Iowa Research Online
The University of Iowa's Institutional Repository

University of Iowa
Iowa Research Online

Theses and Dissertations

Spring 2017

Multipath approaches to avoiding TCP Incast

Lin Song
University of Iowa

Follow this and additional works at: https://ir.uiowa.edu/etd

 Part of the Electrical and Computer Engineering Commons

Copyright © 2017 Lin Song

This dissertation is available at Iowa Research Online: https://ir.uiowa.edu/etd/6859

Recommended Citation
Song, Lin. "Multipath approaches to avoiding TCP Incast." PhD (Doctor of Philosophy) thesis, University of Iowa, 2017.
https://doi.org/10.17077/etd.5vtx-z1li

Follow this and additional works at: https://ir.uiowa.edu/etd

 Part of the Electrical and Computer Engineering Commons

www.manaraa.com

MULTIPATH APPROACHES TO AVOIDING TCP INCAST

by

Lin Song

A thesis submitted in partial fulfillment of the
requirements for the Doctor of Philosophy
degree in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

May 2017

Thesis Supervisor: Associate Professor Mark S. Andersland

www.manaraa.com

Copyright by

LIN SONG

2017

All Rights Reserved

www.manaraa.com

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Lin Song

has been approved by the Examining Committee
for the thesis requirement for the Doctor of Philosophy
degree in Electrical and Computer Engineering at the
May 2017 graduation.

Thesis Committee: _____
Mark S. Andersland, Thesis Supervisor

_____
Er-Wei Bai

_____
Mona K. Garvin

_____
Jon G. Kuhl

_____
Hantao Zhang

www.manaraa.com

To my wife Min and my parents

www.manaraa.com

ACKNOWLEDGMENTS

This journey of my Ph.D. at The University of Iowa has been a long and grueling one. I would not have been able to complete it without the help of so many people to whom I am indebted.

First of all, I would like to thank my advisor, Dr. Mark S. Andersland, for all his guidance and support throughout my Ph.D. study. His expertise, vision and patient guidance helped shape my thinking and will continue to benefit my future career. He helped me get through a lot of hard times and I learned so many things from him both academically and personally. I could not have come this far without his excellent mentorship.

To my committee members, Dr. Er-Wei Bai, Dr. Mona K. Garvin, Dr. Jon G. Kuhl and Dr. Hantao Zhang, thank you all for your time, effort, valuable feedback and insightful comments on my dissertation.

I would like to give special thanks to Dr. Er-Wei Bai and my former advisor Dr. Ray P.S. Han, for their support, encouragement and guidance since I began my studies at The University of Iowa. Sincere acknowledgement goes to ECE department secretaries Catherine Kern and Dina Blanc for their kind help during my long Ph.D. journey.

My life in Iowa City during my Ph.D. journey was made much better thanks to all my friends. I will not name all of them because I am in fear of accidentally leaving someone out. But thank you all for being friends with me and having a wonderful time together.

Most of all thanks are to my family: my wife Min Li and my parents Dr. Guojun Song and Junxia Wang, for their enduring love, immense patience and support, both emotionally and financially, without which it is impossible for me to go this far. Words cannot express my gratitude and love for them.

www.manaraa.com

ABSTRACT

TCP was conceived to ensure reliable node-to-node communication in moderate-bandwidth, moderate-latency, WANs. As it is now a mature Internet standard, it is the default connection-oriented protocol in networks built from commodity components, including Internet data centers. Data centers, however, rely on high-bandwidth, low-latency networks for communication. Moreover, their communication patterns, especially those generated by distributed applications such as MapReduce, often take the form of synchronous multi-node to node bursts. Under the right conditions, the network switch buffer overflow losses induced by these bursts confuse TCP's feedback mechanisms to the point that TCP throughput collapses. This collapse, termed TCP Incast, results in gross underutilization of link capacities, significantly degrading application performance.

Conventional approaches to mitigating Incast have focused on single-path solutions, for instance, adjusting TCP's receive windows and timers, modifying the protocol itself, or adopting explicit congestion notifications. This thesis explores complementary multi-path approaches to avoiding Incast's onset. The principal idea is to use the regularity and high connectivity of typical data center networks, such as the increasingly popular fat-tree topology, to better distribute multi-node to node bursts across the available paths, thereby avoiding the switch buffer overflows that induce TCP Incast.

The thesis's main contributions are: (1) development of new oblivious, multi-path, routing schemes for fat-tree networks, (2) derivation of relations between the schemes and Incast's onset, and (3) investigation of a novel "front-back" approach to minimizing the packet reordering introduced by multipath routing. Formal analyses are focused on relating schemes' worst-case loading of certain network resources – expressed as oblivious

www.manaraa.com

performance ratios (OPRs) – to Incast's onset. Potential benefits are assessed through ns-3 simulations on fat-trees under a variety of communication patterns. Results indicate that over a variety of experimental conditions, the proposed schemes reduce the incidence of TCP Incast compared to standard routing schemes.

www.manaraa.com

PUBLIC ABSTRACT

The transmission control protocol (TCP) is the principle standard governing the reliable exchange of messages in the Internet. Originally conceived for application in wide area computer networks (WANs) encompassing large geographic areas operating at moderate speeds, it faces new challenges in highly localized, high-speed, data center networks supporting newly ubiquitous Internet services such as Internet search and social networking. The challenge is that messaging in these networks often occurs in large, synchronous many-to-one bursts in which many computers simultaneously respond to a single computer's query with parts of the query's answer. Under the right conditions, these simultaneous responses overload network switches, resulting in a steep decline in the TCP data rate. This decline, termed TCP Incast, significantly degrades application performance, impacting customer experience.

Conventional approaches to mitigating Incast have focused on single-path solutions, in which all messages in a stream follow a single network path to the destination. This thesis explores complementary multi-path approaches to avoiding Incast's onset. The principal idea is to use the regularity and high connectivity of typical data center networks, such as the increasingly popular fat-tree architecture, to better distribute messages across the available paths, thereby avoiding the network switch overloads that induce TCP Incast. Potential benefits are assessed through simulations. Results indicate that over a variety of experimental conditions, the proposed schemes reduce the incidence of TCP Incast compared to standard message passing schemes.

www.manaraa.com

TABLE OF CONTENTS

www.manaraa.com

www.manaraa.com

LIST OF TABLES

Table

www.manaraa.com

LIST OF FIGURES

Figure

www.manaraa.com

www.manaraa.com

CHAPTER 1

INTRODUCTION

The TCP "Incast" problem is significant for data centers. It causes data transfers to miss their deadlines, degrades customer experience and eventually impacts revenue. Conventional approaches to overcoming this problem are focused on single-path solutions, e.g., adjusting TCP's receive windows and timers, modifying the protocol itself, or adopting explicit congestion notifications. Today's newer data center topologies allow for multiple paths between nodes. In this dissertation, we develop new oblivious, multi-path, routing schemes for fat-tree networks to alleviate the Incast problem. In addition, we derive relations between the schemes and Incast's onset, and investigate a novel "front-back" approach to minimizing the packet reordering introduced by multipath routing.

1.1 Background and Motivation

1.1.1 The Data Center Incast Problem

Nowadays, with fast growth of the Internet and its increased popularity, large companies such as Google and Microsoft rely on Internet data centers to provide important services like search and social collaboration to their customers. Many applications deployed in these data centers require significant network bandwidth. For example, implementations of the MapReduce [1] programming model, such as Apache Hadoop [2], rely on the shuffling of large amounts of data among data center nodes. On the other hand, these applications often have strict latency requirements. During a web search request, for instance, the client's queries are simultaneously sent to multiple backend servers, whose

www.manaraa.com

responses are then aggregated and processed. Typical deadlines range from 10 to 100ms, and those not received in time are discarded.

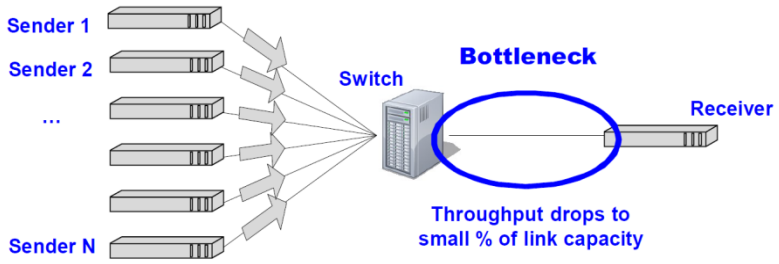


Figure 1.1: A simplified data center Incast model from [3].

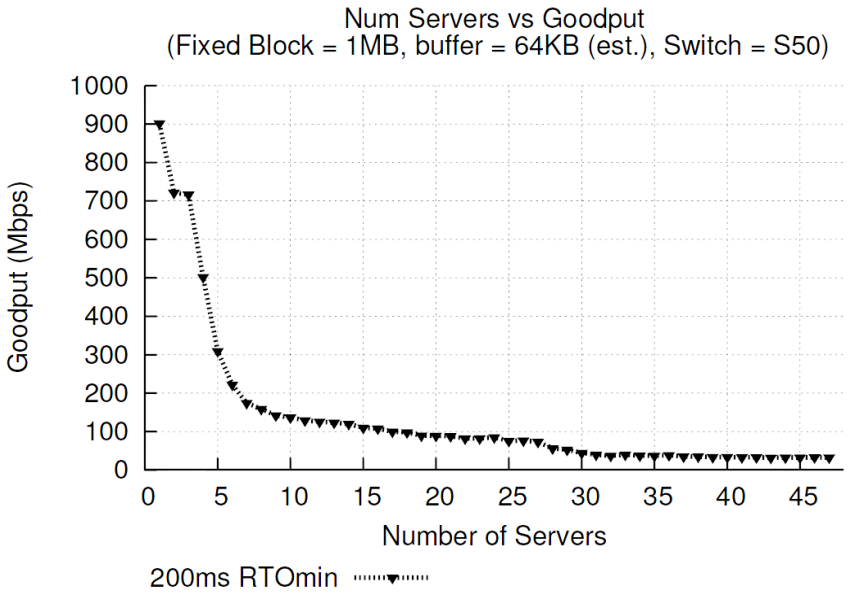


Figure 1.2: TCP Incast collapse on a 48-node cluster from [4].

Most data centers use TCP for inter-node communication. When, as in Figure 1.1, simultaneous data transfers from multiple processing nodes to a single receiving node overfill network switch buffers, the resulting intense packet loss may lead to TCP timeouts

www.manaraa.com

long enough (100s of milliseconds or more) to collapse throughput. This drastic reduction of throughput, termed Incast, causes data transfers to miss their deadlines, affecting the quality of results, and degrading customers' experiences.

From a user's perspective, Incast is caused by the typical "barrier-synchronized request workload" in data center applications such as web search and clustered storage. In this type of workload, each data block is striped over multiple servers. A client node sends simultaneous requests to these servers, which then respond with a fragment of the data block. Traffic in the network is highly synchronized due to multiple servers responding to the client at the same time. This intense, synchronized traffic overflows network switch buffers, resulting in severe packet loss and long TCP timeouts. Because the client must wait for all fragments in the current data block to arrive before sending a new batch of requests, all transfers are halted during the timeout, leading to a steep decline in perceived application-level throughput (goodput).

Shown in Figure 1.2 is a TCP Incast collapse observed on a 48-node cluster. We see that as the number of concurrent servers sending data to the receiver increases, there is a drastic drop in overall application throughput, eventually degrading to less than 5% of the initial value.

Incast was first observed by Andersen et al. in their INCAST project [5, 6]. It was subsequently studied by Chen et al. [3, 7]. In today's data centers, Incast communication patterns can be observed in many popular applications, such as cluster-based storage systems [8, 9], Big Data [10], data analytics [11-13] and Apache Hadoop [2].

To address the Incast problem, researchers have proposed different approaches, including application layer approaches [14-16], modifying the TCP protocol [4, 17-19],

www.manaraa.com

adjusting the TCP congestion or receiver window [20, 21], use of congestion notification [22-25], centralized flow scheduling [26, 27], and use of multiple network paths. We will discuss multipath approaches in more detail in Section 1.2.

1.1.2 Newer Data Center Topologies with Multiple Paths

Although data centers can be built using specialized hardware with custom communication protocols such as Infiniband [28] and Myrinet [29], the high costs of such solutions limit their adoption. Many data centers choose to instead use off-the-shelf commodity products such as Ethernet based routers and switches. Figure 1.3 shows a traditional data center topology where multiple hosts are connected to an access switch1 and multiple access switches are then connected to an aggregation switch, etc. In such topologies, there exist very few to no alternate paths between any two hosts.

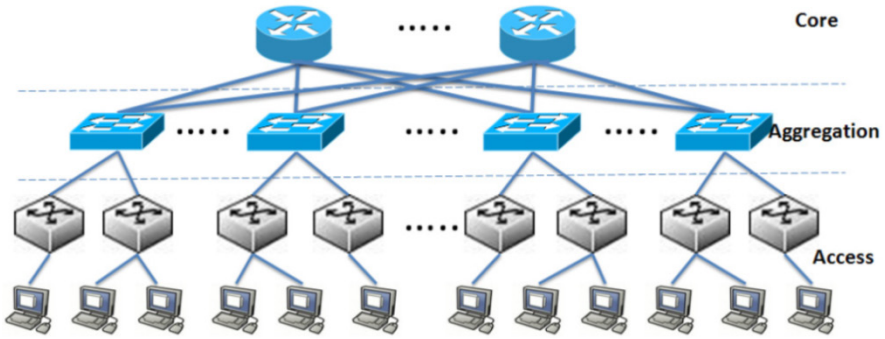


Figure 1.3: A traditional data center topology from [30].

1 We use the term *switch* throughout the rest of this dissertation to refer to devices capable of both layer 2 switching and layer 3 routing.

www.manaraa.com

The evolution of data center topologies allows for multiple paths between each source-destination node pair. Recently, researchers have proposed many new topologies, e.g., the fat-tree [31], VL2 [32], DCell [33], BCube [34], Monsoon [35] and CamCube [36]. The fat-tree is popular in today's data centers. It was proposed in 1985 by Leiserson in [37] and is a special instance of the "Clos topology" [38] invented by Charles Clos for telephone networks in 1953.

1.1.3 The Fat-Tree Architecture

The fat-tree architecture, originally proposed in 1985 by Leiserson in [37], is an increasingly popular choice in today's newer data centers due to its full bisection bandwidth, and has been widely deployed. Typical dimensions of fat-trees used in data centers involve two- or three-tiers of switches plus one-tier of processing nodes.

