a critical bin s_l for path l such that $s_l=\min\{i\in B_l:\sum_{j=1}^{i}r_j>(\rho^*+\alpha_l)\sum_{i\in\mathcal{M}}r_i\}$, where B_l is the set of bins initially assigned to path l.

(ii) For $\forall i \in B_l$, we set the variable x_i^l to be

$$x_i^l = \begin{cases} 1 & \text{if } i < s_l, \\ \frac{(\rho^* + \alpha_l) \sum_{i \in \mathcal{M}} r_i - \sum_{j=1}^{s_l - 1} r_j}{r_{s_l}} & \text{if } i = s_l, \\ 0 & \text{otherwise.} \end{cases}$$

(iii) Repeat Step (i) and (ii) for each $l \in \mathcal{L}^+$ until $\hat{l}$. Define a set D for partially or completely disconnected bins such that $D = \{i \in B_l : x_i^l < 1, \quad l = 1, \cdots, \hat{l}\}$.

Theorem 2. *The set of reassigned bins obtained by Procedure 1 results in the optimal value ρ^* of the LP relaxation (3.3).*

Proof. The current load for path $l = 1, \cdots, \hat{l}$ after Procedure 1 is exactly $(\rho^* + \alpha_l) \sum_{i \in \mathcal{M}} r_i$, and thus we have that the current overflow fraction $\rho_l = \rho^*$ for $l = 1, \cdots, \hat{l}$. According to Eq. 3.5, $\rho_l^0 \leq \rho^*$ for $l = \hat{l} + 1, \cdots, L$. Given that $\sum_{l \in \mathcal{L}} \rho_l^0 = 0$ and the reassignment fraction is no larger than δ, we have

$$\sum_{l \in \mathcal{L}^-} \rho_l \leq \sum_{l \in \mathcal{L}^-} \rho_l^0 + \delta \leq -\sum_{l=1}^{\hat{l}} \rho_l^0 + \delta = -\sum_{l=1}^{\hat{l}} \rho_l = -\hat{l} \cdot \rho^* \leq 0$$

Hence there always exists a remapping solution between the bins in D and paths in $\mathcal{L}^-$ such that $\rho_l \leq 0$ for all $l \in \mathcal{L}^-$. So the maximum overload fraction among all path $l \in \mathcal{L}$ is ρ^*. Based on Theorem 1, ρ^* is the optimal value of the LP relaxation (3.3). Therefore, the set of reassigned bins following Procedure 1 leads to the optimal value of (3.3). □

Bins selection and reassignment

The optimal value ρ^* provides a lower bound for the optimization problem (3.1), guiding us to target at the reasonable level of accuracy without unnecessary

reassignment. Let D^* be the set of bins that we select to be remapped. We follow the procedure described above to find the set of disconnected bins D and put the items from D into D^* excluding the sets of critical bin s_l for each path $l = 1, \cdots, \hat{l}$. Then we compute the current overflow fraction ρ_l for path $l = 1, \cdots, \hat{l}$ and sort the set of path in descending order in terms of ρ_l. We select one bin from each of these paths to be further put into D^* until the number of reassigned bins exceeds K. Once D^* is complete, we use Best Fit algorithm to reconnect the bin set D^*. Algorithm 1 illustrates the pseudocode of our algorithm.

Input: $\{x^l_{i,0}\}$, $\{\alpha_l\}$, $\{r_i\}$, ρ_0, λ, K, $\forall i \in \mathcal{M}$, $\forall l \in \mathcal{L}$
Output: $\{x^l_i\}$
begin
  $opt = \rho_0$, $D^* \leftarrow \emptyset$
  for $j = 1$**to**K **do**
    $\delta \leftarrow j/M$;
    Find ρ^* and $\hat{l}$ using Eq. 3.4 and Eq. 3.5;
    Get D and $\{s_l\}$ by following procedure (i) - (iii);
    $D^* \leftarrow D \setminus \{s_l\}$;
    Compute $\{\rho_l\}$ for $l = 1, \cdots, \hat{l}$;
    foreach $l \in \{\rho_l\}$ *sorted in descending order* **do**
      Find bin b_t in path l such that $|\rho_l - \rho^* - r_t / \sum_{i \in \mathcal{M}} r_i|$ is minimum;
      if $Sizeof(D^*) + 1 \leq K$ **then**
        $D^* \leftarrow D^* \cup b_t$;
      else
        Break;
      end
    end
    Reassign bins in D^* among all paths in $\mathcal{L}$ using Best Fit algorithm;
    Compute $\{x^l_i\}_j$ and ρ_j;
    if $\rho_j + \lambda \cdot Sizeof(D^*) < opt$ **then**
      $opt \leftarrow \rho_j + \lambda \cdot Sizeof(D^*)$;
      $\{x^l_i\} \leftarrow \{x^l_i\}_j$;
    end
  end
end

Algorithm 1: Pseudocode.

Complexity

The time complexity of sorting set $\mathcal{L}$ with L paths is $O(LlogL)$. For a set of $\mathcal{M}$ with M bins and K maximum number of reassigned bins, the runtime complexity of bins selection and reassignment is $O(M + KlogK)$. So the worst-case time complexity of our proposed algorithm is $O(LlogL + K(M + KlogK))$. Note that the computation complexity is not increased with the amount of flows, and thus the splitting scheme is scalable. L is determined by the number of available paths in the network. The value of M and K affect the granularity of the traffic splitting. With the reasonable values of M and K to achieve desired accuracy in practice, both the computation complexity and the implementation complexity of the algorithm is marginal.

3.3 Implementation

Open vSwitch (OVS) is used here as the platform for implementation. We will first explain how traffic splitting is done in OVS including why it cannot provide accurate splits even when all flows are of the same size (Section 3.3.1). Implementation details are provided in Section 3.3.2.

3.3.1 Splitting Approach in OVS

The weighted traffic splitting is realized in group table in OVS. Multiple buckets with corresponding weights can be added to one entry of the group table. Buckets are associated with specific actions such as forwarding the packets to

a particular output port, so that the traffic splitting among multiple outgoing links/paths is realized by the bucket selection process. The bucket selection in OVS group selects the bucket on flow level identified by the 5-tuple.

The algorithm of flow-level weighted splitting implemented in OVS is the following. Suppose one group table entry contains L buckets. Let b_i and w_i denote the bucket id for the ith bucket and its corresponding weight, $i = 1, 2, \cdots, L$. When processing one packet p_k belonging to this group, it first obtains its hash value $a_k = H(p_k)$ by applying hash function H with 5-tuple as the input; then gets a hash value associated with each bucket $\beta_i^k = M(b_i, a_k)$ via hash function M; finally selects the bucket with the highest hash value of β_i^k.

It's worth to point out that the OVS's approach cannot produce accurate ratios, regardless of the quality of hashing function, the amount and heterogeneous of flows. To see that, consider the case of two buckets. Assuming there are enough amount of flows, the hash function $M(\cdot)$ for the two buckets can be viewed as choosing two independent integer random variables from the interval $[0, 2^{32} - s]$. We now compute the probability that the value of $\hat{\beta_1^k} \times w_1$ is greater than the value of $\hat{\beta_2^k} \times w_2$. Let X and Y be the two independent random variables from the interval $[0, a]$ and $[0, b]$ respectively with uniform probability density. The problem to be solved becomes computing $Prob(X - Y > 0)$, the probability that X is larger than Y. Let $f_X(x)$, $f_Y(y)$, $f_Z(z)$ denote the density functions for X, Y, and $Z = X - Y$ respectively. We have

$$f_X(x) = \begin{cases} \frac{1}{a} & 0 \le x \le a, \\ 0 & \text{otherwise.} \end{cases}$$

$$f_Y(y) = \begin{cases} \frac{1}{b} & 0 \le y \le b, \\ 0 & \text{otherwise.} \end{cases}$$

$$f_Z(z) = \int_{-\infty}^{+\infty} f_X(z+y) f_Y(y) dy = \frac{1}{b} \int_0^b f_X(z+y) dy$$

Because $f_X(z+y)$ is 0 unless $0 \le z+y \le a$, i.e. $-z \le y \le a-z$, assuming $a > b$, we have

$$f_Z(z) = \begin{cases} \frac{1}{ab} \int_{-z}^{b} dy = \frac{b+z}{ab} & -b \le z \le 0, \\ \frac{1}{ab} \int_{0}^{b} dy = \frac{1}{a} & 0 \le z \le a-b, \\ 0 & \text{otherwise.} \end{cases}$$

We can easily compute $Prob(X - Y > 0) = Prob(Z > 0) = \frac{2a-b}{2a}$, which is not equal to $\frac{a}{a+b}$ unless $a = b$. We illustrate an example of the density function in Fig **??**.

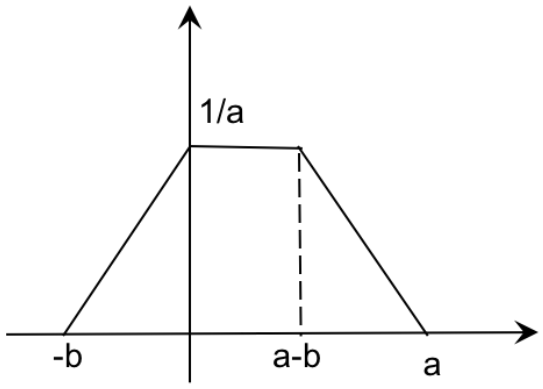


Figure 3.5: Example of density function $f_Z(z)$.

One optional revision with minimum modifications of the code is to select b_I such that $I = i : \max_j(\frac{\sum_{l=1}^{j} w_l}{\sum_{l=1}^{N} w_l} < \frac{\alpha_k}{2^{32}-1})$, which will be considered as a compared scheme in Section 3.4.

3.3.2 Implementation

The implementation of the traffic splitting mechanism is broken down into three components: initial path assignment, load statistics update and bin reallocation. Our implementation is based on the original mechanism of group table's weighted buckets in OVS with minimum modification in user space. Under the structure of a group, we insert a list of 32 bins. Each bin is associated with a bin id, a counter and a bucket.

Initial path assignment: When the first packet of a flow arrives, it is assigned to a specific bin. If the bin is already mapped to a bucket, then packets from this flow are forwarded through the path specified in the bucket. Otherwise, a bucket having the largest underload will be assigned to the bin.

Load statistics update: Our dynamic traffic splitting relies on the updated statistics from each bucket and bin. In OVS the datapath in user space polls the accumulated number of bytes information for each flow from the kernel space once every 0.5s. Our load statistics update requires the minimum implementation complexity by only additionally updating the counter of each bin according to the flow's statistics information. Then the rate (bytes per second) for each bin and each bucket are calculated.

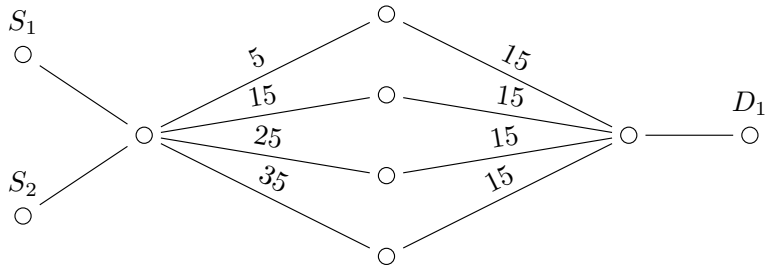
Bin reallocation: The function of bin reallocation can be triggered by two events. A timer fires the bin reallocation process once every T seconds. It aims at improving the accuracy of splitting for existing flows. In addition, it can be triggered immediately by any reconfiguration of target split ratios caused by TE routing re-computation. Once the bin reallocation process is triggered, it first collects the load information from bins and buckets, as well as the current bin

assignment. Then either the ILP problem (3.1) is structured and solved by the GLK solver, or the implemented algorithm described in Section 3.2.3 will be called. The output as the new assignment strategy will be configured.

3.4 Evaluation

3.4.1 Evaluation Environment

Testbed setup: We build a network topology in Mininet consisting of two source subnets S_1 and S_2, one destination subnet D_1 and four paths between them as shown in Fig. 4.3. Each subnet represents multiple hosts. The propagation delay (millisecond) is labeled on the corresponding link. The bandwidth of each link is 100 Mbps. OVS with the implementation of our traffic splitting mechanism is used as the virtual switch.



(a)

Figure 3.6: Network topology.

Traffic trace: To study the flow-level Internet traffic characteristics, we analyze the statistics of traces collected from CAIDA [6] and WIDE [15]. We select three destination subnets and extract the corresponding packets for 60 seconds. Before the performance evaluation, we would like to verify, based on the traces,

Table 3.1: Flow-Level Profile of Traffic Traces

Trace ID	# total flows	Flow size (Kbytes)			Mean through-put (Mbps)	Flow arrival rate (Kbps)			# flows having arrival rate larger than	
		Mean	Min.	Max.		Mean	Min.	Max.	5% of mean through-put	10% of mean through-put
1	6611	4.96	0.044	10655	4.37	5.46	0.016	5483	9	5
2	8403	128	0.06	233869	143.45	133.7	0.021	95946	19	6
3	3993	166.5	0.06	52069	88.67	163.2	0.017	17039	55	22

that the inaccuracy of splitting indeed exists and there is enough room to improve the accuracy. It was noted in [25, 66] that the skewness of flow size distribution is the main cause of splitting inaccuracy. The three traffic traces have different mean flow size and mean throughput as illustrated in Table 3.1, but their flow size distribution and flow arrival rate distribution follow a similar pattern. Fig. 3.7a shows that for each of the trace, over 70% of the flows have size less than the mean flow size, while the maximum flow size could be as large as 2000 times of the mean (for trace 1 and 2). The small portion of super large flows makes it hard for static traffic splitting to perform accurately. We also plot the cumulative distribution function of flow's arrival rate in Fig. 3.7b. The x-axis shows the ratio of the flow arrival rate to the mean throughput. Note that the mean throughput of a trace is the aggregated arrival rates for all flows over 60 seconds, and thus it is possible for a particular flow's arrival rate to exceed the mean throughput. We observe that for each trace there exists a small number of flows having arrival rates higher than 5% of the mean throughput.

We replay the traffic trace using the tool tcpreplay which reproduces the

packet-level synthetic traffic of the collected trace, both in terms of packet arrival time and packet size. We use tcprewrite to overwrite the IP Address and Ethernet Address of the packet's header with the appropriate addresses in our emulation network. The port numbers on the transport layer are kept unchanged in order to maintain the entire flow characteristics of the original trace. The traffic replay are injected into the network from the source host.

Testing scenarios: Our experiments are conducted in two scenarios. The path reassignment is necessary and valuable in either of the two cases: 1) when facing non-uniform flow size distribution, or 2) when the target split ratios are changed. In the first scenario, we assume the target split ratios are not updated so that our reassignment process dynamically improves the accuracy of splitting caused purely by the existing flows in the network. In the second scenario, we take into account the update of target split ratios which is reconfigured by dynamic TE and evaluate how our scheme reacts in real time.

Compared splitting schemes: We consider 5 splitting schemes in our evaluation.

RA-alg: the algorithmic approach to our rate-aware traffic splitting proposed in 3.2.3.

RA-opt: the approach that finds the optimal solution by solving the optimization problem (3.1).

OVS: the original weighted-splitting mechanism in group table implemented in OVS.

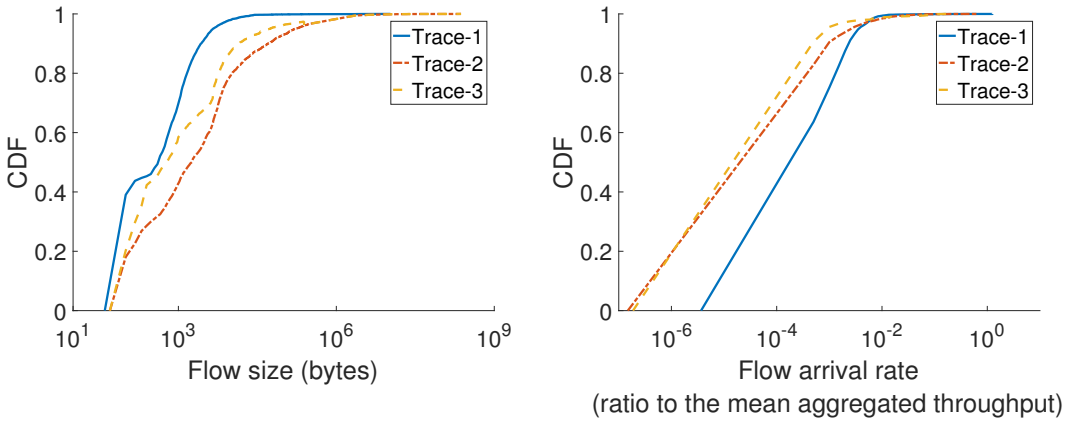
OVS-revised: the revised algorithm for OVS we described in Section 3.3.1.

MBD-/ADBR: a representative scheme of dynamic traffic splitting based on the load statistics [52]. It first disconnects multiple bins in the order of decreasing

size from each overloaded path until all paths are underloaded, and then reconnects each of the disconnected bins to the path with the largest underloaded.

Performance metrics:

The maximum overflow ρ and reassignment fraction Δ are considered as the two primary performance metrics. ρ is obtained by finding the maximum overflow fraction among the four paths. The reassignment fraction Δ is measured by summing up the fraction of traffic being shifted to a different path at each time when the reassignment is triggered. We also consider goodput and packet re-ordering fraction to evaluate the network performance. Goodput measures the application-level throughput that excludes retransmitted packets. Packet re-ordering fraction is the percentage of packets that do not advance the sequence number when arriving at the destination.



(a) CDF of flow size.

(b) CDF of flow arrival rate.

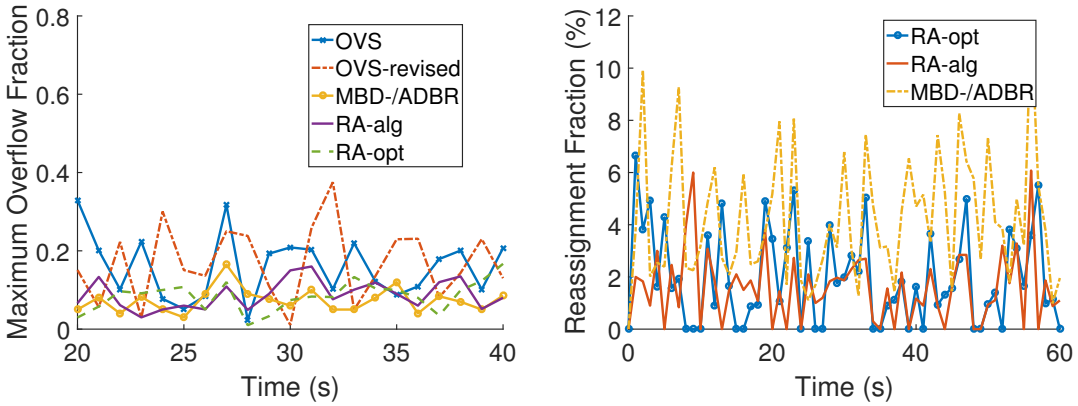
Figure 3.7: CDF of flow size and flow arrival rate for packet traces.

3.4.2 Results

Equal splitting: We first evaluate the performance for equal splitting which targets at splitting the traffic among the four available paths evenly. The desired split ratio for each path is 0.25. We replay the three 60-second traces and send all traffic from the source host subnet S_1 to the destination host subnet D_1. In RA-opt and RA-alg, we set λ to be 0.05 and K to be 6. The update time interval for RA-opt, RA-alg and MBD-/ADBR are all 1 second.

Fig. 3.8 shows the maximum overflow fraction and reassignment fraction over time for trace 1. The average performance of accuracy among all three traces are shown in Fig. 3.10. OVS and OVS-revised do not readjust the paths for existing flows. Due to the flow dynamics and skewed flow size distribution, the maximum overflow in these two schemes can be up to 0.4, as shown in Fig. 3.8a. The other three schemes dynamically readjust the path assignment based on the current load statistics, so the maximum overflow is significantly reduced when they are used. Fig. 3.8b shows the fraction of load being reassigned to different paths for MBD-/ADBR, RA-opt and RA-alg. MBD-/ADBR reassigns as much as 4.51% of total load on average because it redistributes the traffic load in a greedy manner and involves unnecessary reassignment. RA-opt redistributes the load based on the optimal solution, so it minimizes the maximum overflow with only around 2.11% load being reassigned. Compared to the optimal solution provided by RA-opt, RA-alg is more conservative and shows the least reassignment fraction of 1.95%. It still achieves similar level of accuracy as RA-opt and MBD-/ADBR but moves much fewer flows to different paths than MBD-/ADBR.

Unequal splitting: We next consider unequal splitting amongst the four



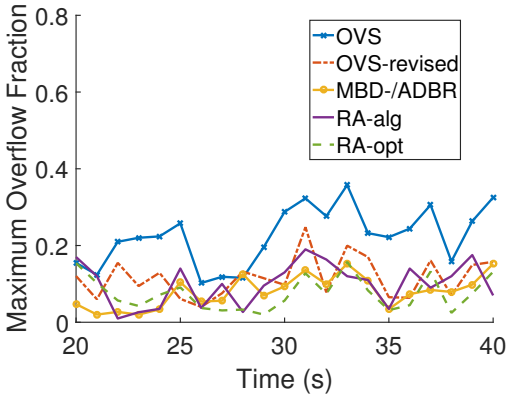
(a) Maximum overflow fraction.

(b) Reassignment fraction.

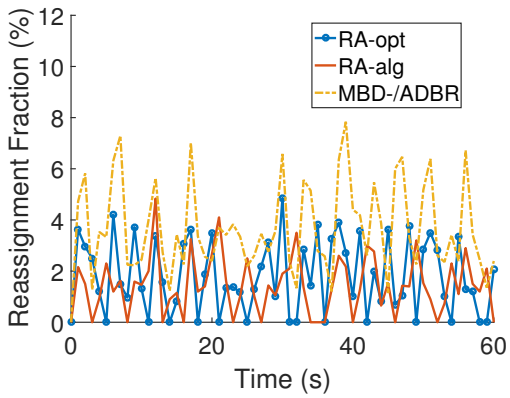
Figure 3.8: Equal splitting.

paths with split ratio 0.1: 0.2: 0.3: 0.4. As we discussed in Section 3.3.1, the current splitting algorithm implemented in OVS fails to perform correctly for weighted splits even when the flow sizes are homogeneous. This is verified in Fig. 3.9a and Fig. 3.10 which show much larger maximum overflow and split ratio deviation (measuring the sum of the load fraction's deviation from each path's target split ratio) for OVS than OVS-revised in the case of unequal splitting. We observe the similar performance comparison results as the equal splitting case: RA-opt achieves the lowest maximum overflow with around 1.87% reassignment fraction; the splitting accuracy for MBD-/ADBR is close to RA-opt but causes 3.15% reassignment on average; RA-alg has the lowest degree of reassignment which is about 1.75% and a bit larger maximum overflow and deviation of load distribution than RA-alg and MBD-/ADBR.

Impact of parameters: Two parameters in our model play important roles in the tradeoff between accuracy and reassignment degree: the reassignment time interval T and the weighting parameter λ. We evaluate their effects on the performance by tuning one parameter while fixing the other. The experiments are based on unequal splitting scenario with RA-opt and the traffic trace replay.

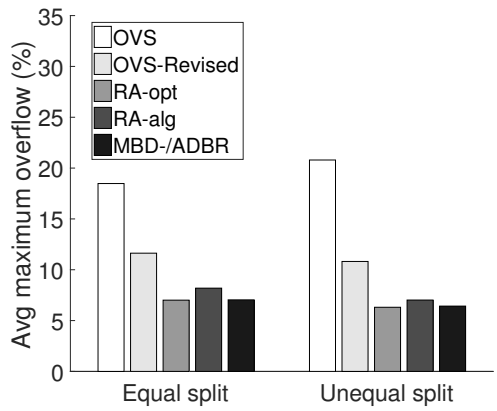


(a) Maximum overflow fraction.