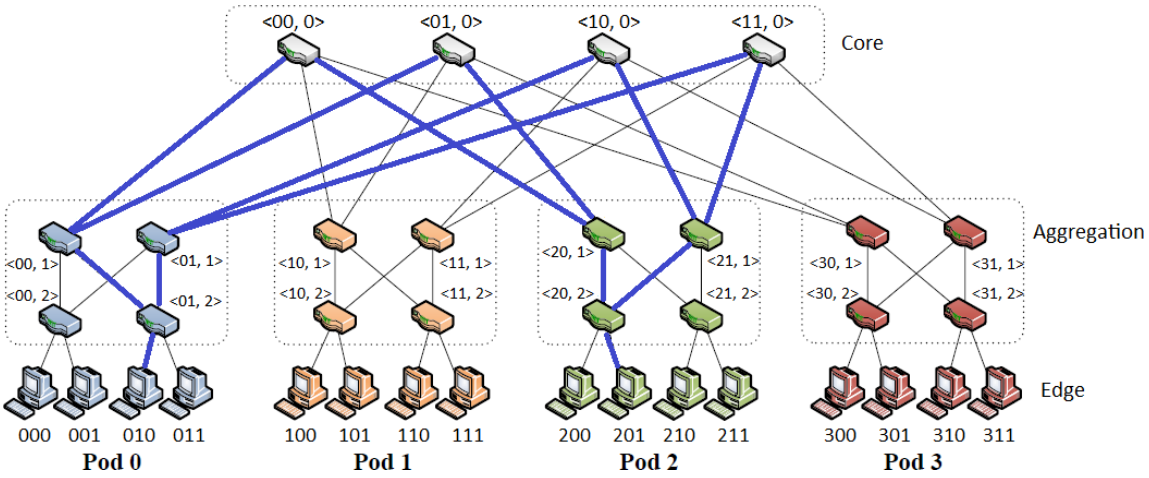


Figure 1.4: A *FT(4, 3)* 4-pod fat-tree topology from [31], with the multiple paths between 010 and 201 highlighted.

www.manaraa.com

Definition 1.1 [39]. A **fat-tree** is an m-port n-tree $FT(m,n)$, built from n tiers of m-port switches (m a power of 2), with the following characteristics:

1. The height of $FT(m,n)$ is $n+1$.
2. $FT(m,n)$ consists of $2 \times \left(\frac{m}{2}\right)^n$ nodes and $(2n-1) \times \left(\frac{m}{2}\right)^{n-1}$ m-port switches.

We label the nodes in $FT(m,n)$ as $P(p = p_0 p_1 \dots p_{n-1})$, where $p \in \{0,1,\dots,m-1\} \times \left\{0,1,\dots,\frac{m}{2}-1\right\}^{n-1}$. The switches are labeled as $SW < w = w_0 w_1 \dots w_{n-2}, l >$, where $l \in \{0,1,\dots,n-1\}$ is the level of the switch and

$$w \in \begin{cases} \left\{0,1,\dots,\frac{m}{2}-1\right\}^{n-1}, & \text{if } l = 0 \\ \{0,1,\dots,m-1\} \times \left\{0,1,\dots,\frac{m}{2}-1\right\}^{n-2}, & \text{if } l \in \{1,2,\dots,n-1\} \end{cases}. \quad (1.1)$$

An m-port 3-tree $FT(m,3)$ has 4 sub-fat-trees ("pods"), with two layers of switches in each: lower-level "edge" switches and upper-level "aggregation" switches, as shown in Figure 1.4. The four paths between 010 and 201 are highlighted. Each edge switch connects $\frac{m}{2}$ processing nodes. At the top level, there are $\left(\frac{m}{2}\right)^2$ "core" switches interconnecting the m pods. In $FT(m,3)$, there are $2 \times \left(\frac{m}{2}\right)^3$ hosts and $5 \times \left(\frac{m}{2}\right)^2$ switches in total.

One major advantage of fat-trees is the high degree of path diversity. For an m-port 3-tree, there are $\left(\frac{m}{2}\right)^2$ equal-cost paths between each source-destination pair, each corresponding to a core switch. Routing between two processing nodes in different sub-fat-trees consists of two phases: from the source node to a core switch ("upward"), and from the core switch to the destination node ("downward"). During the first phase, each switch can select from its $\frac{m}{2}$ different next hops. However, once a core switch is selected, the second phase is deterministic.

www.manaraa.com

1.1.4 Oblivious Routing vs. Adaptive Routing

Oblivious routing refers to routing schemes that are designed without knowledge of the actual traffic demands of a network. For every source-destination node pair, a fixed route, or a randomized rule for choosing routes, is selected in advance, irrespective of how much traffic any pair sends or is expected to send. In contrast, adaptive routing schemes permit the route taken by packets to be affected by the routes taken by other packets in the network. Consequently, the routes selected for source-destination node pairs may change dynamically depending on real-time network traffic conditions.

Because adaptive routing is dependent on real-time traffic conditions, it requires accurate estimates of network traffic flows, which can be difficult to obtain, especially in dynamic environments. Considering these difficulties, recent routing research has focused on identifying good oblivious schemes because they are significantly easier to implement. By designing a good oblivious routing scheme, we can ensure the network is robust to traffic pattern changes. In this dissertation, all our proposed schemes are oblivious schemes.

1.2 Related Work

Can we use the multiple paths available in newer data centers to better balance traffic and thereby avoid Incast? A number of researchers have proposed multipath approaches to load balancing. These include Valiant load balancing (VLB) [32, 35, 40], equal-cost multi-path routing (ECMP) [31], adaptive load balancing (ALB) [41], centralized flow scheduling [42], static VLANs [43-46] and multipath TCP (MPTCP) [47-49] and variants AMTCP [50], MMPTCP [51] and MPTCP-L [52]. Recent surveys of multipath transmission include [53, 54]. Far fewer researchers have considered using these

www.manaraa.com

approaches to combat Incast. These include EW-MPTCP [55], CONGA [56], XMP [57] and FUSO [58]. In the remainder of this subsection we briefly describe the key schemes.

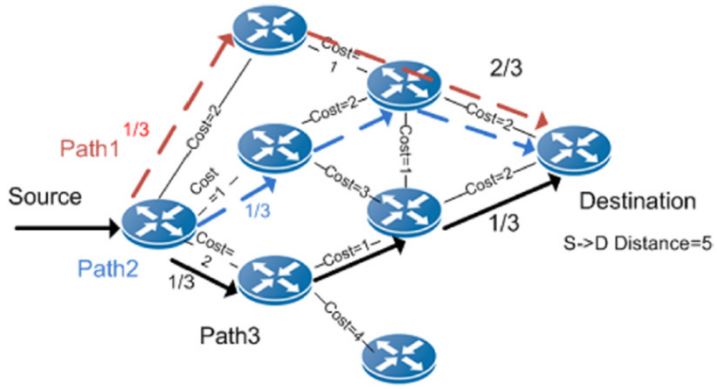


Figure 1.5: ECMP load balancing from [59].

Equal-cost multi-path routing (ECMP) [31] configures ECMP-enabled switches with multiple packet forwarding paths. An arriving packet for which there exists multiple candidate paths is forwarded based on a hash of selected fields in its header ("5-tuple": source and destination IPs, protocol type, source and destination ports). Thus, the traffic load is split across multiple paths. Figure 1.5 shows an example. Traffic from source to destination is routed via three equal-cost paths (all with cost 5), marked by red, blue and black arrows, respectively.

In ECMP, network traffic is split based on flow-granularity, to avoid the problem of packet reordering which degrades TCP throughput. However, all flows may have different timing and/or durations, hence load imbalance can occur in the network which creates hot-spots and leads to poor resource utilization. For example, a hash collision is

www.manaraa.com

possible [42] between large, long-lived flows (i.e. they are routed on the same link), creating a bottleneck and resulting in poor load balancing.

Valiant load balancing (VLB) [32, 35, 40] is a method closely related to ECMP. It tries to achieve load-balancing by first routing each packet from a source switch to a set of randomly selected intermediate switches, which then forward it to the destination. To avoid packet reordering, VLB is usually performed at the flow-level [32] rather than packet-level. The former is largely equivalent to ECMP.

Centralized flow scheduling has also been proposed by researchers [42]. The scheduler assigns large flows to less congested paths, and may reassign existing flows to increase overall throughput in the network. However, it has been shown that the scheduler does not scale, due to its inherent overhead [48]. To more efficiently use a centralized scheduler, other researchers proposed to treat differently flows with short duration, and flows which are long-running and carry significant amounts of data [60-63]. They termed these “mice” and “elephants”, respectively. Different techniques have been proposed for detecting the “elephants”, either from the end hosts or from the switches in the network.

Multipath TCP (MPTCP) [47-49] is an extension for TCP, that allows the use of multiple paths for resilience and load balancing. Published in 2013 as an experimental IETF RFC [49], MPTCP splits each connection into multiple regular TCP subflows, on which traffic is multiplexed based on perceived congestion. A complex formula [64] is used to couple the congestion window increase between subflows. Under certain cases, MPTCP outperforms regular TCP, achieving a higher throughput. However, by routing traffic over multiple paths, MPTCP could induce a high degree of packet reordering, which results in additional delay before in-order data can be delivered to the applications. A survey of TCP packet reordering issues under multipath can be found in [65].

www.manaraa.com

EW-MPTCP [55] is an extension to MPTCP that mitigates Incast collapse through a new congestion control algorithm. Besides the native coupled congestion control [64] performed by MPTCP, each subflow in EW-MPTCP is allowed to perform additional congestion control by dynamically adjusting the congestion window based on the number of responding servers. The goal is to improve fairness at the shared bottleneck links, where multiple TCP subflows in MPTCP could compete with a single path TCP flow.

Congestion Aware Balancing (CONGA) [56] is a network-based distributed load balancing scheme for data centers. Each TCP flow is first divided into flowlets, then allocated to different paths based on real-time congestion information aggregated from multiple switches. However, CONGA relies on global knowledge of congestion in the network, which is difficult to achieve without significant overhead.

Explicit Multipath (XMP) [57] congestion control aims to balance throughput with latency in data center networks. It uses MPTCP and has two components: the Buffer Occupancy Suppression (BOS) algorithm and the Traffic Shifting (TraSh) algorithm. The former employs Explicit Congestion Notification (ECN) [22] to control latency for small flows, while the latter shifts traffic among subflows to improve throughput of large flows.

Fast Multi-path Loss Recovery (FUSO) [58] utilizes multiple paths in data center networks for transport loss recovery, with a focus on reducing the TCP flow completion time (FCT). In the event of potential packet loss on one subflow within the multi-path transport, recovery packets are immediately sent over another subflow with lower packet loss and has space in its congestion window. This is very different from our proposed schemes, because in FUSO, the multiple paths are only used to transmit recovery packets.

www.manaraa.com

The current techniques have either proved too simple to be effective at alleviating Incast, or, in the case of the MPTCP coupled congestion control, too complex to be implemented and could induce packet reordering. What should we do?

1.3 Key Idea

We use network regularity to overcome Incast through multipath routing. Multipath routing [66] is a routing technique that can be used to alleviate network congestion through load balancing. Traffic load is distributed across the network by utilizing the existing multiple routes for routing traffic from a source to a destination. According to the "resource pooling principle" [67], this can effectively shift traffic away from congested links and increase overall network utilization.

Our approaches differ from existing ones in that:

1. They provide a low overhead means to reduce the likelihood of Incast, due to the simple routing setup that reduces packet processing time.
2. They avoid packet reordering and minimize context switches by design.
3. They are scalable and well suited for large networks, because of their compact storage of routing information.

So far, we have investigated two specific approaches:

1. Multipath Routing via Dynamic NIx-Vectors.
2. Virtual Table-Based Multipath Routing (specifically, Dual IPv4/IPv6 Routing).

A key problem with all multipath approaches is packet reordering. We minimize reordering via a novel multipath scheduling algorithm we term the *Front-Back Algorithm*.

www.manaraa.com

1.4 Estimation of Best-Case Improvement

We can estimate the potential gain in perceived application-level throughput (goodput) attainable through Incast avoidance using throughput models developed in [3] and [7]. To begin assume, as in [7], a fixed fragment workload, i.e., all nodes' responses to queries are of fixed size, and assume that TCP's delayed ACK feature is enabled. Let G_S^{IS} and G_S^{IF} denote the throughput of S senders in Incast-susceptible (IS) and Incast-free (IF) networks, respectively, and define:

D – total amount of data to be sent (100 blocks of 256 KB each)

L – total transfer time of the workload without any TCP re-transmission timeout (RTO) events

R – number of RTO events during the transfer

r – value of the minimum RTO (i.e. RTO_{min}) timer value, set to 200ms

I – inter-packet wait time

From [7], the S-sender throughput in an Incast susceptible network can be approximately modeled as

$$G_S^{IS} = \frac{S \times D}{L + (R \times r)} . \tag{1.2}$$

Moreover, from [7], L is related to I and D through

$$L = \frac{D}{\frac{Bandwidth}{S}} + \left(\frac{D}{averageMSS}\right) \times I , \tag{1.3}$$

where *Bandwidth* is assumed to be 1Gbps (= 125,000,000 bytes/s) in the authors' testbed, and *averageMSS* the average maximum segment size, is set to 1500 bytes.

Fitting a piece-wise quadratic polynomial to [7]'s empirical data yields S to R relation

www.manaraa.com

$$R = \begin{cases} 0.679\,S^2 - 4.04\,S + 5.5\ , & S \le 8 \\ -0.0859\,S^2 + 4.24\,S - 11.1\ , & S > 8 \end{cases}. \qquad (1.4)$$

Similarly, fitting a piece-wise quadratic polynomial to [7]'s empirical data one finds that

$$I = \begin{cases} 0.0988\,S^2 - 0.485\,S + 0.786\ , & S \le 8 \\ -0.00433\,S^2 + 0.237\,S + 1.63\ , & S > 8 \end{cases}. \qquad (1.5)$$

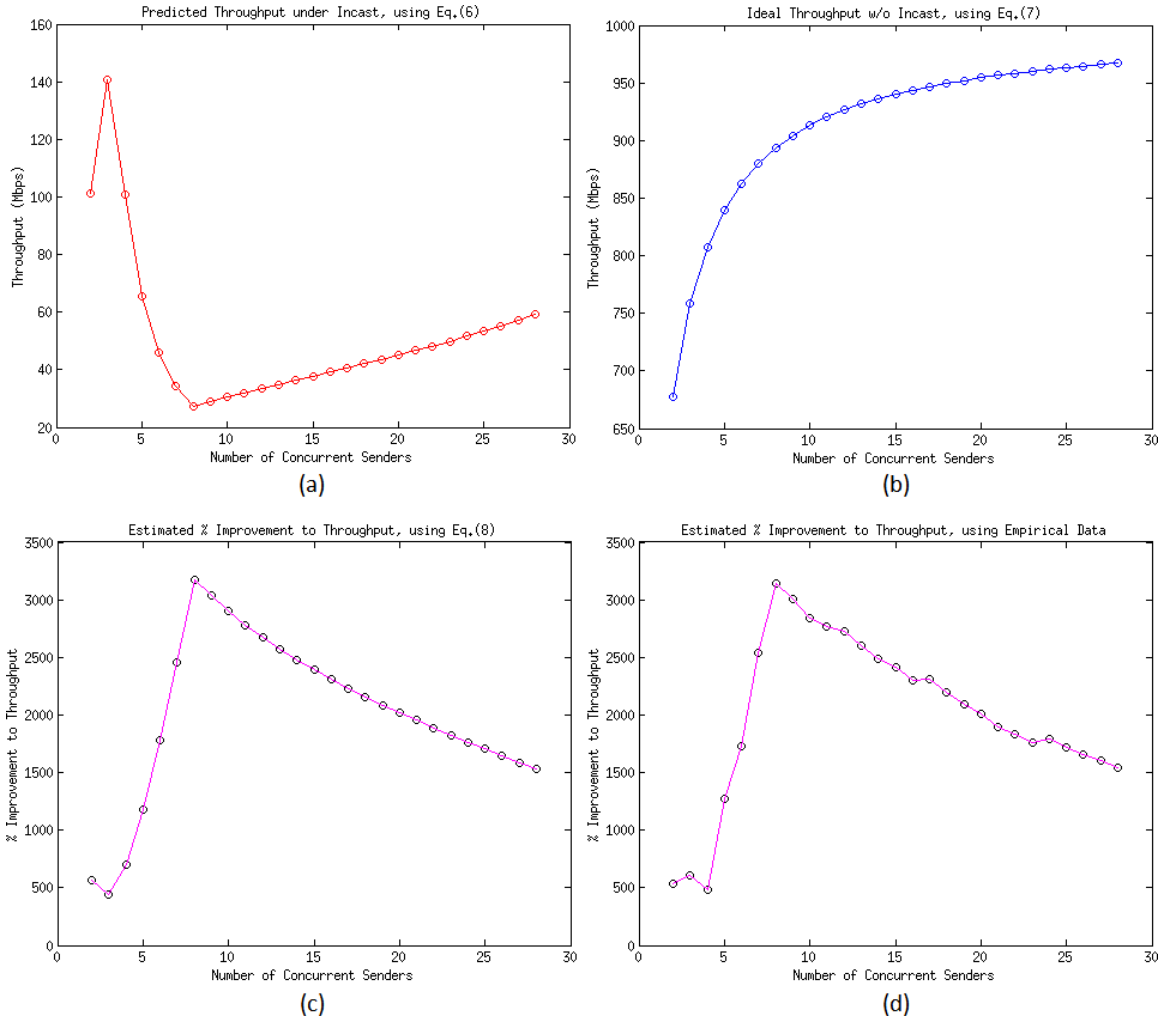


Figure 1.6: Estimated best-case improvement to throughput. (a) Predicted throughput w/ Incast, using Eq. 1.6. (b) Ideal throughput w/o Incast, using Eq. 1.7. (c) Estimated % improvement to throughput, using Eq. 1.8. (d) Same as (c), except using empirical data.