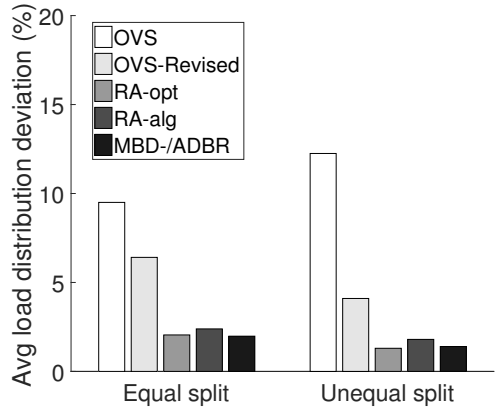


(b) Reassignment fraction.

Figure 3.9: Unequal splitting.



(a) Average maximum overflow fraction.



(b) Average deviation of load distribution.

	RA-opt	RA-alg	MBD-/ADBR
Equal split	2.11	1.95	4.52
Unequal split	1.87	1.75	3.15

(c) Average reassignment fraction.

Figure 3.10: Performance comparison summary.

The parameter λ quantitatively determines how much price the reassignment pays when reducing the level the overflow. $\lambda = 0$ means we do not consider any cost of reassignment. The larger λ, the more cautions to pay when making changes to the original assignment. Table 3.2 shows that as λ increases,

the average maximum overflow and thus the deviation of average load distribution become larger because it is more expensive to change the assignment. The average reassignment fraction consequently is decreased. The splitting accuracy and reassignment fraction is also affected by the reassignment time interval T, as is demonstrated in Table 3.3. Higher frequency helps to achieve smaller splitting errors at the price of causing more load to be reassigned.

Table 3.2: Impact of λ when $T = 1$ s

λ	0.01	0.05	0.08	0.1
Average maximum overflow (%)	2.02	6.23	7.31	8.75
Average deviation of load distribution (%)	0.63	1.5	2.76	3.21
Average reassignment fraction (%)	8.56	1.87	1.14	0.76

Table 3.3: Impact of T when $\lambda = 0.05$

T	0.5s	1s	5s	10s
Average maximum overflow (%)	3.75	6.23	8.52	9.34
Average deviation of load distribution (%)	0.84	1.5	2.97	3.35
Average reassignment fraction (%)	5.46	1.87	0.66	0.29

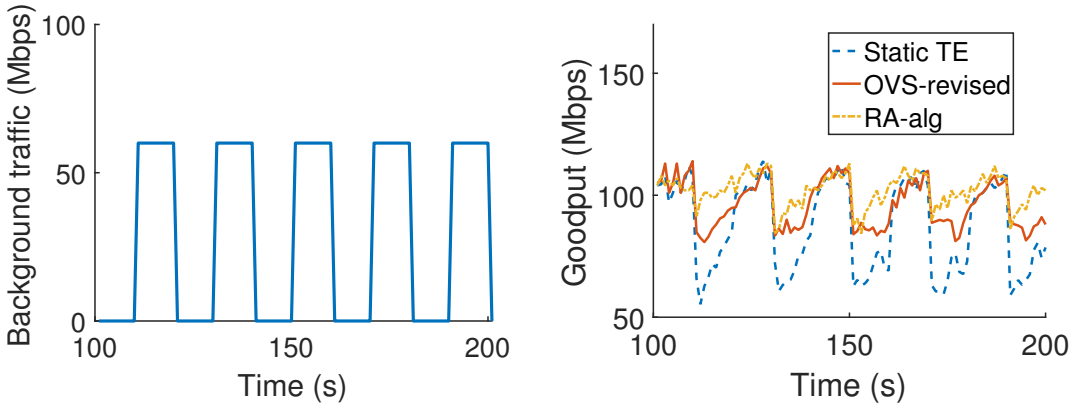
Dynamic splits: We further evaluate the end-to-end performance when dynamic TE is adopted. We consider the objective of dynamic TE as to balance the loss rate among multiple paths. To simplify the dynamic TE behavior, we use two paths with 40 ms and 100 ms round-trip time (RTT) in Fig. 4.3. Trace-driven experiments capture the characteristics of Internet traffic, but do not suffice to show the end-to-end network performance of traffic splitting schemes. We generate 50 TCP flows via iPerf from different hosts at the host subnet S_1 to the

destination subnet D_1. Additional delays ranging from 0 to 20 ms are randomly added for these flows in order to make the TCP flows heterogenous. Each buffer size is 3000 bytes. We further inject a 60 Mbps UDP flow as the background traffic at the host subnet S_2 with the dynamic pattern shown in Fig. 3.4.2. Dynamic TE collects the packet loss information for each interface and updates the target split ratios every 2 seconds via the group table API supported by OVS.

Fig. 3.11b and Table 3.4 show the end-to-end performance when dynamic TE works with different traffic splitting schemes. We also compare with static TE which does not react to the real-time loss to verify that dynamic TE indeed brings benefits. The static TE configures constant target split ratios 0.5:0.5 to be realized by OVS-revised in data plane. It is observed in Fig. 3.11b that when the large background flow is injected, the goodput under static TE scheme shows considerable degradation because it does not move away any TCP flows from the congested path. With dynamic TE, the goodput also drops as soon as the background traffic is added, but is able to climb up gradually since dynamic TE tries to maintain the loss fairness by changing the split ratios. It is further verified in Table 3.4. The aggregated loss rate sums up the loss rate of the buffer associated with each interface. The static TE has a much larger aggregated loss rate than the dynamic TE. The nonzero packets reordering rate for static TE shown in Table 3.4 is due to the multicore design of OVS.

When new target split ratios are configured by dynamic TE, OVS-revised redistributes the flows accordingly but the error between the actual load distribution and the target ones is not controlled. Therefore, the actual behavior that dynamic TE observes deviates from expected. This triggers the dynamic TE to further adjust the load distribution which makes the target split ratios possibly

fluctuate in a large range. That is mainly why the goodput of OVS-revised is lower and its loss rate and packets reordering rate is higher than other splitting approaches, as is shown in Table 3.4. RA-opt performs the most accurate splitting so the aggregated loss rate is the minimum and it achieves the highest goodput among all schemes. RA-alg has the minimum packets reordering rate, but its goodput is slightly lower than RA-opt because its splitting is not as accurate as RA-opt. MBD-/ADBR invokes more path reassignment than necessary, so high packets reordering rate is the main cause of its goodput degradation.



(a) Background traffic pattern.

(b) Goodput of the TCP flows.

Figure 3.11: Goodput when background traffic are added.

Table 3.4: End-to-end Performance

	Static TE	OVS-revised	MBD-/ADBR	RA-alg	RA-opt
Average goodput (Mbps)	86.2	95.6	98.5	102.6	103.4
Aggregated loss (%)	1.837	1.225	1.051	0.998	0.963
Reordering packets (%)	0.374	1.252	1.143	0.805	0.829

3.5 Related Work

We classify the existing traffic splitting models into two categories based on whether the current load information are required in their path selection methods.

Load-Unaware:

Packet-By-Packet Round-Robin: The simplest model of info-unware traffic splitting is packet-by-packet load balancing that selects individual packets amongst the alternative paths in round-robin [72] and weighted round-robin fashion [59].

Fast Switching: Cisco proposed fast switching [51] which restricts the size of lookup table by only storing recently seen flows in a cache. When the cache is used up, the oldest entry is deleted in favor of the new one. The performance of splitting varies depending on the size of the cache.

Hash-based: Hash-based approaches identify flow by applying hashing function to packets such that packets from the same flow have an identical hashing value. With the packet's hashing value, direct hashing approaches select the path taking modulo of the number of multiple paths [25]. Its main limitation occurs when a path is added or removed from the original path set, then a certain amount of flows are redistributed.

Masking Operations: [46] and [71] both exploit the bit-masking operations to achieve traffic splitting with finer granularity, relying on the the matching entry feature of OpenFlow.

Load-Aware:

Flowlet Aware Routing Engine (FLARE): [45] proposed FLARE that split a flow into flowlets based on the inter-arrival time of the flow. The timeout of a flowlet forwarding rule is set on the level of path RTT. The amount of traffic being reallocated mainly depends on the parameter of time threshold. In wide area networks, the inter-arrival timeout need to be large enough to maintain low risk of packet reordering, resulting in limited quantity of flowlets and limited room to adjust the load distribution.

Table-based Hashing with Reassignments: [27] performs adaptive load reallocation based on the table-based hashing. The redistribution decision is made by considering both the traffic load information and the inactive time for each bin. Only one bin being remapped at each time interval restricts the improvement of splitting accuracy.

Single/Multiple Bin Disconnection and Reconnection: [52] considers the bin disconnection and reconnection process separately. A set of greedy algorithms are proposed and evaluated that dynamically adjust the load distribution by bin disconnection and reconnection for single or multiple bins either in progressive or conservative manner.

3.6 Summary

In this chapter we propose a dynamic load distribution scheme that aims to accurately realize given traffic split ratios in switches with small performance degradation based on the collected load sharing statistics. It finds the most ac-

curate traffic splits with minimum route changes. Trace-driven and end-to-end experiments demonstrate that 1) our approach effectively adjusts load distribution in real time to mitigate the inaccuracy of splits caused by the variation of flow size distribution, 2) it outperforms the existing approaches with respect to both higher accuracy and lower level of route changes, and 3) it requires path changes for less flows when routing strategies are reconfigured, hence leads to better flow experience such as higher goodput.

CHAPTER 4

APPLICATION MANAGEMENT: PER-APPLICATION END-TO-END PATH SELECTION

4.1 Introduction

Although Internet Service Providers (ISPs) offer various Service Level Agreement (SLA) to guarantee the average internet performance in long term, most individual users and enterprises still suffer from performance degradation from time to time. Those performance degradations include low speed, large delay and high packet loss rate. Besides the known problems of today's Internet including sudden outages and unpredictable peering relationship changes [16, 12, 5, 1], the Internet traffic growing rapidly and more dynamically makes it more challenging for the ISPs to meet various Quality of Service (QoS) and user perception requirements for applications at all times. To improve the reliability of user's network experience, the very first question is how to realize the end-to-end control without which any effort to seriously boost network performance is bound to fail.

In the existing network infrastructure, to control the end-to-end user experience is non-trivial and highly complex because the Internet is an aggregation of a large number of networks owned by many ISPs with different economic interests [28]. One standard way to obtain end-to-end SLAs is to create private networks through business contracts among them. This obviously is costly and takes lots of time to realize. Alternatively, more and more companies today are considering simply using the Internet as the backbone for their WAN solutions [7, 13, 14], with dynamic management of multiple end-to-end paths established

by diverse providers. For example, enterprises would purchase Internet service from both AT&T and Comcast as the transport to build up their site-to-site connections. They can select the preferred connection dynamically based on the real-time SLA performance results monitored for both connections. This effectively enables the flexible selection among multiple alternatives, but the performance still relies on the Internet quality offered by each given ISP. The flexibility, availability and cost efficiency of improving the Internet performance is increasingly limited by the management and configuration complexity of Internet hardware routers produced by major vendors. The prevailing routing protocols being used in today's Internet such as the Border Gateway Protocol (BGP) are known to have limitations in realizing any performance-aware dynamic routing solution. BGP routing neither incorporates the information of path performance or link capacity, nor allows fine-grained overriding of BGP-specified forwarding behavior. With these severe constraints, network providers and operators face great difficulties in realizing reliable, high-performance end-to-end path management, let alone providing good per-application user experience.

In face of the above mentioned challenges in the Internet infrastructure, overlay architecture, which has been designed and developed for different purposes since a few decades ago [20, 39, 32], can be adopted as an enabler of achieving flexible control of end-to-end routing. It places a virtual network over the physical infrastructure, thus leveraging the dynamic control of network resources on an abstraction layer. Although there exist various overlay-based techniques of performance-aware routing on fast timescale [74, 63], they focus on routing inside the overlay network whereas the performance of the upstream and downstream path in the last mile for the end users is beyond their control. In the case where the communication between the end user and the

edge node is through the public Internet, the end-to-end performance can still be largely affected by any unpredicted delay and congestion over the Internet. If there exists diversity and flexibility in selecting the edge node, it is possible to achieve more reliable end-to-end performance.

In this chapter, we propose the design of a platform that allows flexible and responsive control of the end-to-end path per-application for enterprises' site-to-site Internet service. We present that exploring the geographical and physical diversity in the last-mile can effectively avoid the failure and congestion in the last-mile once being detected, and thus to a large extent provide end-to-end reliability and high performance. Furthermore, the application-based performance can be jointly optimized if we combine the real-time traffic conditions of both the core network and the last-miles to make the end-to-end routing decisions.

4.2 Design

4.2.1 Overlay

Figure 4.1 shows an example of the end-to-end path through an overlay network. The Point of Presence (PoP), as the interface point in each location between the overlay network and the end users, is responsible for assisting the application traffic to enter or leave the overlay core network. Traffic from end users are directed into the ingress PoP, forwarded through the virtual overlay networks to the egress PoP, and ultimately delivered to the destination. Naturally, we break down the end-to-end path into three parts: the virtual overlay network and the two last-miles referring to the connections between the edge

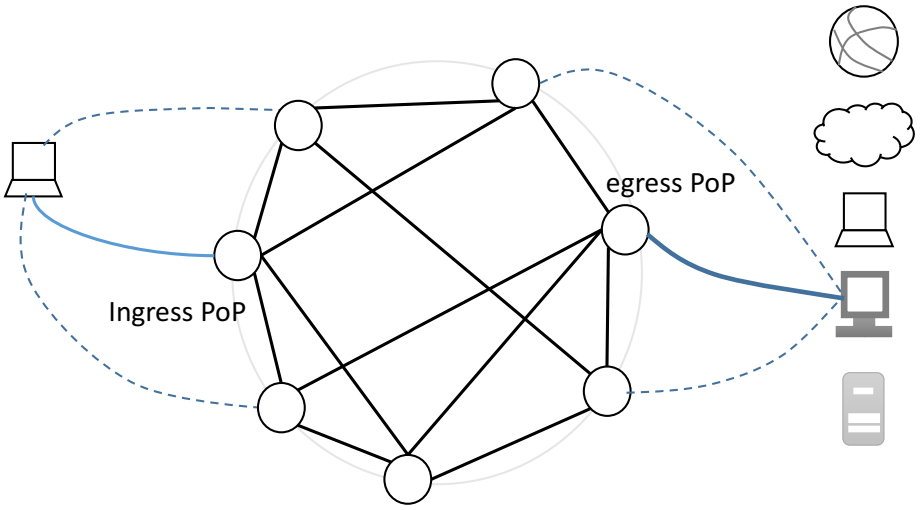


Figure 4.1: End-to-end path through an overlay network.

PoPs and the end users on both sides.

In the overlay network, routing within the core network is managed by the routing protocols implemented in the overlay networks, whereas the routes for the last-mile are beyond the control of the overlay network. Therefore, even if the core network performs application-aware routing management, it is non-trivial for the overlay network to react to the congestion or failure occurred in the last-mile and guarantee the end-to-end performance. For example, if the end user in Figure 4.1 experiences much lower throughput for a file downloading service and the congestion is detected to be in the last-mile connection between the ingress PoP and the end user, then changing the last-mile path may possibly improve the end-to-end performance. Therefore, we aim at developing a platform through overlay network that allows the flexible selection of the ingress and egress PoP among all available ones such that the end-to-end application-based performance can be optimized by jointly considering the performance of the last miles and the core.

4.2.2 Goals and Assumptions

Our design goal is to optimize the end-to-end application-based performance among the last-mile diversity. In particular, when the application-specific performance of the current path becomes consistently lower than some other path, the system is expected to quickly detect it and switch to the better path no matter the issue is caused by the core network or the last mile. This requires the system to have the ability of fine-grained performance monitoring, flexible routing control, and highly responsiveness to performance degradation.

We make the following assumptions in our design. First of all, we assume there are multiple end-to-end paths that satisfy the customer's performance requirement. Without this assumption, it is not possible to explore the last-mile diversity to guarantee reliable performance when the initial path fails. Secondly, the routing for the last mile between the end user and a given ingress/egress PoP is via the Internet which is beyond our control. Furthermore, we assume the connection of any application is through the domain name. The application of the traffic is not limited by web content, but arbitrary application such as file transfer, remote login, and real time streaming.

4.2.3 Application-based End-to-end Routing

Consider a core network with a set $\mathcal{P}$ of PoPs worldwide. For a given user with a pair of source s and destination d, we define a set of ingress PoP candidates $\mathcal{I}_s \in \mathcal{P}$ for the source s and a set of egress PoP candidates $\mathcal{E}_d \in \mathcal{P}$ for the destination d. The end-to-end path denoted as $p_{s,d}$ is a composition of three subpaths $p_{s,i}$, $p_{i,e}$ and $p_{e,d}$. We assume that inside the overlay network the route between

any ingress and egress PoP is controlled by specific performance-aware routing algorithm deployed. Note that the route of the last mile is beyond the control of the overlay network, so controlling the end-to-end routing turns to be determining the ingress PoP and egress PoP for a given source-destination pair.

The performance cost from s to d is dependent of the choice of the ingress PoP i and egress PoP e, denoted as $w_{s,d}^{i,e}$. The optimal path is selected by finding the optimal ingress and egress PoP, denoted as i^* and e^*, that result in the least path cost:

$$w_{s,d}^{i^*,e^*} = \min_{i \in \mathcal{I}_s, e \in \mathcal{E}_d} w_{s,d}^{i,e}.$$

We aim to ensure reliable performance per-application and dynamically update the routing to react to any performance degradation or failures in the currently selected path. Because applications come with their specific QoS and user perception requirements, the end-to-end routing decisions need to be made based on the performance metrics defined for the particular application. Here we discuss about the performance metrics for three different types of applications. The design could be easily expanded to applications that have other performance need.

Delay: For delay-sensitive applications, we consider the aggregated latency of the end-to-end path as the performance cost. Let $T_{s,d}^{i,e}$ denote the latency of the end-to-end path given the ingress PoP i and egress PoP e, and $t_{x,y}$ denote the latency that is actually measured between any arbitrary node x and y. The latency metrics $T_{s,d}^{i,e}$ are additive on the end-to-end path in the following form:

$$T_{s,d}^{i,e} = t_{s,i} + t_{i,e} + t_{e,d}.$$

Therefore, for delay-sensitive applications, the performance cost from source s to destination d through ingress PoP i and egress PoP e is given by $w_{s,d}^{i,e} = T_{s,d}^{i,e}$. The delay in the last miles are monitored from the source and destination end hosts. They use active probing to measure the latency in real time for each ingress/egress PoP candidate.

Loss: For loss-sensitive applications, we take the aggregate loss rate of the end-to-end path as the performance cost. Let $L_{s,d}^{i,e}$ denote the loss rate of the end-to-end path given the ingress PoP i and egress PoP e, and $l_{x,y}$ denote the loss rate between node x and y. Similarly, the loss rate in the last miles is measured via active probing from the end hosts. The aggregated loss rate $L_{s,d}^{i,e}$ on the end-to-end path is in the following form:

$$L_{s,d}^{i,e} = 1 - (1 - l_{s,i})(1 - l_{i,e})(1 - l_{e,d}).$$

Therefore, for loss-sensitive applications, the performance cost from source s to destination d through ingress PoP i and egress PoP e is given by $w_{s,d}^{i,e} = L_{s,d}^{i,e}$.

Throughput: For throughput-sensitive applications, we aim to ensure reliable performance of available TCP throughput. It is known that TCP exhibits complex behavior under window control and congestion control. The available bandwidth is affected by multiple factors including buffer limits, physical capacity and link characteristics. In path selection, only link conditions of latency and loss rate vary among multiple paths while holding all the physical conditions to be identical. As a result, the performance metrics we use in path selection for TCP throughput-o application is the relative evaluation of the available bandwidth given the current delay and loss conditions in each path. The performance metrics are based on the following well-known Mathis model [53]

$$Throughput = \frac{MSS \times C}{RTT \times \sqrt{loss}},$$

where C is a constant that incorporates the loss model and the acknowledgment strategy. The throughput is linearly proportional to $1/RTT \times \sqrt{loss}$ in the presence of packet loss. Here we only evaluate the TCP relative performance to compare among candidate paths, rather than computing the precise value. To find the optimal path that achieves highest TCP throughput, we consider $RTT \times \sqrt{loss}$ as the cost estimation for each end-to-end path.For any $i \in \mathcal{I}_s$ and $e \in \mathcal{E}_d$, the end-to-end path cost from s to d is given by

$$w_{s,d}^{i,e} = T_{s,d}^{i,e} \sqrt{L_{s,d}^{i,e}}.$$

Note that a limit exists in the granularity of loss rate being measured. If the interval of each probing is τ, then the loss rate that can be detected during probing time period Δ_p is no smaller than τ/Δ_p, which we denote as σ. As a result, any loss rate below σ is reported as 0. Since the throughput is linearly proportional to $1/RTT$ in the case of zero packet loss, we set a lower bound for the end-to-end aggregated loss rate to characterize the cost evaluation for zero loss:

$$L_{s,d}^{i,e} = \max\{1 - (1 - l_{s,i})(1 - l_{i,e})(1 - l_{e,d}), \sigma\}.$$

4.3 Implementation

4.3.1 Overall Architecture

Figure 4.2 shows the overall architecture of our end-to-end path selection system. It consists of two critical components: the end-user agent and the edge server. The edge server distributed in each PoP monitors the performance of the core and shares the statistics with the end-user agent. It also performs data

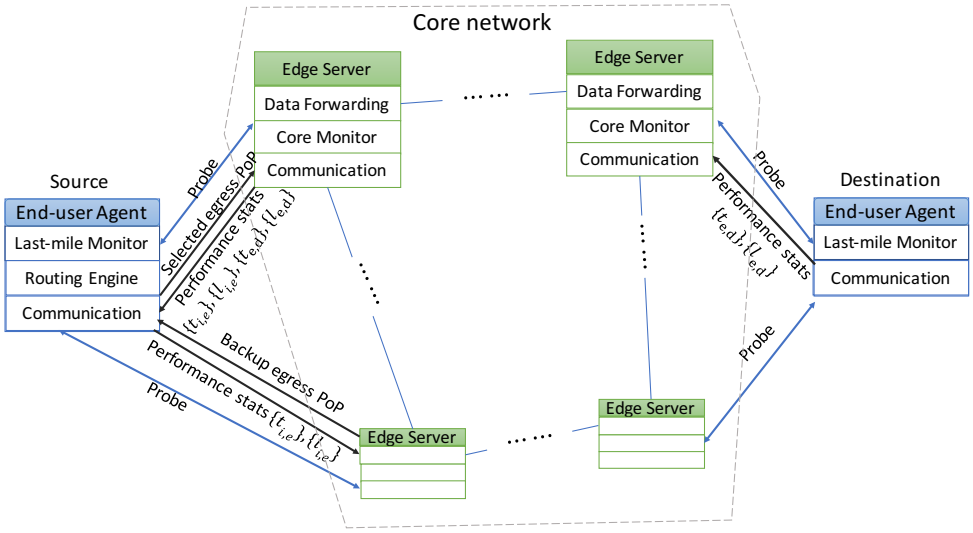


Figure 4.2: End-to-end path selection system architecture.

forwarding on the overlay network. The end-user agent is deployed in the end user's device, responsible for controlling the route, monitoring the last-mile performance, and communicating with the core network. The last-mile performance statistics monitored at the destination agent is propagated backward to the source end-user agent. As the end-user agent communicates with the edge server, decisions on which ingress and egress PoP to select can be made by the source agent based on the real-time understanding of the network.

Routing in the core relies on the specific implementation of the overlay networks which is independent of the selection of ingress and egress PoPs for end-to-end path routing. Being unaware of how traffic is routed in the core network, the end-user agent at the source determines the ingress and egress PoP based on the three sets of statistics collected. We separate the performance monitoring into three segments for the following reasons: it requires less overhead for performance monitor in the core network because the edge server only need to measure the performance per overlay link but not per flow in the core; it takes

less time to respond to outage and performance failure occurred in between the end user and the ingress PoP; it does not impose any limitations on the routing in the core, so routing in the core network can be flexibly controlled and customized by the overlay network.

4.3.2 End-user Agent

End-user agent is the key component for realizing flexible fine-grained routing control and real-time monitoring in the last mile of the end-to-end path. The agent is deployed in the end user's device, performing mainly three functions: end-to-end path selection, last-mile performance monitor, and communication with edge servers. We here describe each of the functions in detail.