www.manaraa.com

Substituting *I* into Eq. 1.3, and *L* and *R* into Eq. 1.2, we find

$$G_S^{IS} = \begin{cases} \dfrac{26214400\,S}{1.86\,S^2 - 9.07\,S + 14.84}, & S \le 8 \\ \dfrac{26214400\,S}{-0.0929\,S^2 + 5.20\,S + 26.3}, & S > 8 \end{cases}. \quad (1.6)$$

To calculate the ideal throughput when Incast is avoided, we use Eq. 1 from [3].

$$G_S^{IF} = \frac{blockSize}{RTT + \dfrac{blockSize \times S}{Bandwidth}} \times S, \quad (1.7)$$

where *blockSize* is 256KB, and *RTT* is set to 2ms which is the observed value in the authors' testbed in [7]. We can then compute

$$\frac{G_S^{IF} - G_S^{IS}}{G_S^{IS}} = \begin{cases} \dfrac{7812500 \times (1.86\,S^2 - 9.28\,S + 14.6)}{16384\,S + 15625}, S \le 8 \\ -\dfrac{78125 \times (9.29\,S^2 - 499\,S - 2607)}{16384\,S + 15625}, S > 8 \end{cases}. \quad (1.8)$$

The estimated throughput in the Incast-susceptible and Incast-free cases, and the possible percent improvement seen in the absence of Incast, are plotted in Figure 1.6. Figure 1.6 (a) shows predicted throughput across *S* senders under Incast, using Eq. 1.6. The ideal throughput without Incast is shown in Figure 1.6 (b) by applying Eq. 1.7. Figure 1.6 (c) illustrates the estimated percent of improvement to throughput, using Eq. 1.8. Figure 1.6 (d) was generated without using the piece-wise fitted equations for calculating *I* and *R*, but instead use the empirically observed data from Figure 16 in [7].

From these plots, we observe that in the best case, there can be more than 3000% maximum improvement to the throughput across *S* senders. While a 3000% improvement may seem high, given the drastic impact Incast has on throughput, as shown in Figure 1.2,

www.manaraa.com

it is not unreasonable. The decrease in improvement starting from $S = 9$ (note that we still have more than 1500% improvement at $S = 25$) can be explained as follows: Because every flow must traverse the "bottleneck" link at the receiver (as shown in Figure 1.1), the ideal aggregated throughput is capped by the bandwidth on that link (1Gbps). On the other hand, the throughput under Incast improves slightly with increasing number of concurrent senders. This is the reason behind the decrease we see in the figure.

1.5 Summary of Our Contributions

In this dissertation, we use network regularity to overcome Incast through multipath routing. In Chapter 2, we develop two novel oblivious, multi-path, routing schemes for fat-tree networks. These include *Multipath Routing via Dynamic NIx-Vectors* and *Virtual Table-Based Multipath Routing*, with *Dual IPv4/IPv6 Routing* as a specific example. Our scalable approaches provide a low overhead means to reduce the likelihood of Incast. We conclude with a discussion of irregular traffic extensions for the proposed schemes.

In Chapter 3, we establish a relationship between the proposed multipath routing schemes and the avoidance of Incast's onset. First, we note, using Kulkarni's model of synchronized, many-to-one TCP flows, that reducing flow loss probabilities delays the onset of Incast. Next, we observe, from well-known results on switch buffer sizing, that flow loss probabilities can be reduced by reducing switch loads. Finally, we show by formal analysis that the worst-case loads – the loads most responsible for TCP Incast – are reduced by our proposed routing schemes.

A key problem, faced by all approaches to routing packets over multiple paths, is packet reordering. In Chapter 4, we investigate a novel "front-back" approach to minimizing the packet reordering introduced by multipath routing. We establish the

www.manaraa.com

optimality of an algorithm implementing our front-back approach for 2 paths, and then briefly discuss how this *Front-Back Algorithm* could be integrated into existing protocols. Its performance is then contrasted to other proposed algorithms for combating multipath reordering, with examples highlighting its advantages. We conclude with a discussion of the *Front-Back Algorithm*'s *N*-path extension ($N > 2$) and examples illustrating the extension's potential advantages.

Our performance analysis in Chapter 5 begins with an investigation of the proposed routing schemes. We focus on their worst-case loading of certain network resources – expressed as oblivious performance ratios (OPRs). We then explore typical Incast traffic patterns in data center networks, and describe a novel method of traffic matrix decomposition to help visually illustrate and classify traffic patterns. Finally, we assess the potential benefits of our schemes through ns-3 simulations on fat-trees under a variety of traffic conditions. Results indicate that over a variety of experimental conditions, the proposed schemes reduce the incidence of TCP Incast compared to standard routing schemes.

www.manaraa.com

CHAPTER 2

MULTIPATH ROUTING FOR DATA CENTER NETWORKS

In this chapter, we investigate the root cause and key points related to TCP Incast, and develop two novel oblivious, multi-path, routing schemes for fat-tree networks. We conclude with a discussion of irregular traffic extensions for the proposed schemes.

2.1 What Causes Incast? – A Brief Investigation

The problem of data center Incast was first observed by Andersen et al. in their INCAST project [5, 6]. It arises in high bandwidth, low latency, networks when simultaneous data transfers from multiple senders to a single receiver overflow a bottleneck switch buffer. The ensuing intense synchronous packet loss may induce excessively large TCP re-transmission timeouts (on the order of hundreds of milliseconds) that reduce perceived application-level throughput (goodput) to a fraction of available bandwidth. Thus, data transfers miss their deadlines, affecting quality of the result, degrading customer experience and eventually impacting revenue.

The root causes for data center Incast include synchronized many-to-one request patterns (in applications such as MapReduce [1]) and the high bandwidth, low latency Ethernet links causing a mismatch of TCP's retransmission timeout and increased penalty. This is further aggravated by the highly synchronous nature of data transfer in data center applications, leading to a drastic drop in network or application throughput.

Three key features of data centers make them particularly susceptible to Incast. First, data center operators typically rely on commodity switches to reduce costs. These switches usually have limited buffer space. Second, data transfer patterns in data center

www.manaraa.com

networks are usually “bursty”, in the form of barrier-synchronized request workloads. For example, in the “MapReduce” application, simultaneous requests from a master node are periodically sent to multiple slave nodes. Upon receiving these requests, the slave nodes respond with data they have or perform calculations. Third, to maintain data integrity, the request/response process is usually synchronous with tight deadlines, i.e. the next batch of requests cannot start until all slave nodes have replied to the current batch of requests. Therefore, if there is switch buffer overflow during a batch, the system will wait for the uncompleted responses and delaying processing of the next batch and thereby decreasing performance.

2.2 Multipath Routing via Dynamic NIx-Vectors

To take advantage of the multiple paths, better balance traffic and reduce the likelihood of data center Incast, we have developed two novel routing schemes, *Multipath Routing via Dynamic NIx-Vectors* and *Virtual Table-Based Multipath Routing*. To the best of our knowledge, no similar schemes have been proposed in the literature.

2.2.1 What is a NIx-Vector

The original NIx-Vector concept was introduced by Riley et al. in 2001 [68]. NIx-Vector (neighbor-index vector) Routing is a form of source routing. It stores, very compactly in the packet header, a complete source-to-destination routing path. Because of its efficient route storage, NIx-Vector Routing is well-suited for large network topologies.

As packets are being generated at a source node for transmission, a routing cache is first searched for a previously built NIx-Vector for the destination. If none exists, the NIx-Vector is built by performing a breadth-first search (BFS), and then storing a neighbor-

www.manaraa.com

index for each hop along the path indicating which outgoing interface should be used to route the packet. At the time of routing, the appropriate neighbor-index is extracted from the NIx-Vector by switches at each hop, which then transmit the packet through the indicated interface. This process repeats until the destination is reached.

Table 2.1: NIx-Vector creation example from [68].

Hop	C_i	B_i	*NIx*	*NVU*	*NIx-Vector* (binary)
0	2	1	1	1	1
1	3	2	2	3	1 10
2	6	3	5	6	1 10 101
3	13	4	4	10	1 10 101 0100
4	27	5	20	15	1 10 101 0100 10100
5	5	3	3	18	1 10 101 0100 10100 011
6	14	4	7	22	1 10 101 0100 10100 011 0111
7	5	3	1	25	1 10 101 0100 10100 011 0111 101
8	2	1	0	26	1 10 101 0100 10100 011 0111 101 0
9	1	1	0	27	1 10 101 0100 10100 011 0111 101 0 0

Table 2.1 [68] shows an example for NIx-Vector creation. In the table, C_i is the number of neighbors at each hop; B_i equals $\lceil \log_2 C_i \rceil$, the number of bits needed for storing routing decision; *NIx* specifies the neighbor-index chosen for packet routing; *NVU* is the used counter; and *NIx-Vector* lists the NIx-Vector generated with that hop's information appended.

2.2.2 Details of Proposed Algorithm

We modify NIx-Vector routing for use in data center networks. In the original NIx-Vector concept, if multiple shortest paths were found during the breadth-first search (BFS),

www.manaraa.com

only the first path is used to compute the NIx-Vector. To take advantage of the multiple paths available in newer data center topologies such as the fat-tree, we modify the said BFS algorithm, to return multiple (or multiple sets of) shortest paths on demand. In other words, our modified algorithm is flexible. For example, we can achieve randomized load balancing (RLB) if the shortest path used to compute the NIx-Vector is randomly selected from all shortest paths discovered by BFS. A comparison of this algorithm with the original BFS is shown in Figure 2.1.

```
unmark all vertices
  mark starting vertex x
  list L = x
  tree T = x
  while L nonempty
     choose some vertex v from front of list L
     visit v
     for each unmarked neighbor w
        mark w
        add it to end of list L
        add edge vw to T
```

```
unmark all vertices
  mark starting vertex x
  list L = x
  tree T = x
  while L nonempty
     choose some vertex v from front of list L
     visit v
     list P = all unmarked neighbors of v
     randomize element order in list P
     while P nonempty
        choose some vertex w from front of list P
        mark w
        add it to end of list L
        add edge vw to T
```

Figure 2.1: (a) Pseudocode of original BFS from [69],
(b) Our modified BFS algorithm to achieve RLB.

Compared to normal table-based routing, where only one path is used, the new scheme allows better load balancing within the data center and hence, helps to alleviate the Incast problem.

To enable multipath capabilities while avoiding packet reordering, we combine *Dynamic NIx-Vector Routing* with a novel *Front-Back Algorithm*, to be discussed in Chapter 4.

www.manaraa.com

2.3 Virtual Table-Based Multipath Routing

2.3.1 Algorithm Description

We also investigate an approach we term *Virtual Table-Based Multipath Routing*, which takes advantage of multiple network layer stacks installed on a switch to balance traffic.

This approach has some resemblance to "ensemble routing" [43, 44], which exploits multiple virtual local area networks (VLANs) to route traffic. However, our approach has several significant differences:

1. It uses static routing tables which do not have the overhead of VLAN packet tags.
2. It avoids the costs of VLAN creation and management.
3. Unlike VLAN which operates in Layer 2, the data link layer of the network stack, our scheme operates in Layer 3, the network layer.

2.3.2 Example: Dual IPv4/IPv6 Routing

Dual IPv4/IPv6 Routing can serve as an example of *Virtual Table-Based Routing*. With the rapid adoption of IPv6 in recent years, many switches come with support for both IPv4 and IPv6, and companies are rapidly deploying IPv6 in their infrastructures. By setting up different routing tables for IPv4 and IPv6, we can obtain two different paths for each source-destination pair.

To enable multipath capabilities while avoiding packet reordering, we combine *Dual IPv4/IPv6 Routing* with the *Front-Back Algorithm*, to be discussed in Chapter 4.

Our setup of dual IPv4/IPv6 static routing is inspired by the *source modulo k (s-mod-k)* [70, 71] and *destination modulo k (d-mod-k)* [39, 72-74] routing schemes for system

www.manaraa.com

area networks. To illustrate how these schemes work, we use the network in Figure 1.4 as an example.

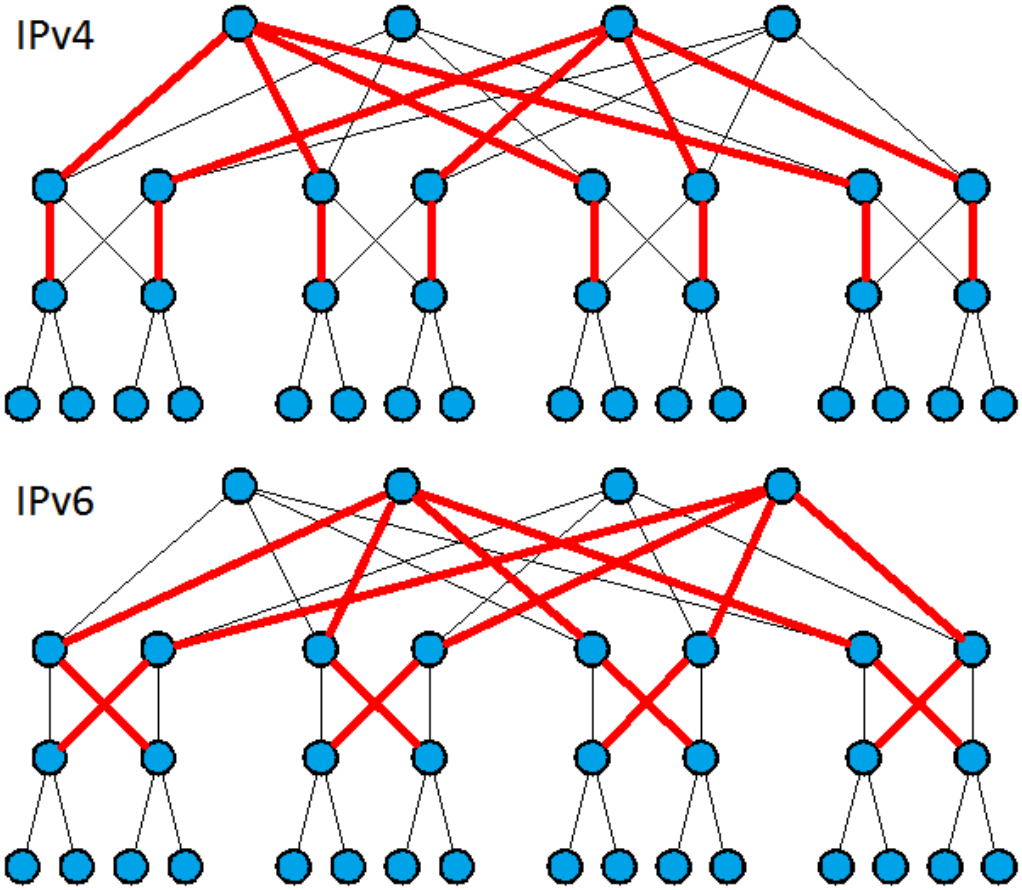


Figure 2.2: Our setup of static routing for IPv4 (top) and IPv6 (bottom).