Path Selection: The agent in the source end user controls the route for the end-to-end path, excluding the sub-route between the ingress PoP and egress PoP which is controlled within the core network independently. The routing control on the end-user agent involves three steps: determining the ingress and egress PoP, directing traffic to the correct ingress PoP and notifying the core network with the correct egress PoP. The ingress and egress PoP are determined by consulting the solutions for the optimal application-specific path described in Section 4.2.3. The source agent stores the delay and loss rate metrics collected for each subpath, as are required in the routing computation. Table 4.1 shows an example of the routing table maintained by the source agent with an update interval of Δ_r. It contains the *Current Route* being used with respect to the ingress and egress PoP, as well as the *Proposed Route* that has the least end-to-end path cost. In order to avoid path flipping, the new routing decision is made in a

conservative and cautious way such that the *Proposed Route* becomes the *Selected Route* only if either of the two conditions holds: it has a cost below 80% of the *Current Route*'s cost; or it is consistently the *Proposed Route* for the last three update cycles. Otherwise the *Selected Route* remains to be the *Current Route*. Furthermore, we insert the *Backup Route* into the routing table which represents the best egress PoP for each given ingress PoP other than the one associated with the *Selected Route*.

Table 4.1: Routing table maintained in the end-user agent.

Ingress PoP	Egress PoP	Cost	Current Route	Proposed Route	Selected Route	Backup Route
i_1	e_1	w^{i_1,e_1}	×	×	×	×
i_1	e_2	w^{i_1,e_2}	✓	×	×	✓
i_2	e_1	w^{i_2,e_1}	×	✓	✓	
i_2	e_2	w^{i_2,e_2}	×	×	×	

If an update of *Selected Route* involves the change of the ingress PoP, the end-user agent in the source will modify the local DNS hosts file with the corresponding ingress PoP IP address so that any traffic destined to the given destination are redirected to the ingress PoP provided. Besides attracting traffic to the correct ingress PoP, the agent need to notify the selected ingress PoP about the selected egress PoP, and notify each alternate ingress PoP about its corresponding *Backup Route* to have a default route installed. This notification messages are transmitted through the communication channels between the agent and edge servers.

Performance Monitor: End-user agent uses active probing to measure the RTT and loss rate between the source and ingress PoP, and between the destination and egress PoP. Originating the probing packets at the end-user agent

for the last-mile measurement prevents the probing from being blocked by the firewall in the end user. The edge server deployed in ingress and egress PoP is responsible for replying to the received probing packets. The probing packet is sent once every τ time interval. By checking the sequence numbers, the end-user agent measures the RTT and the round-trip loss rate based on the reply packets every Δ_p time period. $t_{s,i}$ and $t_{e,d}$ are computed by averaging the RTTs among all probing packets. $l_{s,i}$ and $l_{e,d}$ are viewed as the fraction of lost probing packets over all sent packets. Since we set symmetric routing for the forward and backward paths, it is reasonable to estimate the loss rate for the round trip.

Communication: The communication between the end-user agent and the edge servers include three types of control messages: handshake messages, measurement statistics and notification of egress PoP selection. At the startup stage of the end-user agent, the source agent discovers the alive edge servers via ping packets and selects the K closest PoP into the ingress PoP set $\mathcal{I}_s$ based on the RTTs of the ping packets. Similarly, the destination agent selects the set of egress PoP $\mathcal{E}_d$. The relationship of source and ingress PoP is confirmed by handshake messages exchange between the end-user agent and the edge server of the selected PoP. Once the control channel is established in the last miles, the source agent periodically pulls the measurement statistics of the core network from the edge servers of the ingress PoPs, and pushes notification of egress PoP selection to the corresponding edge servers. The last-mile performance statistics measured by the destination end-user agent are propagated back to the source end user via the current returning path.

4.3.3 Edge Server

The edge server is distributed in each edge PoP of the core network.

Performance Monitor: In the core network, edge servers do not measure the performance metrics for a particular user pair or application, but rather the performance of the overlay link between two PoPs which is determined by the current aggregated traffic. The edge server in each edge PoP monitors the latency and loss rate between itself and each other PoP using active probing. Because the underlying physical infrastructure is invisible for the overlay network, it is even more important to be responsive to failures or changes in the network conditions. The performance of the overlay link between any two PoPs is expected to be measured on fast timescale. Once receiving pull request from the source ingress PoP, the edge servers immediately reply back with the latest statistics.

Packet Forwarding: The other job of the edge servers is to perform data forwarding. Various techniques exist in realizing data forwarding through overlay networks. One approach being widely adopted is to encapsulate the IP packets with an overlay header that contains the address information of the overlay node. Another way is to manipulate the IP addresses in the original packet without inserting additional header. By changing the destination IP address of the original packet to the proper edge nodes, the traffic from end users are directed to the overlay network. Since the original destination is missing in the packet header, this approach requires a mapping between the source and the edge nodes to be stored in the edge nodes before user's traffic arrive. We use the latter mechanism in our experiments to avoid the additional overhead caused by encapsulation and decapsulation.

4.4 Evaluation

In this section, we evaluate our design of end-to-end path selection system. We describe the testbed setup and testing scenarios in Section 4.4.1, and present the experimental results in Section 4.4.2 that show the performance improvements achieved.

4.4.1 Evaluation Environment

Testbed setup: We set up a cross-continental overlay network shown in Figure 4.3 on Global Net [9], an ISP that offers commercial Internet service. The overlay network topology consists of 6 PoPs in different geographical locations, being connected via vxlan tunnel. Software switch is running in each of the PoP, and Open Shortest Path First (OSPF) is implemented as the routing protocol within the core network. As we mentioned above, routing inside the core network is invisible to the end-user agent. The end-user agent is deployed in the end users located in Oregon and London, and the edge server is running in each PoP. The link capacity is 100 Mbps for each link on the overlay network, less than the available bandwidth 200 Mbps in the last miles. We set probing interval τ to be 10ms, and both the measurement duration Δ_p and the agent's routing update interval Δ_r to be 30 seconds. Therefore the minimum detectable loss rate σ is 1/3000. We further choose the size of both the ingress PoP set and the egress POP set to be 2, to reasonably bound the last-mile delay. At the startup stage, the source agent discovers its ingress PoP set $\mathcal{I}_s$ to be {San Francisco, Chicago}. The destination agent discovers its egress PoP set $\mathcal{E}_d$ to be {London, Amsterdam}.

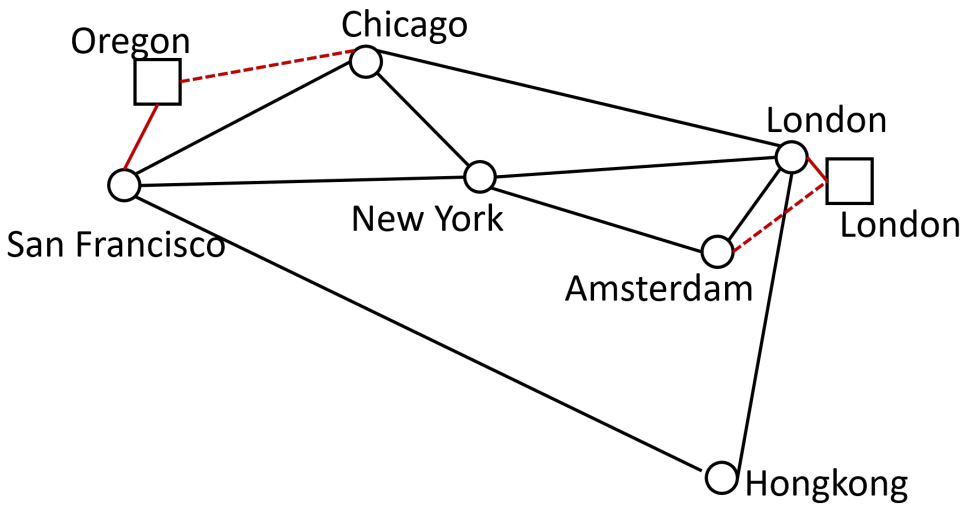


Figure 4.3: Network topology.

Testing scenarios: We conducted the evaluation from three aspects. First, we evaluated how our system reacts to network condition changes in the last mile and in the core by introducing emulated loss into the network. We used iPerf to send a single TCP flow from the source host in Oregon to the destination host in London, and measure the throughput. Our evaluation of the end-to-end throughput optimization considered network condition changes in either the last mile or the core network. We compared the throughput of our routing solutions against the throughput over a static path which selects the geographically closest ingress and egress PoP which is San Francisco and London respectively.

Secondly, we evaluated how much improvement our system makes by optimizing the end-to-end performance compared to optimizing the last miles and the core separately. We added disruptor into the core network with either a loss increase or a node failure, and let the core network react to the network condition changes. In the case that the last miles and the core network are optimized separately, the ingress and egress PoP selection is only affected by the network

condition changes in the last mile, but does not rely on the disruptor we added. We compared its behavior and performance results with the end-to-end optimization in our system.

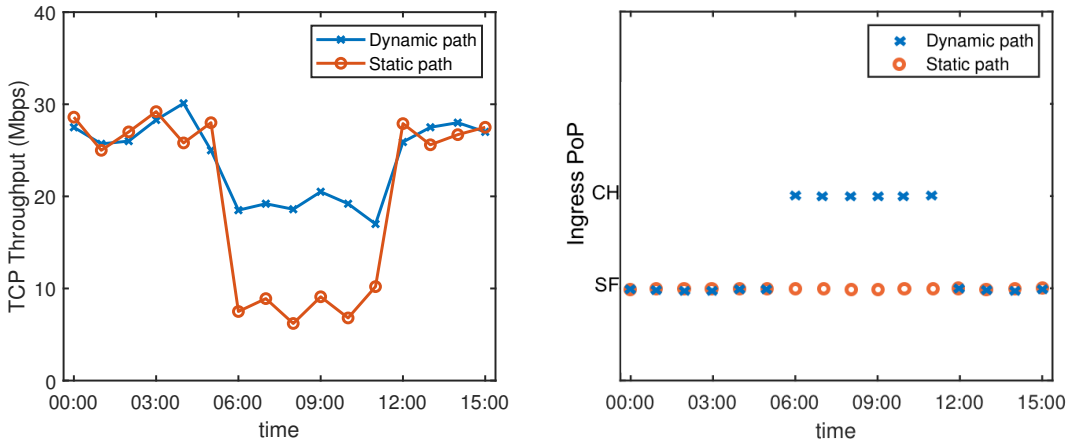
Thirdly, we evaluated how the system improves the user's application experience. We considered service of file uploading and take the transfer speed and transfer time as the performance metrics of interest. Multiple file transfer sessions were ensured to be running concurrently and sharing the network so that we could test how the path selection system responds to the current end-to-end path conditions and how much it could improve the file uploading performance.

Lastly, we evaluated the system's behavior when there are diverse applications sharing the network. We repeated the experiments of file transfer and additionally injected three video streams into the network from the source host to the destination host. We considered the video streams as latency-sensitive traffic which desire the path with smallest end-to-end latency. Therefore, the initial ingress and egress PoP for the video streams were San Francisco and London.

4.4.2 Results

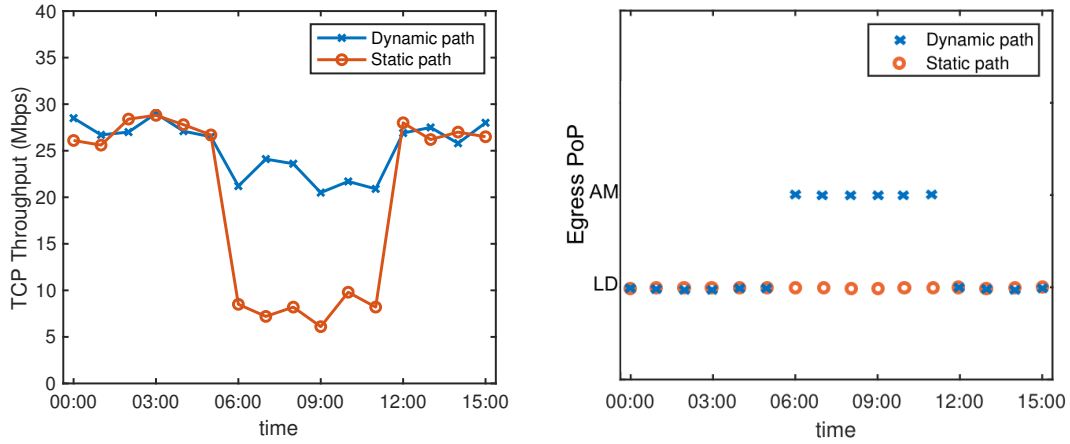
Loss increase in the network:

We evaluate the effectiveness of our system when facing loss in the current last mile. Internet is possible to suffer from loss at any point of the last mile, but here we simplify the testing scenarios by simulating the loss at the interface of the edge PoP to present the proof of concept. The loss is simulated using



(a) TCP throughput. (b) Selection of ingress PoP.