In *d-mod-k,* during the first stage of routing (edge switch to aggregation switch), packets with consecutive destinations are to be shuffled between the two up-links. To do this, the least significant bit (LSB) of destination node label is checked, and the packet is forwarded to the switch with the same LSB in its label. During the second stage of routing

www.manaraa.com

(aggregation switch to core switch), the second least significant bit (SLSB) of destination node label is checked, with a similar forwarding process as in the first stage.

The *s-mod-k* scheme only differs from *d-mod-k* in one respect: it checks source nodes' labels instead of destination nodes' labels. Researchers have shown that *s-mod-k* and *d-mod-k* have similar performance [75].

We set up dual IPv4 and IPv6 static routing as shown in Figure 2.2. Thickened lines denote the default upward forwarding route for each switch for IPv4 and IPv6, respectively. Compared to the *s-mod-k* and *d-mod-k* schemes, our approach has the following advantages:

1. Due to the use of static routing tables, the switches are freed from the tasks of constantly checking source/destination addresses and matching LSB or SLSB bits for every packet. Instead, they can now be installed with a single default upward forwarding route. This saves time and greatly lowers processing overhead.
2. Our approach avoids traffic imbalance in the cases that the load from each node is different.

To illustrate point 2 above, assume that the loads from nodes 000, 001, 010 and 011, in Figure 1.4, are 8, 4, 2 and 10, respectively. Now look only at the leftmost sub-fat-tree. Under our scheme, the loads on the upward ports of switches <00, 2> and <01, 2> are 6, 6, 6 and 6 (left to right). Likewise, the loads on switches <00, 1> and <01, 1> are 6, 6, 6 and 6. Under the *s-mod-k* routing scheme, the loads on the upward ports of switches are less balanced, namely, 8, 4, 2 and 10, on switches <00, 2> and <01, 2> and 8, 2, 4 and 10 on switches <00, 1> and <01, 1>.

www.manaraa.com

2.4 Irregular Traffic Extensions

Applications deployed in real data centers may have very different communication patterns (flow timing and length), leading to irregular traffic within the network. If we make the additional assumption of irregular network traffic, how should our proposed schemes change?

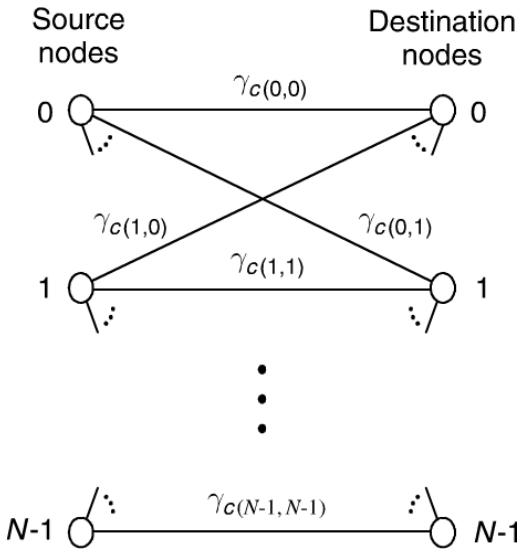


Figure 2.3: Bipartite graph for finding the maximum-load traffic pattern from [76].

In this scenario, we expect to change the algorithm for path selection in our proposed schemes. The techniques that we will discuss in Chapter 3 for analyzing the incurred switch load can be utilized to pre-calculate the load on each switch in the network under different traffic patterns and routing policies. With this information, we can adjust path selection to optimize for the worst-case pattern within the given set of traffic patterns. Specifically, for the proposed scheme *Multipath Routing via Dynamic NIx-Vectors*, instead

www.manaraa.com

of randomly picking a path from the pool of available shortest paths, we now set the probability that a switch is to be selected based on this information.

Going in the reverse direction, what if we have already selected a routing scheme, and want to know its performance under different traffic patterns? The authors of [76] derived a procedure for computing the worst-case traffic pattern, i.e., the pattern that generates the maximum load over all links. To find the pattern that gives the highest load on a link c, the first step is to generate a bipartite graph as shown in Figure 2.3. In the graph, vertices for every source node are placed on the left, and those for the destination nodes are placed on the right. Each edge is labeled $\gamma_{c(s,d)}$ denoting the load on switch c by unit traffic from s to d. By finding a maximum-weight matching of the graph, we obtain the permutation that induces maximum load on c. This procedure is repeated for all links in the network to find the worst-case traffic pattern.

2.5 Conclusions

In this chapter, we first highlighted the root cause and key points related to TCP Incast. These include the synchronized many-to-one request patterns in data centers, the commodity network switches with limited buffer space, and the high bandwidth, low latency Ethernet links causing a mismatch of the TCP timeout.

To take advantage of the existing multiple paths in newer data center topologies such as the fat-tree, we developed two novel oblivious, multi-path routing schemes *Multipath Routing via Dynamic NIx-Vectors* and *Virtual Table-Based Multipath Routing*, with *Dual IPv4/IPv6 Routing* as a specific example. Our scalable approaches provide a low overhead means to reduce the likelihood of Incast. We concluded this chapter with a discussion of irregular traffic extensions for the proposed schemes.

www.manaraa.com

CHAPTER 3

BRIDGING THE PROPOSED SCHEMES TO INCAST

In this chapter, we establish a relationship between the proposed multipath routing schemes and the avoidance of Incast's onset. First, we note, using Kulkarni's model of synchronized, many-to-one TCP flows, that reducing flow loss probabilities delays the onset of Incast. Next, we observe, from well-known results on switch buffer sizing, that flow loss probabilities can be reduced by reducing switch loads. Finally, we show by formal analysis that the worst-case loads – the loads most responsible for TCP Incast – are reduced by our proposed routing schemes.

3.1 TCP Incast and Flow Loss Rate

TCP Incast is typically defined as the drastic drop in perceived application-level throughput (goodput), observed when bursts of synchronized many-to-one TCP flows overfill network switch buffers, resulting in packet losses that confuse TCP's congestion control mechanisms to the point that timeouts become excessive. Data center transfers are particularly susceptible to Incast because their networks exhibit high bandwidth with low latency, and most single-node requests spawn synchronized multi-node replies from large numbers of nodes.

One of the earliest explicit models of the relationship between packet loss and perceived application layer throughput in the presence of large numbers of competing synchronized flows was proposed by Kulkarni in his 2012 dissertation [77]. Kulkarni identifies two principle causes of Incast. The first, which he terms Anterior Block Transfer Timeout (ABTT), arises when, due to TCP's short-term unfairness, some synchronized,

www.manaraa.com

many-to-one block transfers finish earlier than others. As the transfers are synchronized, those that finish early must wait for the others to finish. TCP, seeing the bandwidth freed by the early finishers' waits, increases the transmission windows of those flows that remain active to sizes that could not be sustained had the early finishers' flows transmissions not been suspended. Consequently, when the next round of block transfers begins, the buffers in some bottleneck switch are overwhelmed by these outsized window transmissions, resulting in substantial packet losses. Should any flow lose all of the packets in its transmission window, a timeout ensues that, again due to block synchronization, halts all flows' transmissions until the timeout expires and the packets are successfully retransmitted, thereby collapsing throughput.

The second key cause, termed Intermediate Block Transfer Timeout (IBTT), arises when a sender responding to a many-to-one query does not receive enough duplicate ACKs, after a packet loss, for TCP to trigger an immediate resend of the missing packets. Instead, the sending flow's TCP waits one timeout period before resending the packets and then resets the sender's transmission window to one. Once again, all flows' transmissions are halted until the timeout expires and the last packets in the block are successfully retransmitted, collapsing throughput. Kulkarni shows that IBTT is the principle cause of Incast when the number of involved flows is small while ABTT predominates otherwise. More importantly for our purposes, he demonstrates, via an analytic extension of Padhye et al.'s well-known single-flow TCP model [78] to the case of multiple synchronized flows that, as one might intuitively expect, the onset of Incast due to both mechanisms is highly correlated to flow packet loss, and consequently, that strategies that reduce packet loss, such as increasing switch buffer size, delay the onset of Incast. Figure 3.1, from [77]

www.manaraa.com

illustrates the fidelity of the model compared to ns-2 simulation under the conditions superimposed on the model.

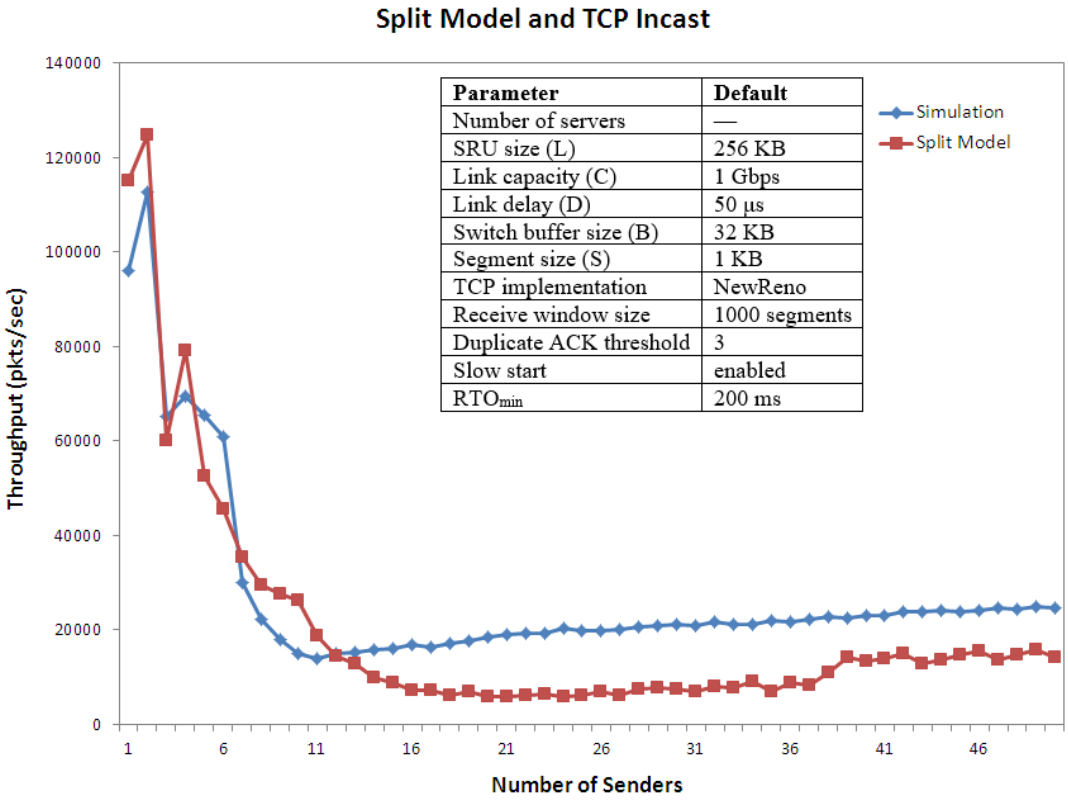


Figure 3.1: Comparison of the split model to Incast simulation results, with the simulation conditions overlaid. [77]

Next, we use established results from switch buffer sizing theory to show that we can reduce the loss rate of TCP flows in fat-tree networks by reducing the switch loads.

3.2 Switch Load and Flow Loss Rate

The topic of switch buffer sizing has been an area of active research since the early 1990s. It is of interest to us because it can be used to size buffers and avoid Incast given

www.manaraa.com

worst case bounds on switch buffers. In a classic 1994 paper [79], Villamizar and Song demonstrated that, although WAN TCP throughput can be collapsed by sufficiently large reductions in queuing capacity, such collapses can be avoided when the random early detection queueing discipline (RED) is employed, and queuing capacities are maintained at or above links' bandwidth-delay products. This results in the widely-used guideline that each link should have a buffer size of $B = \overline{RTT} \times C$, where $\overline{RTT}$ is the average round-trip time of flows through that link, and C is the link's data rate.

In [80], Morris analyzed the effect of passing large numbers of TCP flows through switches. Packet loss rates approaching 50% were observed, causing noticeable delays for users. He proposed that instead of allocating only one round-trip time of buffering, switches should be provisioned to have buffer space proportional to the number of active flows. In a subsequent paper [81], he proposed a model aimed at explaining as well as predicting loss rates for TCP traffic, and introduced a new algorithm for switch buffering, he termed "flow-proportional queuing (FPQ)." Under FPQ, TCP is controlled by changing the queue length in the switch in proportion to the number of active TCP connections. It is claimed that FPQ can accommodate heavy TCP traffic without causing high loss rates.

In 2004, Appenzeller et al. [82], proposed the "Stanford model" for switch buffer sizing. Their main claim was that the widely-used guideline that each link should have a buffer of $B = \overline{RTT} \times C$ no longer holds for backbone switches. They showed that when large numbers of multiplexed TCP connections are in their congestion avoidance phases, they tend to become un-synchronized and hence require smaller buffers. As the standard deviation of aggregate load decreases with $\sqrt{n}$, their new guideline was $B = (\overline{RTT} \times C)/\sqrt{n}$, where n is the number of flows (short-lived and long-lived) sharing the target link.

www.manaraa.com

The authors claimed that use of the new rule allowed buffer size requirements to be reduced by up to 99% with no noticeable change in throughput.

A critical assumption of the model is that TCP flows have varying round-trip times, which lead to de-synchronization as they traverse common links. Without this assumption, the flows can become synchronized and behave like a single "big" TCP flow, which then again requires a buffer size of $B = \overline{RTT} \times C$. Because the RTT is largely fixed in data center networks, the above paper's results are not applicable to our work.

More recently, a series of papers by Dovrolis et al. made further contributions to buffer sizing. In [83], they raised concerns about the "Stanford model", pointing out that while setting $B = (\overline{RTT} \times C)/\sqrt{n}$ does not lead to significant link utilization loss when a link carries many TCP flows, it can cause excessively high loss rates (up to 5%~15%) for these flows. This results in TCP re-transmissions and throughput drops, as well as significant variability in throughput and latency.

The paper distinguished between "long" and "short" TCP flows. Short flows spend all their time in the slow-start phase, while long flows follow the linear "saw tooth" pattern in the phase of congestion avoidance. An important observation made in [83] is that one cannot classify flows simply by looking at their sizes. When the network is uncongested, even a big-size flow can keep staying in slow-start phase. On the other hand, in congested networks, short flows can go into the congestion avoidance phase early and spend most of their time there. The authors also claimed that those long flows which are bottlenecked at links other than the target link should be excluded from the n flows used in calculating bandwidth in the Stanford model. Flows that are primarily limited by the TCP receiver's advertised window should also be excluded.

www.manaraa.com

An additional important result in [83] is the authors' expression of the loss rate p experienced by TCP flows at a bottleneck link as a function,

$$p = \frac{(0.87\ N)^2}{\left(C\ \times\ T_p + B\right)^2}\ , \tag{3.1}$$

of the buffer size B, where N denotes the number of flows, T_p, the round-trip propagation delay, and C, the link's capacity. In other words, the loss rate p *increases with the square* of the number of flows N at the bottleneck link. Furthermore, for fixed N, the loss rate p *decreases slowly* with increasing buffer size B, following a power law. Thus, one can reduce the loss rate p of TCP flows by reducing the traffic load on network switches, or the number of flows *N* traversing each switch in the fat-tree network.

3.3 Reduced Switch Load under Proposed Schemes

In this section, we first outline important properties of fat-trees, and define key fat-tree routing terminology. We then show, via formal analysis that the worst-case switch loads – the loads most responsible for TCP Incast – are reduced by our proposed routing schemes.

3.3.1 Fat-Tree Properties

m-port n-trees $FT(m,n)$, fat-trees constructed from n-tiers of m-port switches, have the following properties:

Property 1 [84]**.** $FT(m,n)$ contains m, m-port n-1-trees (sub-fat-trees denoted by $SUBFT(m,n-1)$), $m \times \frac{m}{2}$, m-port n-2-trees (sub-fat-trees denoted by $SUBFT(m,n-2)$), …, and $m \times \left(\frac{m}{2}\right)^{n-2}$, m-port 1-trees (sub-fat-trees denoted by $SUBFT(m,1)$).

www.manaraa.com

Property 2 [84]**.** Let $SUBFT(m,t)$ be the smallest sub-fat-tree in $FT(m,n)$ that contains processing nodes a and b. Then there exist $\left(\frac{m}{2}\right)^{t-1}$ different shortest paths from a to b. If such a sub-tree does not exist, there are $\left(\frac{m}{2}\right)^{n-1}$ different shortest paths from a to b. In this case, a and b are in different top level sub-fat-trees $SUBFT(m,n-1)$.

Property 3 [84]**.** In $FT(m,n)$, when there exist $\left(\frac{m}{2}\right)^{x}$ different shortest paths from processing node s to processing node d, each of the level $n-1-t$, $0 \leq t \leq x$, up/down links that carry traffic from s to d is used by $\left(\frac{m}{2}\right)^{x-t}$ shortest paths.

Property 4. In $FT(m,n)$, when there exist $\left(\frac{m}{2}\right)^{x}$ different shortest paths from processing node s to processing node d, each of the level $n-1-t$, $0 \leq t \leq x$, switches that carry traffic from s to d is used by $\left(\frac{m}{2}\right)^{x-t}$ shortest paths.

Property 5 [84]**.** In $FT(m,n)$, a level t, $0 \leq t \leq n-1$, up link carries traffic from at most $\left(\frac{m}{2}\right)^{n-1-t}$ source nodes. A level t down link carries traffic to at most $\left(\frac{m}{2}\right)^{n-1-t}$ destination nodes.

Property 6. In $FT(m,n)$, a $level$ 0 ("core") switch carries traffic from at most $2 \times \left(\frac{m}{2}\right)^{n}$ source nodes. A $level\ t$, $0 < t \leq n-1$, switch carries traffic from/to at most $\left(\frac{m}{2}\right)^{n-t}$ source/destination nodes in that $SUBFT(m,t)$.

Proof: By construction, $FT(m,n)$ has $2 \times \left(\frac{m}{2}\right)^{n}$ processing nodes. In the worst case, traffic from all nodes passes through a single $level$ 0 switch. On the other hand, each $level\ t$ $(0 < t \leq n-1)$ switch has $\frac{m}{2}$ "up" and "down" links, respectively. Thus, from fat-tree Property 5, each switch carries traffic from/to at most $\left(\frac{m}{2}\right)^{n-1-t} \times \frac{m}{2} = \left(\frac{m}{2}\right)^{n-t}$ source/destination nodes in that $SUBFT(m,t)$. □

www.manaraa.com

3.3.2 Fat-Tree Routing: Definitions

To meaningfully compare routing schemes, it is essential to have metrics. Given that Incast is a worst-case phenomenon, it natural to look for worst case metrics. In this section, we extend [84]'s approach to assessing routing schemes' effects on worst case link loads to that of assessing routing schemes' effects on worst case switch loads. Since in fat-trees all links have the same capacity, comparing maximum link loads is equivalent to comparing maximum link utilizations.

We begin by reviewing traffic and routing characterizations, and link load bounds, from [84]. Throughout we assume that all fat-tree links are full-duplex, with the same capacity in both the up (away from the processing nodes) and down (toward the processing nodes) channel directions.

Definition 3.1 [84]**.** Matrix TM with entries $tm_{i,j} \geq 0, 0 \leq i \leq N-1$ and $0 \leq j \leq N-1$, is said to be a **traffic matrix** for fat-tree $FT(m,n)$, when $N = 2 \times \left(\frac{m}{2}\right)^{n}$ and tm_{ij} specifies the amount of traffic sent from node i to node j.

To lower bound the worst-case link load induced by a traffic matrix TM, note first that the total traffic sent from any node i is $\sum_j tm_{i,j}$, and the total traffic received at i is $\sum_j tm_{j,i}$. Because i connects to the fat-tree via a single local full-duplex link, it follows that the maximum load on i's local link is $max\{\sum_j tm_{i,j}, \sum_j tm_{j,i}\}$. Consequently, the worst-case link load induced by TM over all links' up and down channels under any routing policy is lower bounded by maximum over all nodes' link's local up and down channels. This bound motivates the following definition.

www.manaraa.com

Definition 3.2 [84]**.** The **base load** imposed on $FT(m,n)$ by traffic matrix TM is

$$baseload(TM) = \max_{0 \le i \le N-1} \left\{ max \left\{ \sum_j tm_{i,j}, \sum_j tm_{j,i} \right\} \right\}. \quad (3.2)$$

The actual worst-case link load induced in a fat-tree by traffic matrix TM depends on the fat-tree's routing scheme.

Definition 3.3 [84]**.** A **routing** r for fat-tree $FT(m,n)$ is defined by

(1) A set of paths $P_{i,j} = \left\{ P_{i,j}^1, P_{i,j}^2, \dots, P_{i,j}^{|P_{i,j}|} \right\}$, between each source-destination (SD) pair (i,j) in $FT(m,n)$,

(2) A path weighting $f_{i,j} = \left\{ f_{i,j}^k \middle| k = 1,2, \dots, |P_{i,j}| \right\}$, $\sum_k f_{i,j}^k = 1$, specifying the fraction of traffic to be routed via each path.

Note that this definition encompasses both single and multipath routing schemes. When $|P_{i,j}| > 1$ with no $f_{i,j}^k = 1$, r is said to be a multipath routing. Otherwise it is single path routing.

Given traffic matrix TM and routing r, we assume the contribution of path $P_{i,j}^k$ traffic to link l's up or down channel's load under r is $tm_{i,j} \times f_{i,j}^k$ when the channel belongs to path $P_{i,j}^k$, and 0 otherwise. This is the standard linear network assumption for oblivious routing [76]. Hence, the load imposed on channel l_D, $D \in \{up, down\}$, of link l of $FT(m,n)$ by routing r is

$$load(l_D, r, TM) = \sum_{i,j,k \; such \; that \; l_D \in P_{i,j}^k} tm_{i,j} \times f_{i,j}^k ; \quad (3.3)$$

and the load imposed by routing r on $FT(m,n)$'s most congested link channel, termed the **maximum link load** in [84], is

www.manaraa.com

$$MLOAD(r,TM) = \max_{l \in Links} \{\max\{load(l_{up}, r, TM), load(l_{down}, r, TM)\}\}. \quad (3.4)$$

Similarly, given traffic matrix TM and routing r, the contribution of path $P_{i,j}^k$ traffic to switch s's load under r is $tm_{i,j} \times f_{i,j}^k$ when s belongs to path $P_{i,j}^k$, and 0 otherwise. Hence, the load imposed on switch s of $FT(m,n)$ by routing r is

$$loadsw(s,r,TM) = \sum_{i,j,k \text{ such that } s \in P_{i,j}^k} tm_{i,j} \times f_{i,j}^k; \quad (3.5)$$

and the load imposed by routing r on $FT(m,n)$'s most congested switch is

$$MLOADSW(r,TM) = \max_{s \in Switches} \{loadsw(s,r,TM)\}. \quad (3.6)$$

$MLOAD(r,TM)$ is the worst-case link load incurred by $FT(m,n)$ under routing r given traffic matrix TM. The min-max link-optimal routing r for this traffic matrix, minimizes $MLOAD(r,TM)$.

Definition 3.4 [84]**.** The **min-max optimal** (minimum worst case) **link load** imposed on $FT(m,n)$ by traffic matrix TM is

$$OPTU(TM) = \min_{r \text{ is a routing}} \{MLOAD(r,TM)\}. \quad (3.7)$$

Similarly, $MLOADSW(r,TM)$ is the worst-case switch load incurred by $FT(m,n)$ under routing r given traffic matrix TM. The min-max switch-optimal routing r for this traffic matrix minimizes $MLOADSW(r,TM)$.

Definition 3.5. The **min-max optimal** (minimum worst case) **switch load** imposed on $FT(m,n)$ by traffic matrix TM is

$$OPTUSW(TM) = \min_{r \text{ is a routing}} \{MLOADSW(r,TM)\}. \quad (3.8)$$

www.manaraa.com

Intuitively, because a switch's load must be at least as large as the load of any attached link, the min-max optimal switch load bounds the min-max optimal link load.

Lemma 3.1. *Given* $FT(m,n)$*, with* m *a power of 2 and* $n \geq 1$*, for all* TM*,*

$$baseload(TM) \leq OPTU(TM) \leq OPTUSW(TM) \leq baseload(TM) \times 2 \times \left(\frac{m}{2}\right)^n. \quad (3.9)$$

Proof: Note first that as $FT(m,n)$ contains $2 \times \left(\frac{m}{2}\right)^n$ processing nodes, no switch can have a load of more than $2 \times \left(\frac{m}{2}\right)^n \times baseload(TM)$. So, $OPTUSW(TM) \leq baseload(TM) \times 2 \times \left(\frac{m}{2}\right)^n$. Next, fix TM and let r denote the routing that achieves $OPTUSW(TM)$. Then, by $OPTUSW(TM)$'s, $OPTU(TM)$'s, and $baseload(TM)$'s definitions,

$$\begin{aligned} OPTUSW(TM) &= MLOADSW(r,TM) \\ &\geq MLOAD(r,TM) \\ &\geq OPTU(TM) \\ &\geq baseload(TM), \end{aligned} \quad (3.10)$$

where the first inequality follows from the fact that $MLOADSW(r,TM) \geq loadsw(s,r,TM) \geq MLOAD(r,TM)$ for the switch s attached to the link achieving $MLOAD(r,TM)$, with equality when, for instance, TM specifies that traffic be sent from a single node to a single node. □

In the next section, we derive worst-case switch load bounds for specific routing schemes, and in the process, significantly tighten Lemma 3.1's upper bound.

www.manaraa.com

3.3.3 Formal Analysis

In this section, we derive bounds on the worst case switch load for our proposed schemes, and compare these bounds to those obtained for standard schemes.

The routing schemes (and variants) to be considered include:

1. *Normal Routing*, which corresponds to the single-path IPv4 table-based scheme,
2. *Dynamic NIx-Vector Routing*, which is a NIx-vector routing variant that uses our modified BFS algorithm to select one path from the multiple available shortest paths, and
3. *Multipath Routing via Dynamic NIx-Vectors*, which extends NIx-vector routing to the multipath case using the *Front-Back Algorithm* (introduced in Chapter 4) to distribute traffic across the shortest paths.

3.3.3.1 Normal Routing

Theorem 3.1. *Given Normal Routing (NR) on* $FT(m,n)$*, with* m *a power of 2 and* $n \geq 1$*, for all* TM*, the load on any* $level\ t$ *switch,* $0 \leq t \leq n-1$*, satisfies*

$$MLOADSW_t(NR,TM) \leq baseload(TM) \times 2 \times \left(\frac{m}{2}\right)^{n-t}. \qquad (3.11)$$

Proof: Under *Normal Routing* all traffic between source nodes in different $n-1$ level sub-fat-trees of fat-tree $FT(m,n)$ passes through $FT(m,n)$'s leftmost $level\ 0$ switch. Hence this switch is maximally loaded by TMs under which all source nodes' traffic is directed towards $n-1$ level sub-fat-trees other than their own. By fat-tree Property 6, there are $2 \times \left(\frac{m}{2}\right)^{n}$ source nodes in $FT(m,n)$, and by Definition 3.2, each can send no more than $baseload(TM)$ units of traffic. Hence inequality (3.11) holds for all TM, when $t = 0$,

www.manaraa.com

with equality when, for instance, TM specifies that each source node send one unit of traffic to every source node outside of its $n-1$ level sub-fat-tree.

Similarly, under *Normal Routing*, all traffic between source nodes in a particular $n-t$ level sub-fat-tree of $FT(m,n)$, $1 \leq t \leq n-1$, and source nodes outside of this sub-fat-tree, passes through this sub-fat-tree's leftmost $level\ 0$ switch (a $level\ t$ switch of $FT(m,n)$). Hence this switch is maximally loaded by TMs under which all of its source nodes' traffic is directed towards, or originates from, source nodes outside of its sub-fat-tree. By fat-tree Property 6, there are $\left(\frac{m}{2}\right)^{n-t}$ source nodes in each $n-t$ level sub-fat-tree, and by Definition 3.2, each can send and receive no more than $2 \times baseload(TM)$ units of traffic. Hence inequality (3.11) holds for all TM, when $0 < t \leq n-1$, with equality when, for instance, TM specifies that each source node send one unit of traffic to every source node outside of its $n-t$ level sub-fat-tree. □

By Definition 3.2, $baseload(TM) \times 2 \times \left(\frac{m}{2}\right)^{n}$ is the largest traffic load that the $2 \times \left(\frac{m}{2}\right)^{n}$ source nodes in $FT(m,n)$ can generate, and $baseload(TM) \times 2 \times \left(\frac{m}{2}\right)^{n-t}$ is the largest traffic load that the $\left(\frac{m}{2}\right)^{n-t}$ source nodes in $SUBFT(m,n-t)$ can collectively send and receive. Accordingly, (3.11) hold for all routings r and all traffic matrices TM.

Corollary 3.1. *For all routings r on $FT(m,n)$, with m a power of 2 and $n \geq 1$, and for all TM, the load on any level t switch, $0 \leq t \leq n-1$, satisfies*

$$MLOADSW_t(r,TM) \leq baseload(TM) \times 2 \times \left(\frac{m}{2}\right)^{n-t}. \square \qquad (3.12)$$

Note that since the bounds in (3.12) are tight for $r = NR$, NR is in fact a "worst case" routing.

www.manaraa.com

3.3.3.2 Dynamic NIx-Vector Routing

In this section, we briefly investigate the min-max link and switch load performance of single path *Dynamic NIx-Vector Routing* to establish benchmarks against which the performance of our proposed *Multipath Dynamic NIx-Vector Routing* scheme can be compared. We limit our analysis to $FT(m,2)$ and $FT(m,3)$ because optimal routing schemes are known for these two cases and because these fat-tree types are adequate to support most applications. An $FT(32,3)$, for instance, can support 8192 processing nodes.

The following theorem, on which our results are based, is an immediate consequence of Theorems 6 and 7 in [84].

Theorem 3.2 [84]**.** *Given Dynamic NIx-Vector Routing (DNVR) on* $FT(m,n)$, with m a power of 2 and $n \in \{2,3\}$,

$$MLOAD(DNVR,TM) \leq \begin{cases} baseload(TM) \times \left\lceil \sqrt{\frac{m}{2}} \right\rceil, & \text{if } n = 2 \\ baseload(TM) \times \frac{m}{2}, & \text{if } n = 3 \end{cases}. \qquad (3.13)$$

Proof: As outlined in Chapter 2, $DNVR$ uses a modified BFS algorithm to dynamically return shortest paths between all source nodes i and j in $FT(m,n)$. It is then free to assign shortest paths to source-destination pairs in any desired demand oblivious fashion.