Figure 4.4: End-to-end throughput in the presence of 0.1% loss between the source and San Francisco PoP.

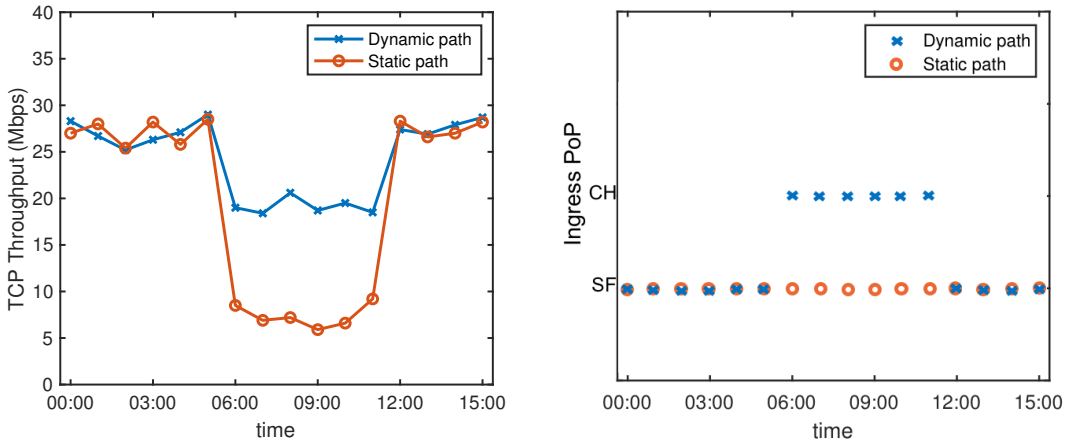


(a) TCP throughput. (b) Selection of egress PoP.

Figure 4.5: End-to-end throughput in the presence of 0.1% loss between London PoP and the destination.

the netem [10] functionality provided by Linux tc tool. The measurement of TCP throughput for the current selected path is sampled every hour by running iPerf for 200 seconds.

In the first scenario we imposed loss onto ingress PoP. Figure 4.4a shows the throughput results of each measurement sampling. At the beginning, the se-



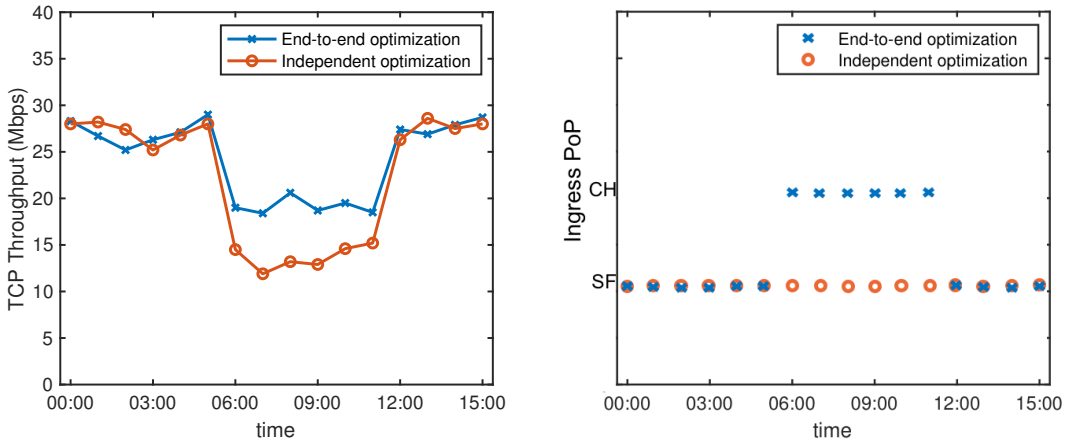
(a) TCP throughput.

(b) Selection of ingress PoP.

Figure 4.6: End-to-end throughput in the presence of 0.1% loss in New York PoP.

lected ingress PoP was consistently San Francisco. 0.1% loss was added to San Francisco at time 6:00. In the presence of loss, the throughput of the original path significantly dropped down to less than 10 mbps. With the ability of monitoring the real-time performance and exploring the route diversity, our system changed the ingress PoP from San Francisco to Chicago, as is shown in Figure **??**. The performance degradation after the path change was mainly caused by the delay increase in the new path. We removed the simulated loss rate at time 12:00 and the throughput after that went up because the path was changed to the original one. So Figure 4.4a further demonstrates that our system is also able to improve the performance by finding better path if existing.

In the second scenario, we considered adding 0.1% loss on the current egress PoP (London) between time 6:00 and 12:00. As illustrated in Figure 4.5a, the above 70% throughput drop for the static path is similar to that in the first scenario. The destination agent detected the loss in the last mile within 30 seconds. With the knowledge of the loss increase in the current path, the source agent se-



(a) TCP throughput.

(b) Selection of ingress PoP.

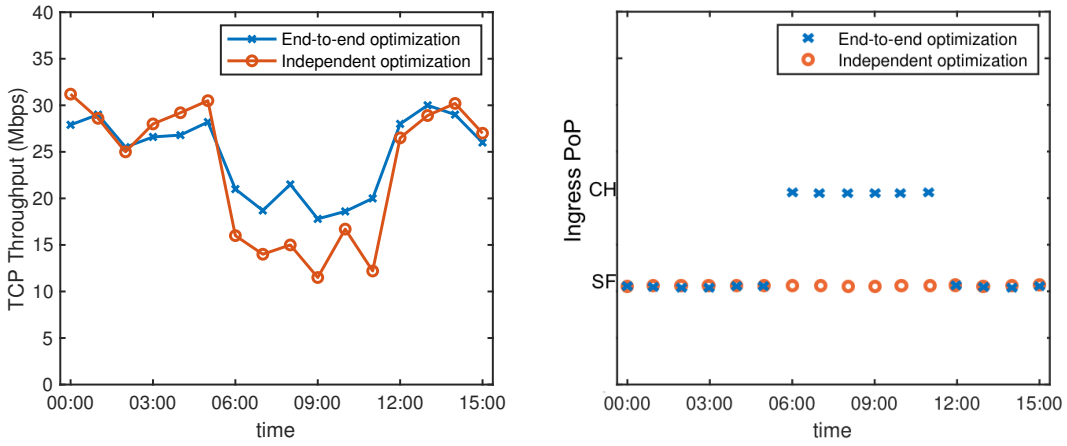
Figure 4.7: End-to-end throughput compared to independent optimization in the presence of 0.1% loss in New York PoP.

lected an alternate path with San Francisco as the ingress PoP and Amsterdam as the egress PoP (in Figure 4.5b), resulting in around 10% throughput decrease on average due to the delay growth.

In the third scenario, we considered the loss increase in the core network. Although the routing inside the core network is uncontrollable and invisible for the end-user agent, it strives to select the best pair of ingress and egress PoP among all candidates, with the real-time information of latency and loss rate between each pair collected from the edge servers. Figure 4.6a shows the throughput results when we simulated 0.1% loss in New York PoP from time 6:00 to 12:00. Our end-user agent in the source learned about the loss increase in the subpath and selected an alternate path with Chicago as the ingress PoP and London as the egress PoP (in Figure 4.6b).

End-to-end optimization versus separate optimization:

Our system achieves the end-to-end optimization by jointly considering the



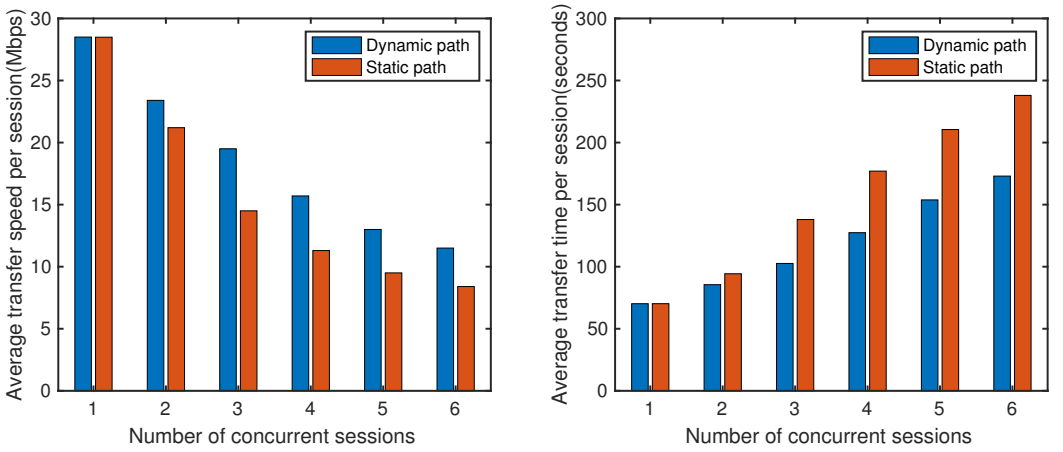
(a) TCP throughput.

(b) Selection of ingress PoP.

Figure 4.8: End-to-end throughput compared to independent optimization in the presence of failure in New York PoP.

last miles and the core network rather than optimizing them separately. We evaluate the performance improvement made by the end-to-end optimization over the independent optimization in the last miles and the core. Our evaluation in this part involves scenarios of unexpected loss increase and link failure in the core network. We assumed the routing strategy implemented in the core network was able to detect loss and failures, and reroute to the optimal path inside the core network. In the case of independent optimization, the selection of ingress and egress PoP is purely determined by performance in the last miles, and thus remains the same as long as no significant condition changes occur in the last miles.

When we added 0.1% loss in New York PoP, the original route between San Francisco PoP and London PoP via New York PoP is no longer the optimal. Therefore, the core network reroutes the traffic through Chicago. With the knowledge of the updated performance metrics for each candidate pair of ingress and egress PoP, the end-user agent in the source selects the optimal pair



(a) Average transfer speed for one session. (b) Average transfer time for one session.

Figure 4.9: Performance of file transfer as the number of sessions changes.

to be Chicago and London (in Figure 4.7b). Since the ingress and egress PoP remained to be San Francisco and London for independent optimization, the throughput is more than over 20% lower than that of the end-to-end optimization, as is shown in Figure 4.7a. When we failed New York PoP, the core network again rerouted the traffic from San Francisco PoP to London PoP via Chicago to avoid the failed node. For the end-to-end optimization, the source agent updated the ingress PoP to be Chicago once finding this path had a smaller end-to-end path cost (in Figure 4.8b). Similarly, Figure 4.8a shows that the new path selected by the agent outperforms the independent optimization solution by around 20% higher throughput.

Multiple file transfer sessions:

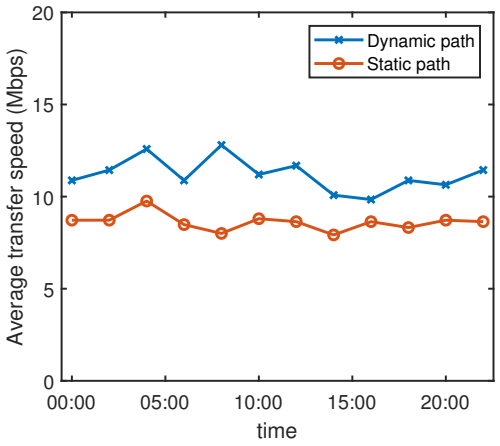
We next evaluated how the system improves the user's experience of file transfer service. In order to evaluate the application-level performance-based path selection, we continuously initiated a new session for uploading a file of size 2 Gbits from the source host to the destination host every 20 seconds while

the previous sessions are still in progress. Figure 4.9 demonstrates that for both the case of static path and dynamic path, as more sessions were running concurrently and competing the bandwidth, the transfer speed for each session would become lower and it would take longer time for each session to complete. In the case of static path, being unaware of the link condition, the new flows continued using the initial path with San Francisco and London as the ingress and egress PoP. Therefore as new flows entered the bottleneck link, all existing sessions would suffer from significant performance degradation triggered by loss increase. However, with the knowledge of the end-to-end loss increase on the initial path, our system would avoid selecting that path for the new flows and find the best ingress and egress PoP with least path cost. Therefore, our system offered much higher file uploading speed and less uploading time than the static path solution shown in Figure 4.9a and Figure 4.9b.