In Theorem 6 of [84] it is shown that in the case $n = 2$, it is possible to assign shortest paths to source-destination pairs in such a way that no link ever carries traffic to or from more than $\left\lceil \sqrt{\frac{m}{2}} \right\rceil$ source nodes. It is also shown that in the case $n = 3$ it is possible to assign shortest paths to source-destination pairs in such a way that no link ever carries traffic to or from more than $\frac{m}{2}$ source nodes. Under these $DNVR$ assignments it is immediate that the maximum load experienced by fat-tree $FT(m,n)$ satisfies (3.13) for $n = 2$ and $n = 3$. □

www.manaraa.com

Theorem 3.3. *Given Dynamic NIx-Vector Routing (DNVR) on* $FT(m,n)$*, with* m *a power of 2 and* $n \in \{2,3\}$*, for all* TM*, the load on any level* t *switch,* $0 \le t \le n-1$*, satisfies*

$$MLOADSW_t(DNVR,TM) \le \begin{cases} baseload(TM) \times \min\left\{\left\lceil\sqrt{\frac{m}{2}}\right\rceil, \left(\frac{m}{2}\right)^{n-1-t}\right\} \times m, & \text{if } n=2 \\ baseload(TM) \times \min\left\{\frac{m}{2}, \left(\frac{m}{2}\right)^{n-1-t}\right\} \times m, & \text{if } n=3 \end{cases}. \quad (3.14)$$

Proof: Fix TM, let s denote a switch that achieves $MLOADSW(DNVR,TM)$ under $DNVR$, and let $l_0, l_1, \ldots, l_{m-1}$ denote the links attached to its m ports. As all traffic that enters s leaves s, the load on switch s equals half the sum of the load on its links' up and down channels. Hence, by $MLOADSW(DNVR,TM)$'s definition, and (3.13),

$$\begin{aligned} MLOADSW(DNVR,TM) &= loadsw(s,DNVR,TM) \\ &= \frac{1}{2}\sum_{p=0}^{m-1}\left(load\left(l_{p,up},DNVR,TM\right) + load\left(l_{p,down},DNVR,TM\right)\right) \\ &\le \begin{cases} \frac{1}{2}\sum_{p=0}^{m-1}\left(baseload(TM) \times \left\lceil\sqrt{\frac{m}{2}}\right\rceil \times 2\right), & \text{if } n=2 \\ \frac{1}{2}\sum_{p=0}^{m-1}\left(baseload(TM) \times \frac{m}{2} \times 2\right), & \text{if } n=3 \end{cases} \\ &= \begin{cases} baseload(TM) \times \left\lceil\sqrt{\frac{m}{2}}\right\rceil \times m, & \text{if } n=2 \\ baseload(TM) \times \frac{m}{2} \times m, & \text{if } n=3 \end{cases}. \end{aligned} \quad (3.15)$$

On the other hand, by Corollary 3.1, for $0 \le t \le n-1$,

$$MLOADSW_t(r,TM) \le baseload(TM) \times 2 \times \left(\frac{m}{2}\right)^{n-t}. \quad (3.16)$$

Merging (3.15) and (3.16) we obtain (3.14). □

www.manaraa.com

3.3.3.3 Multipath Routing via Dynamic NIx-Vectors (using all paths)

The routing scheme *Multipath Routing via Dynamic NIx-Vectors (using all paths)* ($MDNVR$) works as follows: Let X be the number of shortest paths between a source-destination node pair (i,j), and let these X different shortest paths be $P_{i,j}^1, P_{i,j}^2, \dots, P_{i,j}^X$. $MDNVR$ uses the *Front-Back Algorithm* (Chapter 4), to allocate the same amount of traffic $f_{i,j}^1 = \cdots = f_{i,j}^X = \frac{1}{X}$, to each path.

Theorem 3.4. *Given Multipath Routing via Dynamic NIx-Vectors (using all paths) (MDNVR) on $FT(m,n)$, with m a power of 2 and $n \geq 1$, for all TM,*

$$MLOAD(MDNVR, TM) = baseload(TM); \tag{3.17}$$

and the load on any switch satisfies

$$baseload(TM) \leq MLOADSW(MDNVR, TM) \leq baseload(TM) \times m. \tag{3.18}$$

Proof: Consider first (3.17). Our proof approach is adapted from [84]. Since, by Lemma 3.1, $baseload(TM) \leq MLOAD(MDNVR, TM)$, it suffices to show that $MLOAD(MDNVR, TM) \leq baseload(TM)$. By fat-tree Property 3, for all source nodes i and j in $FT(m,n)$, shortest path traffic from i to j passes through at most $\left(\frac{m}{2}\right)^{n-1-t}$ *level* t link up-channels, $0 \leq t \leq n-1$. By fat-tree Property 5, each of these channels carries shortest path traffic from at most $\left(\frac{m}{2}\right)^{n-1-t}$ source nodes $i_0, i_1, \dots, i_{\left(\frac{m}{2}\right)^{n-1-t}-1}$. Under $MDNVR$ routing, traffic from each of these $\left(\frac{m}{2}\right)^{n-1-t}$ nodes is uniformly distributed across these $\left(\frac{m}{2}\right)^{n-1-t}$ links. Hence, recalling that the total traffic departing from source node i under traffic matrix TM can be expressed as $\sum_{i \neq j} tm_{ij}$, and that, by

www.manaraa.com

Definition 3.2, no link's up-channel load can exceed $baseload(TM)$, we have, for all $level\ t$ link up channels l,

$$\begin{aligned} load(l, MDNVR, TM) &\leq \sum_{p=0}^{\left(\frac{m}{2}\right)^{n-1-t}-1} \frac{\sum_{i\neq j} tm_{ij}}{\left(\frac{m}{2}\right)^{n-1-t}} \\ &\leq \sum_{p=0}^{\left(\frac{m}{2}\right)^{n-1-t}-1} \frac{baseload(TM)}{\left(\frac{m}{2}\right)^{n-1-t}} \\ &= baseload(TM). \end{aligned} \tag{3.19}$$

Since, by $FT(m,n)$'s symmetry, the same arguments hold for all $level\ t$ link down channels, (3.17) follows from the definition $MLOAD(MDNVR, TM)$.

Consider next (3.18). Since, by Lemma 3.1, $baseload(TM) \leq MLOADSW(MDNVR, TM)$, it suffices to establish the upper bound. Fix TM, let s denote a switch that achieves $MLOADSW(MDNVR, TM)$ under $MDNVR$, and let $l_0, l_1, \ldots, l_{m-1}$ denote the links attached to its m ports. As all traffic that enters s leaves s, the load on switch s equals half the sum of the load on its ports' link's up and down channels. Hence, by $MLOADSW(MDNVR, TM)$'s definition, and (3.17),

$$\begin{aligned} &MLOADSW(MDNVR, TM) = loadsw(s, MDNVR, TM) \\ &= \frac{1}{2}\sum_{p=0}^{m-1}\left(load\left(l_{p,up}, MDNVR, TM\right) + load\left(l_{p,down}, MDNVR, TM\right)\right) \\ &\leq \frac{1}{2}\sum_{p=0}^{m-1}(baseload(TM) \times 2) \\ &= baseload(TM) \times m. \square \end{aligned} \tag{3.20}$$

www.manaraa.com

As $MLOAD(MDNVR,TM)$ is both lower bounded (by Lemma 3.1) and upper bounded (by Theorem 3.4) by $baseload(TM)$, the following corollary is clear.

Corollary 3.2. *Given $FT(m,n)$, with m a power of 2 and $n \geq 1$, for all TM,*

$$MLOAD(MDNVR,TM) = OPTU(TM) = baseload(TM). \square \quad (3.21)$$

Corollary 3.2 establishes that $MDNVR$ is the min-max link-optimal routing for $FT(m,n)$. In fact, $MDNVR$ is also a min-max switch-optimal routing. But showing this takes a bit more effort.

To begin we note that both bounds in (3.18) are tight. To see this, consider first a traffic matrix $TM1$ that specifies that a single source node send traffic to another source node within its smallest (1 level) sub-fat-tree. Under $TM1$ the switch attached to these two nodes load equals $baseload(TM)$ while all other switches' loads are zero. Hence $MLOADSW(MDNVR,TM1) = baseload(TM1)$.

Consider next a traffic matrix $TM2$ that specifies that each source node in $n-1$ level sub-fat-tree X send one unit of traffic to the node in the same position in the adjacent $n-1$ level subtree $(X+1) \bmod m$. As each node sends one unit of traffic they all send $baseload(TM2)$. Under $TM2$ with $MDNVR$ routing, the $baseload(TM2) \times 2 \times \left(\frac{m}{2}\right)^n$ traffic generated by these $2 \times \left(\frac{m}{2}\right)^n$ nodes is uniformly distributed across the $\left(\frac{m}{2}\right)^{n-1}$ $level$ 0 switches hence $MLOADSW_0(MDNVR,TM2) = (baseload(TM2) \times 2 \times \left(\frac{m}{2}\right)^n)/\left(\frac{m}{2}\right)^{n-1} = baseload(TM2) \times m$. Similarly, under $TM2$ with $MDNVR$ routing, the $baseload(TM) \times 2 \times \left(\frac{m}{2}\right)^{n-t}$ up and down traffic generated by the $\left(\frac{m}{2}\right)^{n-t}$ nodes in each $n-t$ level sub-fat-tree is uniformly distributed across $\left(\frac{m}{2}\right)^{n-t}$ $level\ t$ switches. Hence

www.manaraa.com

$MLOADSW_t(MDNVR, TM2) = (baseload(TM2) \times 2 \times \left(\frac{m}{2}\right)^{n-t}) / \left(\frac{m}{2}\right)^{n-t} = baseload(TM2) \times m$, as was the case for the $level\,0$ switches. Thus $MLOADSW(MDNVR, TM2) = baseload(TM2) \times m$.

The implication of having these distinct bounds is that the bounding arguments that established $MDNVR$'s min-max link-optimality cannot be used to establish its min-max switch optimality. Instead we demonstrate optimality by showing that for every TM, deviating from $MDNVR$ can only increase a fat-tree's maximum switch load.

Theorem 3.5. *Given $FT(m, n)$, with m a power of 2 and $n \geq 1$, for all TM,*

$$MLOADSW(MDNVR, TM) = OPTUSW(TM). \tag{3.22}$$

Proof: Fix TM and let s be a switch with $loadsw(s, MDNVR, TM) = LOADSW(MDNVR, TM)$. By Definition 3.5, it suffices to show that

$$LOADSW(r, TM) \geq loadsw(s, MDNVR, TM) \text{ for all } r. \tag{3.23}$$

There are two cases. Suppose first that s is a $level\ n-1$ switch. Then since all paths through s link directly to a source node, all paths are unique. So no change in r can reduce s's switch load, and hence (3.23) holds. Suppose next that s is a $level\ t$ switch, $0 \leq t < n-1$, and let $SUBFT(m, n-t)$ denote the smallest sub-fat-tree containing s. Under $MDNVR$ routing all traffic between nodes i and j is evenly distributed among all paths between i and j, and hence by fat-trees symmetry, all $level\ t$ switches, in $SUBFT(m, n-t)$, along those paths. By superposition it follows that the aggregate traffic traversing $level\ t$ in $SUBFT(m, n-t)$ is likewise equally distributed among all $level\ t$ switches in $SUBFT(m, n-t)$. But as this aggregate is fixed by TM, any routing r that decreases switch s's load increases the load of some other $level\ t$ switch in $SUBFT(m, n-t)$. So once again (3.23) holds. □

www.manaraa.com

3.3.3.4 Multipath Routing via Dynamic NIx-Vectors (using $\lceil X/k \rceil$ paths)

Theorem 3.5 establishes the min-max optimality of $MDNVR$. $MDNVR$, however, routes over all shortest paths between nodes. When this is not possible but reduced switch loads are still desired, it is natural to investigate the performance gains attainable via multipath routing over a subset of the available paths.

Let $\lceil\ \rceil$ denote the next largest integer (ceiling) function. The routing scheme *Multipath Routing via Dynamic NIx-Vectors (using* $\lceil X/k \rceil$ *paths)* ($MDNVRk$) works as follows: Let X be the number of shortest paths between a source-destination node pair (i, j). $MDNVRk$ uses the *Front-Back Algorithm* (Chapter 4) to allocate the same fraction of traffic $f_{i,j}^1 = \cdots = f_{i,j}^{\lceil X/k \rceil} = \frac{1}{\lceil X/k \rceil}$ to each of a randomly selected subset $P_{i,j}^1, P_{i,j}^2, \ldots, P_{i,j}^{\lceil X/k \rceil}$, of these X paths. When, for instance, $k = 2$ and $X = 4$, the algorithm uses $\lceil 4/2 \rceil = 2$ paths. We assume $k \leq \left(\frac{m}{2}\right)^{n-1}$, which is the maximum number of shortest paths between any two nodes in $FT(m, n)$.

Our proposed routing scheme from Chapter 2, *Dual IPv4/IPv6 Routing with Front-Back,* utilizes both IPv4 and IPv6 to set up two different paths. It can be considered as a special case of $MDNVRk$ on $FT(4, 3)$, with fixed routing.

Theorem 3.6. *Given Multipath Routing via Dynamic NIx-Vectors (using* $\lceil X/k \rceil$ *paths) (*$MDNVRk$*) on* $FT(m, n)$*, with* m *a power of 2 and* $n \geq 1$*, for all* TM*, the maximum load on level* t *links,* $0 \leq t \leq n - 1$*, satisfies*

$$baseload(TM) \leq MLOAD_t(MDNVRk, TM) \leq baseload(TM) \times \min\left\{k, \left(\frac{m}{2}\right)^{n-1-t}\right\}; \quad (3.24)$$

and the maximum load on level t *switches,* $0 \leq t \leq n - 1$*, satisfies*

$$baseload(TM) \leq MLOADSW_t(MDNVRk, TM) \leq baseload(TM) \times \min\left\{k, \left(\frac{m}{2}\right)^{n-1-t}\right\} \times m. \quad (3.25)$$

www.manaraa.com

Proof: Consider first (3.24). Our proof approach is adapted from [84]. Since, by Lemma 3.1, $baseload(TM) \leq MLOAD_t(MDNVRk, TM)$, it suffices to establish the upper inequality in (3.24). By fat-tree Property 3, for all source nodes i and j in $FT(m,n)$, shortest path traffic from i to j passes through at most $\left(\frac{m}{2}\right)^{n-1-t}$ $level\ t$ link up-channels, $0 \leq t \leq n-1$. By fat-tree Property 5, each of these channels carries shortest path traffic from at most $\left(\frac{m}{2}\right)^{n-1-t}$ source nodes $i_0, i_1, \ldots, i_{\left(\frac{m}{2}\right)^{n-1-t}-1}$. Under $MDNVRk$ routing, traffic from each of these $\left(\frac{m}{2}\right)^{n-1-t}$ nodes is uniformly distributed across $\left\lceil \left(\frac{m}{2}\right)^{n-1-t} / q \right\rceil$ randomly selected links where $q = \min\left\{k, \left(\frac{m}{2}\right)^{n-1-t}\right\}$. In the worst case, all nodes' traffic is distributed over the same $\left\lceil \left(\frac{m}{2}\right)^{n-1-t} / q \right\rceil$ links. Hence, recalling that the total traffic departing from source node i under traffic matrix TM can be expressed as $\sum_{i \neq j} tm_{ij}$, and that, by Definition 3.2, no link's up-channel load can exceed $baseload(TM)$, we have, for all $level\ t$ links' up channels l,

$$
\begin{aligned}
load_t(l, MDNVRk, TM) &\leq \sum_{p=0}^{\left(\frac{m}{2}\right)^{n-1-t}-1} \frac{\sum_{i \neq j} tm_{ij}}{\left\lceil \left(\frac{m}{2}\right)^{n-1-t} / q \right\rceil} \\
&\leq \sum_{p=0}^{\left(\frac{m}{2}\right)^{n-1-t}-1} \frac{baseload(TM)}{\left(\frac{m}{2}\right)^{n-1-t} / q} \\
&= baseload(TM) \times q \\
&= baseload(TM) \times \min\left\{k, \left(\frac{m}{2}\right)^{n-1-t}\right\}.
\end{aligned}
\tag{3.26}
$$

www.manaraa.com

Since, by $FT(m,n)$'s symmetry, the same arguments hold for all $level\ t$ link down channels, (3.24) follows from the definition $MLOAD_t(MDNVRk,TM)$.

Consider next (3.25). Since, by Lemma 3.1, $baseload(TM) \le MLOADSW_t(MDNVRk,TM)$, it again suffices to establish the upper bound. Fix TM, let s denote a switch that achieves $MLOADSW_t(MDNVRk,TM)$ under $MDNVRk$, and let $l_0, l_1, \ldots, l_{m-1}$ denote the links attached to its m ports. As all traffic that enters s leaves s, the load on switch s equals half the sum of the load on its ports' link's up and down channels. Hence, by $MLOADSW_t(MDNVRk,TM)$'s definition, and (3.24),

$$
\begin{aligned}
&MLOADSW_t(MDNVRk,TM) = loadsw_t(s,MDNVRk,TM)\\
&= \frac{1}{2}\sum_{p=0}^{m-1}\left(load_t\left(l_{p,up},MDNVRk,TM\right) + load_t\left(l_{p,down},MDNVRk,TM\right)\right)\\
&\le \frac{1}{2}\sum_{p=0}^{m-1}\left(2 \times baseload(TM) \times \min\left\{k,\left(\frac{m}{2}\right)^{n-1-t}\right\}\right)\\
&= baseload(TM) \times \min\left\{k,\left(\frac{m}{2}\right)^{n-1-t}\right\} \times\ m. \square
\end{aligned}
\tag{3.27}
$$

Like the $MLOADSW(MDNVR,TM)$ bounds given in Theorem 3.4, the $MLOADSW(MDNVRk,TM)$ bounds in Theorem 3.6 are tight.

Paralleling Theorem 3.4's $MLOADSW(MDNVR,TM)$ bound's tightness arguments suffices to show that Theorem 3.6's $MLOADSW(MDNVRk,TM)$ bounds are also tight. By paralleling Theorem 3.5's optimality arguments one can also show that $MDNVRk$'s switch load is min-max optimal among all $\lceil X/k \rceil$-path policies. That is, in the absence of knowledge of TM, unequally weighting the traffic assigned to the available paths is suboptimal.

www.manaraa.com

3.3.3.5 Summary

In this section, we derived worst case switch load bounds for four fat-tree routing schemes. These bounds are summarized in Table 3.1.

Table 3.1: Lower and upper bounds for the maximum switch load under different routing schemes.

$MLOADSW(\cdot, TM)$	Lower Bound	Upper Bound
NR	$baseload(TM)$	$baseload(TM) \times 2 \times \left(\frac{m}{2}\right)^n$
$DNVR$	$baseload(TM)$	$baseload(TM) \times \left\lfloor \sqrt{\frac{m}{2}} \right\rfloor \times m$, if $n = 2$ $baseload(TM) \times \frac{m}{2} \times m$, if $n = 3$
$MDNVRk$	$baseload(TM)$	$baseload(TM) \times k \times m$
$MDNVR$	$baseload(TM)$	$baseload(TM) \times m$

Table 3.2: Comparison of baseload normalized maximum switch loads under different routing schemes.

$\frac{MLOADSW(\cdot, TM)}{baseload(TM)} \leq$ \ Fat-tree size [max # of shortest paths X]	$FT(2,2)$ [1]	$FT(4,2)$ [2]	$FT(4,3)$ [4]	$FT(8,3)$ [16]
NR	2	8	16	128
$DNVR$	2	8	8	32
$MDNVRk$ ($\left\lfloor \frac{1}{2}X \right\rfloor$ paths)	-	8	8	16
$MDNVRk$ ($\left\lfloor \frac{3}{4}X \right\rfloor$ paths)	-	-	5.33	10.67
$MDNVR$ (all paths)	2	4	4	8

www.manaraa.com

Clearly, compared to *Normal Routing* (NR), the proposed schemes – *Dynamic NIx-Vector Routing* ($DNVR$), *Multipath Dynamic NIx-Vector Routing (using all paths)* ($MDNVR$), and *Multipath Dynamic NIx-Vector Routing (using* $\lceil X/k \rceil$ *paths)* ($MDNVRk$), can reduce the worst case switch load observed in fat-trees.

To illustrate we compute these bounds, for the four policies, for the fat-trees pictured in Figures 3.2-3.4 and $FT(8,3)$ (not pictured). The results are summarized in Table 3.2.

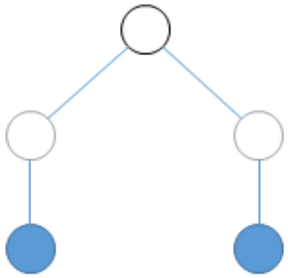


Figure 3.2: A simple fat-tree *FT(2, 2)* with 2 nodes and 3 switches.

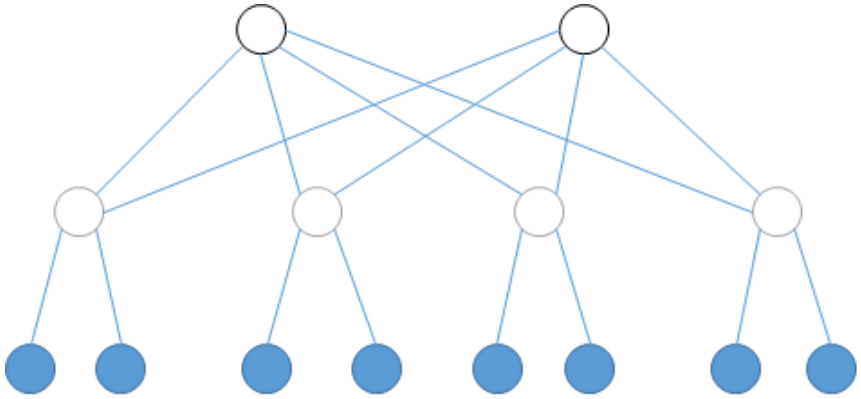


Figure 3.3: An example fat-tree *FT(4, 2)* with 8 nodes and 6 switches.

www.manaraa.com

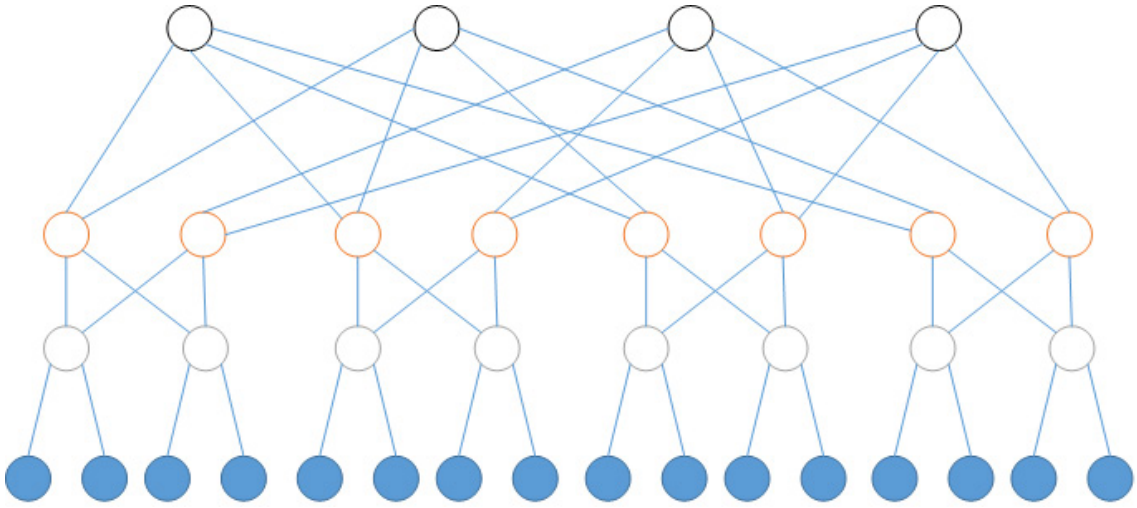


Figure 3.4: An example fat-tree *FT(4, 3)* with 16 nodes and 20 switches.

3.4 Conclusions

In this chapter, we established a relationship between the proposed multipath routing schemes and the avoidance of TCP Incast's onset. First, we noted, using Kulkarni's model of synchronized, many-to-one TCP flows, that the onset of Incast is highly correlated to flow packet loss, and consequently, that strategies that reduce packet loss, such as increasing switch buffer size, delay the onset of Incast. Next, we observed, from well-known results on switch buffer sizing, that the loss rate of TCP flows can be reduced by reducing switch loads. Finally, we defined key fat-tree routing terminology, and showed by formal analysis that, given $FT(m, n)$, for all traffic matrix TM, the worst-case loads – the loads most responsible for TCP Incast – are reduced by our proposed routing schemes.

www.manaraa.com

CHAPTER 4
THE FRONT-BACK ALGORITHM AND ITS PERFORMANCE EVALUATION

In this chapter, we investigate a novel "front-back" approach to minimizing the packet reordering introduced by multipath routing. We follow this with a brief discussion of integration issues, and a performance comparison with existing algorithms. We conclude with a discussion of the algorithm's extension to *N*-paths.

4.1 Reordering Avoidance: The Front-Back Algorithm

A key problem, faced by all approaches to routing packets over multiple paths, is packet reordering. Although in data centers shortest path latencies are typically small and uniform due to the data center's regular structure, flow packets taking different paths may still arrive out of order due to differences in the queueing delays encountered in switches along their paths. Returning packets to their original order consumes time, buffer, and computing resources. It may also drastically reduce throughput if it persists as TCP, which cannot distinguish between lost and reordered packets, reduces its congestion window, and begins unnecessarily retransmitting packets that it perceives have been lost [85, 86].

Our approach to mitigating reordering's effects is two pronged:

1. It opens separate TCP connections on each path to decouple the paths' transport layers, and
2. It distributes data across the paths in a manner that ensures that the work that must be done to return the received packets to their original order is minimized.

www.manaraa.com

The approach assumes that the receiving node maintains a buffer of the same size as the block data to be transmitted, and that the data is barrier-synchronized, an assumption consistent with typical data center traffic. As the case in which only two paths are available is significant in its own right, for simplicity we will explain our *Front-Back Algorithm*'s operation in this case first. The *N*-paths ($N > 2$) extension is provided in Section 4.4.

The 2-path *Front-Back Algorithm* ($FB2$) works as follows:

Step 1: Open separate TCP connections for Path 1 and Path 2.

Step 2: Simultaneously begin data transfer from the front of the data block on Path 1 and the back of the data block on Path 2.

Step 3: Complete transfer when the transfer streams "meet" in the middle of the block.

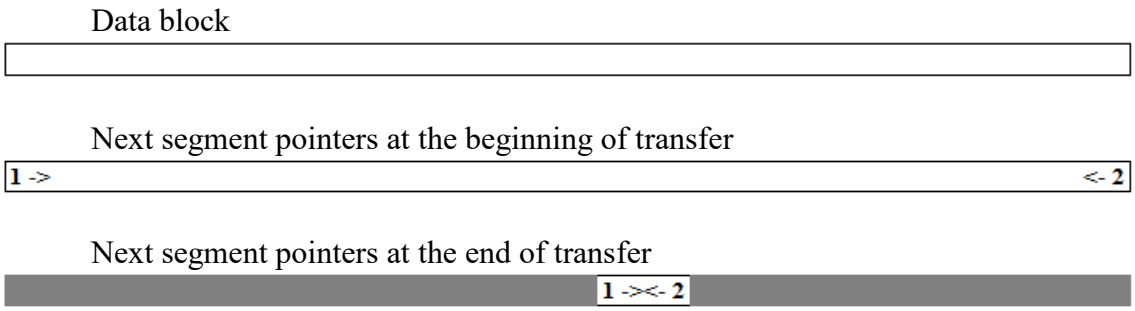


Figure 4.1: An illustration of the 2-path *Front-Back Algorithm*'s operation.

The algorithm's operation is illustrated in Figure 4.1. Let the first bar represent the block data to be transferred. As the second bar depicts, when transfer begins, data transfer on Path 1 and data transfer on Path 2 begin simultaneously from the front and back of the data block, proceeding forward and backward, respectively. As depicted on the third bar, transfer completes when the two transfers "meet" somewhere in the middle of the block.

www.manaraa.com

As Figure 4.1 highlights, at termination, the 2-path *Front-Back Algorithm* has partitioned the data block into two sub data blocks, one containing those segments that were transferred via Path 1, and another containing those segments that were transferred via Path 2. If the path conditions – for instance, the bottleneck capacities and end-to-end delays of the paths – were to change, so could the partition. The same can be said for all *N*-path partitioning algorithms p for mapping data block segments to the *N* paths.

How should we judge the performance of these algorithms? We argue that as the receiving application is assumed to have a buffer in which to reorder the block data arriving from the *N*-paths the appropriate measures are *finish time* – the time at which the reassembly buffer fills, and *disorder* – measured by the amount of effort the application must exert to fill the buffer.

Naturally, finish times are highly dependent on channel conditions. Nonetheless, a poor path partition, say choosing to send all block data via only the slowest available path, will affect it. Disorder seems easier to measure. The receiving application cares little how the reassembly buffer fills so long as it fills in the expected order. We measure disorder – effectively, deviations from the expected order – by counting the number of times the reassembly buffer's *N*-path pointers must "jump" from one position in the buffer to another to complete reassembly of the data received from the *N* paths given path conditions pc. We term these jumps, *context switches*, and denote them by $q(p, pc)$.

In the best case, no context switches are required, as data arrives in the expected order, and the pointers simply increment from their initial values until the buffer fills. In the worst case, the context switches after every segment arrives. This trivially occurs in the 2-path case, for instance, when under ideal path conditions, the partitioning algorithm sends

www.manaraa.com

all even indexed data segments on one channel, and all odd indexed data segments on the other. Then, upon receipt of each segment, each pointer must jump to the next even or odd buffer location before writing the received segment.

Let $P(p, pc)$ denote the partition of a data block produced by an *N*-path partitioning algorithm p under path conditions pc. Then we have the following, simple but useful lemma.

Lemma 4.1. Independent of the path conditions pc, the number of **context switches** $q(p, pc)$ required to fill the reassembly buffer given a data block partition $P(p, pc)$ produced by a *N*-path algorithm p is $q(p, pc) = |P(p, pc)| - N$.

Proof: Writing data in $|P(p, pc)|$ distinct buffer locations using *N* distinct pointers requires $|P(p, pc)| - N$ pointer jumps. □

Lemma 4.2. Independent of the path conditions pc, $q(FB2, pc) = 0$.