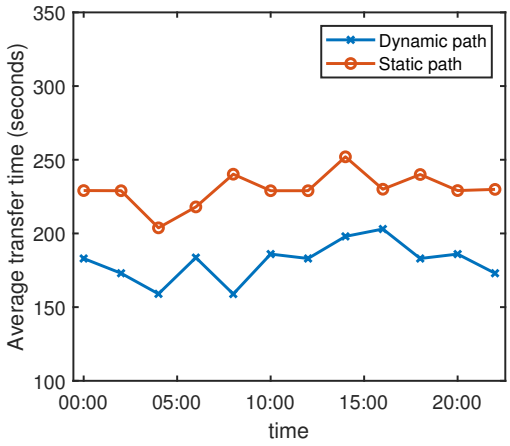
We also show in Figure 4.10 the average transfer speed and the average transfer time per session when there are six sessions running concurrently. We repeated the test every 2 hours for one day, and the results in Figure 4.10a show that the average transfer speed achieved by our system is around 30% higher than the static path. Similarly, each transfer session would take around 30% less time to complete with our dynamic path selection, as is shown in Figure 4.10b. Figure 4.10b demonstrates the number of file transfer sessions that were routed through San Francisco as the ingress PoP for each test, and the others were routed through Chicago PoP.

Diverse applications:

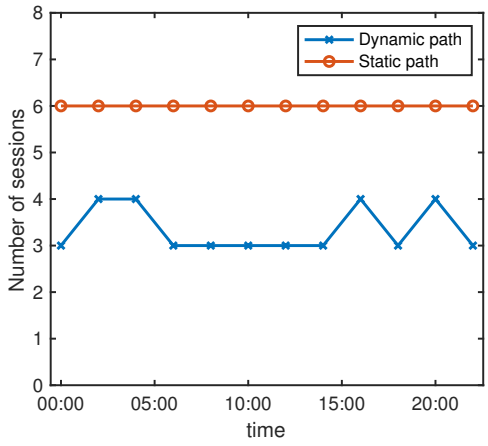
Lastly, we added diverse applications into the network. We started three video streaming via VLC from the source to the destination host, and repeated



(a) Average transfer speed for one session.



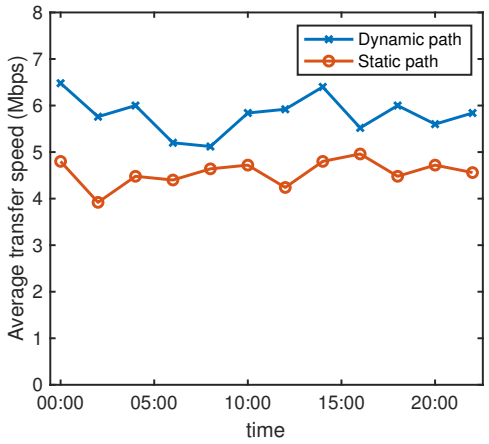
(b) Average transfer time for one session.



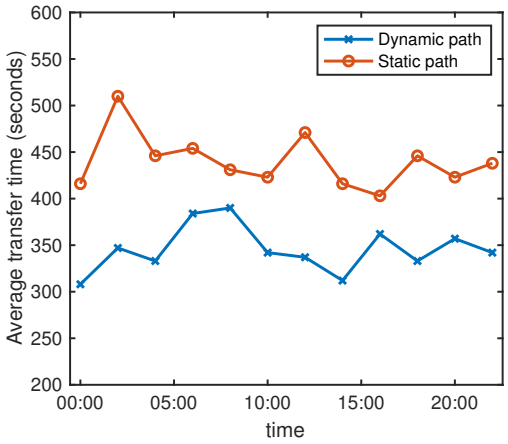
(c) Number of sessions routed through San Francisco as the ingress PoP.

Figure 4.10: Performance of six concurrent file transfer sessions.

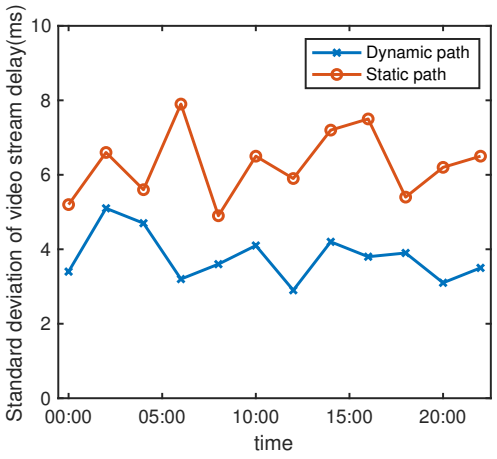
the file uploading tests. The total transmission rate for the three video streams were varying in real time but keeps below 15 Mbps. Latency was chosen as the performance metrics for the video streams. During the tests, the end-user agent dynamically selected the end-to-end path for the two applications independently based on their corresponding performance metrics, whereas in the case of static path all flows used the San Francisco and London as the ingress and egress PoP. As is shown in Figure 4.11, with the time-varying video traffic



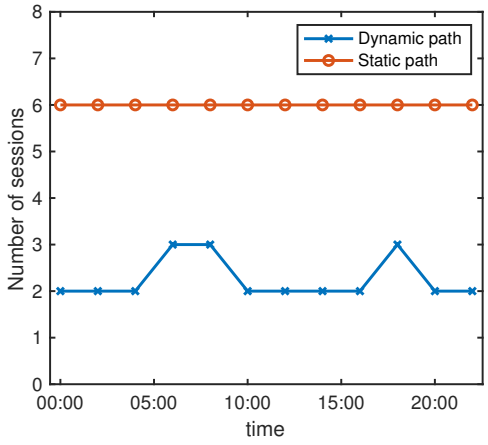
(a) Average transfer speed for one session.



(b) Average transfer time for one session.



(c) Standard deviation of delay for video streams.



(d) Number of sessions routed through San Francisco as the ingress PoP.

Figure 4.11: Performance of six concurrent file transfer sessions and video streams when sharing the bandwidth.

additionally sharing the bandwidth of the overlay links, the average file transfer speed per session slowed down and the average transfer time increased for both the case of static path and dynamic path compared to the results in Figure 4.10. However, our system made the path decisions by exploring the per-application costs for all the available paths so that the resource competition among the applications and multiple sessions was mitigated. Figure 4.11d shows that our system routed only 2 or 3 file transfer sessions through San Francisco as the ingress

Pop, and the rest of them were routed through Chicago to avoid the congestion. As a result, both the file transfers and video streams experienced significantly better performance in terms of higher average transfer speed (Figure 4.11a), less transfer time (Figure 4.11b) and lower standard deviation of video stream delay (Figure 4.11c) than the static path solution.

4.5 Related Work

Overlay networks: various overlay networks have been developed for different purposes since a few decades ago [20, 39, 32]. [20] proposed the Resilient Overlay Network (RON) that could fast recover from failures and performance issues of the Internet by taking advantage of underlying Internet redundancy. Although the overlay network is well developed, the existing performance-aware routing techniques via the overlay network are mostly dedicated to the core network without jointly considering the last miles. As overlay network is widely used particularly in Content Delivery Networks (CDN) such as [3, 4], end-to-end reliable performance is a major design goal that most CDNs focus on. However, their techniques are not applicable to the end-to-end performance problem with a pair of end users, because the servers under this context belong to a part of the CDNs which are controlled within the core networks rather than end hosts outside. Furthermore, the solutions to CDNs only apply to web-based applications traffic.

Routing within the core network is managed by the routing protocols implemented in the overlay networks, whereas the routes for the last-mile are beyond the control of the overlay network. [74] and [63] explore the diversity of

the routes, provided by multiple ISPs, between the end user and a fixed PoP in peering edge architecture. For example, they can select the preferred the routes Our approach selects the route among multiple PoPs and jointly considers the performance of the last miles and the core, which is not only different but also complimentary with their solutions. Moreover, it is applicable to other architectures without multiple routes options between the end user and one PoP.

Attracting traffic to the core: the state-of-art approaches to attracting end-user traffic to the core network include DNS [58, 36] and Anycast [47, 23]. Most DNS services do not consider the real-time performance when deciding the ingress PoP. Even for those who consider performance metrics such as latency, the measurement is done on coarse scale in both temporal and spatial dimension (at most once per day, only measure latency between regions). Furthermore, the DNS service is not aware of the end-user rather than the local DNS resolver on behalf of the end user which could be geographically far away or experience completely different performance. The server will make decision on the ingress PoP for each resolver rather than the actual clients, so the performance information for the client is not possible to be mapped to the resolver. One solution is the EDNS client-subnet-prefix standard (ECS) [29] that allows the resolver to specify a prefix of the client's IP when requesting domain name translations on behalf of a client. This enables the end-user mapping on the level of client subnet, but it does not consider the specific end-to-end performance for each end user [26]. Moreover, the proposed protocol is far from being adopted by all ISPs. Anycast approach makes the end user's IP address to be transparent. However, it lacks the flexibility of performance-aware routing control because the performance metrics used in BGP protocol do not represent the real-time performance information.

SDN and cloud networking: [38, 37] designed an architecture for traffic forwarding among multiple autonomous systems. Their design of SDN-based exchange points provides innovative solutions to applications such as inbound traffic engineering, wide-area server load balancing, application-specific peering and redirection through middlebox. We both exploit the benefit of SDN specifically from the aspect of route efficiency improvement, but for different use cases. They provide a way for IXP participants to compose general policies and forward traffic between autonomous systems. However, we aim at facilitating a particular carrier to optimize the route by selecting the preferable ingress and egress edge routers.

[65, 43, 64] proposed the cloud-based middlebox infrastructure for middlebox service providers that offer services in the cloud such as HTTP proxies, SIP proxies, WAN optimization, deep-packet-inspection, content caching, and content transcoding. Similarly they take DNS-based redirection approach to bring traffic to cloud provider PoPs. Unlike the application-specific metrics we define based on performance requirements and user experience, they generically choose latency as the metric for PoP smart selection. Their architecture aims at facilitating middlebox service providers to outsource enterprise middlebox processing to the cloud which mainly involves the traffic between enterprise and the cloud applications, whereas we focus on providing platform for the carriers to deliver the enterprises' site-to-site traffic.

4.6 Summary

We propose a platform for end-to-end path selection in overlay network architecture. With the knowledge the network topology and conditions, it strives to achieve the optimal end-to-end application-based performance by exploring the last-mile diversity. It allows the flexible and responsive per-end-user selection of the edge node for the overlay networks, and thus can fast recover from network failures and performance degradation. We present our design of the end-to-end per-application performance optimization system with detailed discussion of each component including dynamic routing engine, performance monitor and information exchange. Our experimental results demonstrate that the application-based end-to-end performance can be significantly improved in face of network condition changes in both loss and latency.

CHAPTER 5

FUTURE WORK

In this dissertation, we explored traffic management in time, space and application dimension. We described how to realize and analyze adaptive traffic engineering in high frequency, how to split traffic accurately in fine granularity, and how to achieve end-to-end path optimization per application. Further, there remains many interesting research topic along these directions.

5.1 Responsive Traffic Engineering

We provided the framework for analyzing the adaptive traffic engineering that performs in high frequency and studied the tradeoff between stability and responsiveness. A TE algorithm performing at fast timescale benefits from being responsive to varying traffic demand and network conditions, but it also means frequent routing changes. Any update in routing strategy with respect to a given commodity flow requires certain portion of traffic to be moved from their original paths to new ones. Under the stability condition, the TE solutions are guaranteed to eventually stabilize and converge to the optimal set point with finite steps. However, different trajectories that the TE solutions follow lead to different convergence time and degree of path changes. As moving traffic from one path to another can potentially cause packets reordering and thus performance degradation, it is ideal to keep existing traffic stick to their original paths unless changing is indeed inevitable or worthwhile. The future direction here is to design responsive TE algorithms that adaptively converge to the optimal routes in response to the current network states while effectively minimize the path changes for traffic involved in the routes update. With the goal of achiev-

ing the responsiveness to the dynamics of the networks and fast convergence, over-reacting can easily occur if the route update is not cautious enough. In face of the changes in network states, the TE algorithm is expected to set the best trajectory from the old set point to the new one based on the convergence speed as well as the overshoot. The goal of the design is to avoid overshoot of traffic amount for each path and unnecessary path oscillations over multiple steps, instead of merely optimizing convergence speed.

5.2 Fine-grained Accurate Traffic Split

We proposed a rate-aware flow-level traffic split mechanism for the data plane. Our design assumes that the target multipath routing solution for each commodity flow is given as the input, which is computed by TE algorithms in the control plane. The goal is to split the traffic accurately following the given split ratios for every commodity flow with small cost of flow route changes. Due to the fact that traffic are aggregated into the pipe, the deviation from target split ratios for different commodities may well possibly complement each other from the perspective of the aggregate link load, which is what ultimately matters. In other words, we may not need to get per commodity flow split very accurately but still achieves accurate aggregate load balancing. Therefore another future direction along this line is to answer the question of when it is necessary to achieve accuracy for split ratios per commodity flow. With the ultimate goal of mitigating the load balancing degradation and congestion caused by inaccurate traffic split, it is equivalently effective to realize aggregated split ratios accurately for each link, under the constraint that the QoS requirements are satisfied for each commodity flow. Moreover, the overall performance should be

even better because the routes adjustment is further minimized while the level of accuracy for the fraction of traffic on each aggregated link retains.

5.3 End-to-end Optimization for Cloud Applications

We presented our design of the end-to-end performance optimization system that mainly focuses on branch-to-branch traffic for enterprises. It allows per-application selection of the ingress and egress PoPs which are the edge nodes for the overlay network. The design has assumed that the end-user agent can be implemented in the enterprise end users at each branch. Another major type of Internet traffic is cloud application traffic, in which case the servers are usually hosted in the network of the cloud service providers while the clients are the remote end users. With such feature, the cloud application traffic gains more flexible control and management for the routing and monitoring than branch-to-branch traffic. As the cloud applications are growing rapidly, it is an interesting research direction to investigate how the platform can be extended to perform in the cloud network and optimize the performance of the cloud application traffic, especially how it can be applied to the prevailing infrastructure of cloud service providers such as Google, Facebook, etc. In such large-scale networks with diverse applications, scalability of routing and packet forwarding scheme requires great attention in the design.

BIBLIOGRAPHY

[1] 14,000 incidents: a 2017 routing security year in review. https://www.manrs.org/2018/01/14000-incidents-a-2017-routing-security-year-in-review/.

[2] 2017 internet society global internet report: Paths to our digital future. https://future.internetsociety.org/wp-content/uploads/2017/09/2017-Internet-Society-Global-Internet-Report-Paths-to-Our-Digital-Future.pdf.

[3] Akamai CDN. https://www.akamai.com/.

[4] Azure CDN. https://azure.microsoft.com/en-us/services/cdn/.

[5] Bgp hijinks and hijacks - incident response when your backbone is your enemy. https://www.giac.org/paper/gcih/20548/bgp-hijinks-hijacks-incident-response-backbone-enemy/123924.

[6] The CAIDA UCSD Anonymized Internet Traces - February 2012. http://www.caida.org/data/passive/passive_2012_dataset.xml.

[7] Cisco SD-WAN. https://www.cisco.com/c/en/us/solutions/enterprise-networks/sd-wan/index.html.

[8] Cisco visual networking index: Forecast and methodology, 2016-2021. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html.

[9] Global net. https://www.theglobal.net/.

[10] Netem. https://wiki.linuxfoundation.org/networking/netem.

[11] Open vswitch – an open virtual switch. http://www.openvswitch.org.

[12] The outages archive. https://puck.nether.net/pipermail/outages.

[13] Sprint SD-WAN. https://business.sprint.com/solutions/software-defined-wan/.

[14] White paper: Understanding sd-wan managed services. https://wiki.mef.net/display/CESG/SD-WAN+Managed+Services.

[15] Wide backbone traffic traces. http://mawi.wide.ad.jp/mawi/samplepoint-F/2017/201701101400.html.

[16] Giuseppe Aceto, Alessio Botta, Pietro Marchetta, Valerio Persico, and Antonio Pescapé. A comprehensive survey on internet outages. *Journal of Network and Computer Applications*, 2018.

[17] Sugam Agarwal, Murali Kodialam, and TV Lakshman. Traffic engineering in software defined networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 2211–2219. IEEE, 2013.

[18] Ian F Akyildiz, Ahyoung Lee, et al. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 2014.

[19] Mohammad Alizadeh et al. CONGA: Distributed congestion-aware load balancing for datacenters. In *SIGCOMM*, 2014.

[20] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. *Resilient overlay networks*, volume 35. ACM, 2001.

[21] David Applegate and Edith Cohen. Making routing robust to changing traffic demands: algorithms and evaluation. *IEEE/ACM Transactions on Networking*, 2006.

[22] Daniel O Awduche. MPLS and traffic engineering in IP networks. *Communications Magazine, IEEE*, 1999.

[23] Hitesh Ballani and Paul Francis. Towards a global ip anycast service. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 301–312. ACM, 2005.

[24] Matthew Caesar and Jennifer Rexford. Bgp routing policies in isp networks. *IEEE network*, 19(6):5–11, 2005.

[25] Zhiruo Cao, Zheng Wang, and Ellen Zegura. Performance of hashing-based schemes for internet load balancing. In *INFOCOM 2000. Nineteenth*

Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. IEEE, 2000.

[26] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 167–181. ACM, 2015.

[27] Tat Wing Chim, Kwan L Yeung, and King-Shan Lui. Traffic distribution over equal-cost-multi-paths. *Computer Networks*, 2005.

[28] David Clark. The design philosophy of the darpa internet protocols. *ACM SIGCOMM Computer Communication Review*, 18(4):106–114, 1988.

[29] Carlo Contavalli, Wilmer Van Der Gaast, D Lawrence, and Warren Kumari. Client subnet in dns queries. Technical report, 2016.

[30] C Desoer and Yung-Terng Wang. On the generalized nyquist stability criterion. *IEEE Transactions on Automatic Control*, 25(2):187–196, 1980.

[31] Anwar Elwalid et al. MATE: MPLS adaptive traffic engineering. In *INFOCOM*, 2001.

[32] Hans Eriksson. Mbone: The multicast backbone. *Communications of the ACM*, 37(8):54–60, 1994.

[33] Simon Fischer et al. REPLEX: dynamic traffic engineering based on wardrop routing policies. In *CoNEXT*, 2006.

[34] Bernard Fortz and Mikkel Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 2004.

[35] Robert G Gallager. A minimum delay routing algorithm using distributed computation. *IEEE transactions on communications*, 1977.

[36] Ivano Guardini, Paolo Fasano, and Guglielmo Girardi. Ipv6 operational experience within the 6bone. In *Proc. Internet Society (INET) Conf*, 2000.

[37] Arpit Gupta, Robert MacDavid, Rüdiger Birkner, Marco Canini, Nick Feamster, Jennifer Rexford, and Laurent Vanbever. An industrial-scale software defined internet exchange point. In *NSDI*, volume 16, pages 1–14, 2016.

[38] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. Sdx: A software defined internet exchange. *ACM SIGCOMM Computer Communication Review*, 44(4):551–562, 2015.

[39] RA Hagens, NE Hall, and MT Rose. Use of the internet as a subnetwork for experimentation with the osi network layer. Technical report, 1989.

[40] Frank E Heart, Robert E Kahn, Severo M Ornstein, William R Crowther, and David C Walden. The interface message processor for the arpa computer network. In *Proceedings of the May 5-7, 1970, spring joint computer conference*, pages 551–567. ACM, 1970.

[41] Chi-Yao Hong et al. Achieving high utilization with software-driven WAN. In *SIGCOMM*, 2013.

[42] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 2013.

[43] Keon Jang, Justine Sherry, Hitesh Ballani, and Toby Moncaster. Silo: Predictable message latency in the cloud. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 435–448. ACM, 2015.

[44] Srikanth Kandula et al. Walking the tightrope: Responsive yet stable traffic engineering. In *SIGCOMM*, 2005.

[45] Srikanth Kandula, Dina Katabi, et al. Dynamic load balancing without packet reordering. *ACM SIGCOMM Computer Communication Review*, 2007.

[46] Nanxi Kang, Monia Ghobadi, et al. Efficient traffic splitting on commodity switches. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2015.

[47] Dina Katabi and John Wroclawski. A framework for scalable global ip-anycast (gia). *ACM SIGCOMM Computer Communication Review*, 30(4):3–15, 2000.

[48] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[49] Murali Kodialam et al. Oblivious routing of highly variable traffic in service overlays and IP backbones. *IEEE/ACM Transactions on Networking*, 2009.

[50] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[51] Ka-Cheong Leung and Victor OK Li. Generalized load sharing for packet-switching networks. i. theory and packet-based algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 2006.

[52] Ruediger Martin, Michael Menth, and Michael Hemmkeppler. Accuracy and dynamics of hash-based load balancing algorithms for multipath internet routing. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*. IEEE, 2006.

[53] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[54] Nick McKeown. Software-defined networking. *INFOCOM keynote talk*, 17(2):30–32, 2009.

[55] John McQuillan, Ira Richer, and Eric Rosen. The new routing algorithm for the arpanet. *IEEE transactions on communications*, 28(5):711–719, 1980.

[56] Robert M Metcalfe and David R Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, 1976.

[57] Nithin Michael and Ao Tang. Halo: Hop-by-hop adaptive link-state optimal routing. *IEEE/ACM Transactions on Networking (TON)*, 2015.

[58] Erik Nygren, Ramesh K Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3):2–19, 2010.

[59] Abhay K Parekh and Robert G Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM transactions on networking*, 1993.

[60] Jonathan Perry et al. Fastpass: A centralized zero-queue datacenter network. In *SIGCOMM*, 2015.

[61] Sumet Prabhavat, Hiroki Nishiyama, et al. On load distribution over multipath networks. *IEEE Communications Surveys & Tutorials*, 2012.

[62] Yakov Rekhter, Tony Li, and Susan Hares. A border gateway protocol 4 (bgp-4). Technical report, 2005.

[63] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 418–431. ACM, 2017.

[64] Justine Sherry, Peter Xiang Gao, Soumya Basu, Aurojit Panda, Arvind Krishnamurthy, Christian Maciocco, Maziar Manesh, João Martins, Sylvia Ratnasamy, Luigi Rizzo, et al. Rollback-recovery for middleboxes. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 227–240. ACM, 2015.

[65] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[66] Weiguang Shi, Mike H MacGregor, and Pawel Gburzynski. Load balancing for parallel forwarding. *IEEE/ACM Transactions on Networking (TON)*, 2005.

[67] Ashwin Sridharan et al. Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks. *IEEE/ACM Transactions on Networking*, 2005.

[68] Shekhar Srivastava et al. Determining link weight system under various objectives for OSPF networks using a lagrangian relaxation-based approach. *IEEE Transactions on Network and Service Management*, 2005.

[69] Nguyen B Truong, Gyu Myoung Lee, and Yacine Ghamri-Doudane. Software defined networking-based vehicular adhoc network with fog computing. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1202–1207. IEEE, 2015.

[70] Daphne Tuncer, Marinos Charalambides, Stuart Clayman, and George Pavlou. Adaptive resource management and control in software defined networks. *IEEE Transactions on Network and Service Management*, 12(1):18–33, 2015.

[71] Daphne Tuncer, Marinos Charalambides, Stuart Clayman, and George Pavlou. Flexible traffic splitting in openflow networks. *IEEE Transactions on Network and Service Management*, 2016.

[72] Curtis Villamizar. Ospf optimized multipath (ospf-omp). 1999.

[73] Dahai Xu et al. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. *IEEE/ACM Transactions on Networking*, 2011.

[74] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 432–445. ACM, 2017.