Proof: By construction, $|P(FB2, pc)| = 2$, hence by Lemma 4.1, $q(FB2, pc) = 0$. □

Theorem 4.1. With respect to both minimizing finish time, and minimizing packet disorder (as measured by context switches), $FB2$ is an optimal 2-path partitioning algorithm.

Proof: Consider first, the finishing time. By construction, if at any instant, one or more paths are able to deliver a segment to the application's reassembly buffer, under $FB2$ they deliver a new segment because, under $FB2$, the forward and backward paths' segment sequences don't overlap until all segments have been sent. Hence, $FB2$'s finishing time is at least as early as that of any other algorithm. Consider next, the disorder. As $q(FB2, pc) = 0$ by Lemma 4.2, $q(FB2, pc) \leq q(p, pc)$ for all p. The result follows. □

4.2 Integration with Multipath Routing

Integration of our *Front-Back Algorithm* with *Multipath Dynamic NIx-Vector Routing* and *Dual IPv4/IPv6 Routing* is not difficult. For *Multipath Dynamic NIx-Vector*

www.manaraa.com

Routing (using $\lceil X/k \rceil$ paths), the $\lceil X/k \rceil$ paths are randomly selected from the multiple shortest paths discovered by BFS for each source-destination pair. For *Dual IPv4/IPv6 Routing*, we apply the *Front-Back Algorithm* by letting the "sending from front of file" part use IPv4 and "sending back from end of file" part use IPv6. As illustrated in Figure 4.2, the front-back functionality could be implemented as session layer between the application and transport layers. This session layer would handle block partitioning and reassembly, and keep count of the total number of segments transferred and successfully received on all paths to ensure timely termination of all TCP connections.

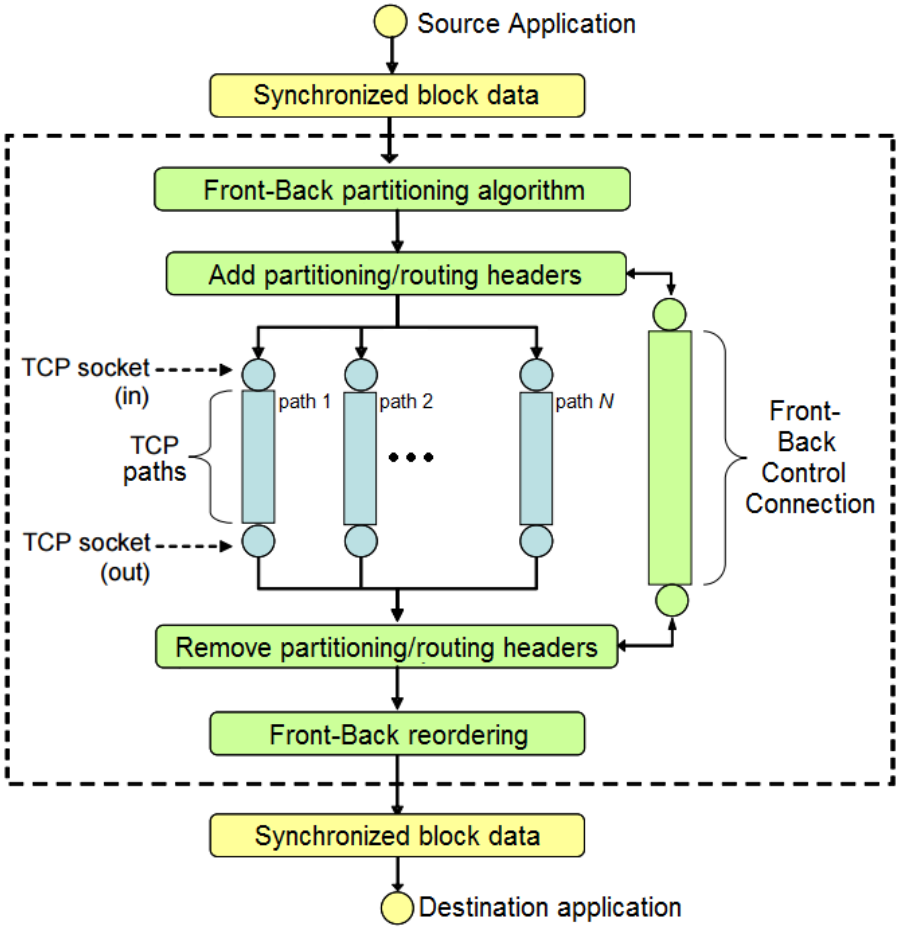


Figure 4.2: *Front-Back Algorithm* architecture.

www.manaraa.com

4.3 Comparison with Existing Algorithms

In this section, we contrast the performance of our *Front-Back Algorithm* to that of other approaches to combating multipath reordering. Existing schemes fall roughly into three categories: (1) those that attempt to avoid the problem by ensuring that most packets from the same flow follow the same path, (2) those that attempt to manage it by coordinating path assignment with path characteristics, and (3) those that attempt to overcome it by opening a separate TCP flow for each path and then more carefully assigning segments to paths.

Examples of schemes in the first category include: ECMP [31], which forces all packets from the same TCP flow to follow the same path and FLARE [87], which attempts to direct all bursts of packets (flowlets) from the same TCP flow to the same path. While constraining flows to specific paths indeed reduces multipath reordering, these approaches all have the downside of limiting switches' abilities to balance load to the granularity of flows.

Examples of schemes in the second category include: PATTHEL [88], which manages the multipath in the session layer, and Multi-Path TCP [89], which attempts to aggregate the available paths into a single TCP channel. Although these schemes typically do a good job of load balancing, because they balance loads packet-by-packet, they can induce a high degree of packet reordering.

Schemes in the third category, tend to view multipath reordering as a file partitioning problem. History-based TCP [90], partitions data blocks to be transferred into sub-blocks proportional to each path's rate and then transfers the blocks on these paths. When rates fluctuate, it performs poorly because the paths' transfers no longer finish concurrently, forcing the faster paths to idle, or "context switch" to another path's block. Arrival-time matching load balancing [91] attempts to fix the problem by maintaining

www.manaraa.com

running estimates of each path's delay, and adopting a "least delay path" selection rule. This scheme fares poorly in instances where path delays change rapidly. Finally, Dynamic TCP [90] attempts to avoid the adaptation problem completely by adopting a "last idle path" selection rule and keeping the transferred sub-data blocks' size small. This limits the amount of disorder, but generates large overheads due to the large number of block requests needed to keep the paths busy.

Our *Front-Back Algorithm* belongs to this third category, but its approach to the reordering problem is different than those of the other category 3 approaches. Whereas the other policies attempt to limit reordering by partitioning the data block to be sent to match the path characteristics, e.g., partitioning data into blocks proportional to the paths' rates [90], or partitioning data to equalize the path delays it experiences [91], our algorithm aims to partition data to minimize the receiver's reassembly effort as measured by context switches. The following example highlights the potential dangers of partitioning to match path characteristics as opposed to partitioning to minimize reassembly effort.

Example 1: 200000 segments of data are to be transferred over two paths. Based on the available estimates, the sending node believes that the path throughputs are identical with rate = 10 segment/s.

Two partition algorithms are considered for assigning segments to paths, *Divide2* and 2-path *Front-Back*. In each round *Divide2* assigns segments to the paths in proportion to the paths' estimated throughputs. When a path completes its assigned segments, *Divide2* splits the unfinished portion of the other path's assigned segments between the two paths. As such it can be viewed as an adaptive 2-path version of the history-based TCP parallel access algorithm in [90]. In each round, *Front-Back* simply assigns segments front-to-back, to one path, and back to front, to the other.

www.manaraa.com

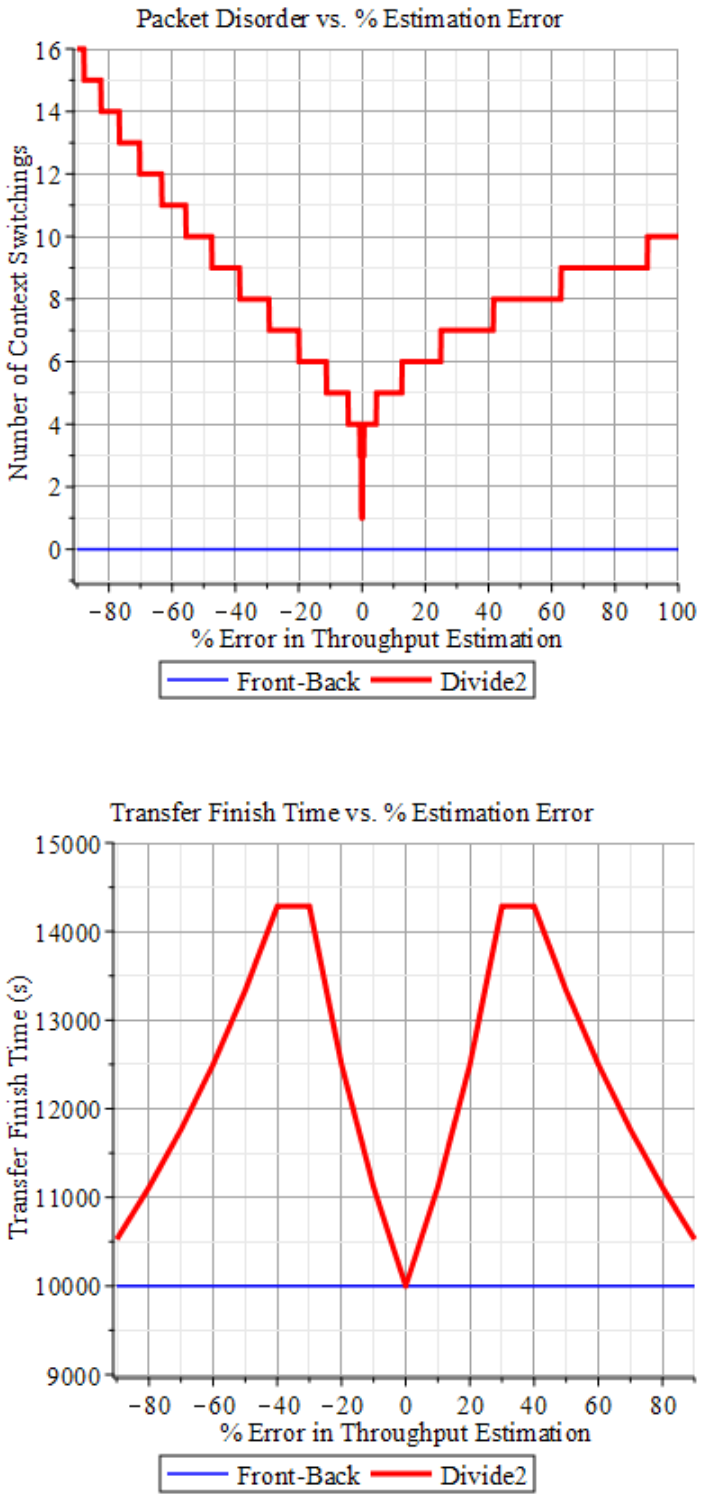


Figure 4.3: Packet Disorder and Transfer Finish Time as functions of throughput estimation error for *Front-Back* and *Divide2*.

www.manaraa.com

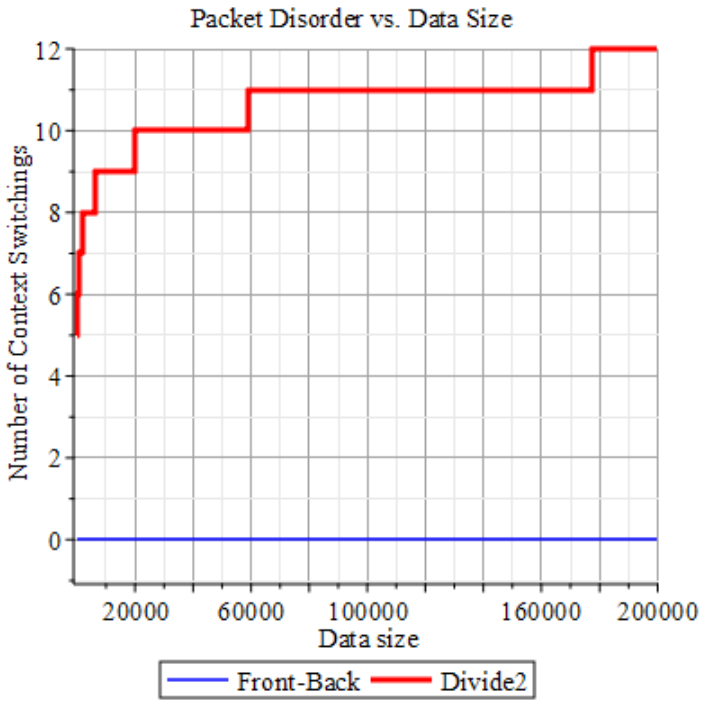


Figure 4.4: Packet Disorder as a function of data size for *Front-Back* and *Divide2*.

Algorithm performance is evaluated by two metrics: Packet Disorder, and Transfer Finish Time. As outlined in Section 4.1, Packet Disorder is measured by counting the number of context switches (reassembly buffer pointer "jumps") required to reassemble the received segments, the more work required, the more disordered the segments. Transfer Finish Time is the time at which all 200000 segments have been ordered and delivered to the application.

Plots of the two algorithms' Packet Disorder and Transfer Finish Times, versus the percentage error of the algorithms' estimates of Path 2's throughput relative to Path 1's throughput, are shown in Figure 4.3. The plots not only demonstrate – consistent with Theorem 4.1 – the superiority of the *Front-Back Algorithm* with respect to these two

www.manaraa.com

measures, but highlight the danger of relying on channel estimates to ensure in-order delivery of flow segments traversing multiple paths. Clearly, *Divide2*'s Packet Disorder and Transfer Finish Time are quite sensitive to quality of its channel estimates. As illustrated in Figure 4.4, which plots Packet Disorder as a function of the data transferred, given that the algorithms' estimates of Path 2's throughput relative to Path 1's throughput have a 50% error, this sensitivity only grows as the amount of data to be transferred grows.

4.4 Further Development – Generalization to *N*-paths

4.4.1 How the Generalization Works

In this section we investigate the extension of the *Front-Back Algorithm* to N paths ($N > 2$). The N-path *Front-Back Algorithm* (FBN) works as follows:

1. Form pairs (groups) of paths such that the largest difference among the groups' net throughputs is smallest. When N is odd, one group will contain a single path.
2. Partition the data block to be transferred proportional to the groups' net throughputs.
3. For the partitions obtained, run the *Front-Back Algorithm* on each in parallel. When N is odd, the partition with one assigned path transfers data normally.
4. When a group finishes transferring its partition's data, merge it with the group with the largest unfinished partition, repartition the merged group's unsent data among the merged group's paths using the process outlined in steps 1 and 2, and run the *Front-Back Algorithm* on the new groups' partitions.
5. Repeat this process until data transfer is complete.

The algorithm's operation in the case of four paths is illustrated in Figure 4.5. Assume that we have four paths with throughputs 3, 4, 6 and 7, respectively. Let the first

www.manaraa.com

bar represent the block data to be transferred. As the second bar depicts, we form path pairs [1, 4] and [2, 3], and partition the file by making the first 1/2 Partition A, and the rest Partition B. As the third bar depicts, we run the *Front-Back Algorithm* in partitions A and B in parallel. Assume that Path 1 experiences congestion so that Partition B finishes first. At this point, we repartition the remainder of Partition A among all paths and continue. To do the splitting, we form pairs [1, 3], [2, 4] (instead of [1, 2], [3, 4]) so that the largest difference between pairs' throughputs is smallest. Data transfer then continues as shown in the fourth bar. This process repeats as necessary, until data transfer is complete.

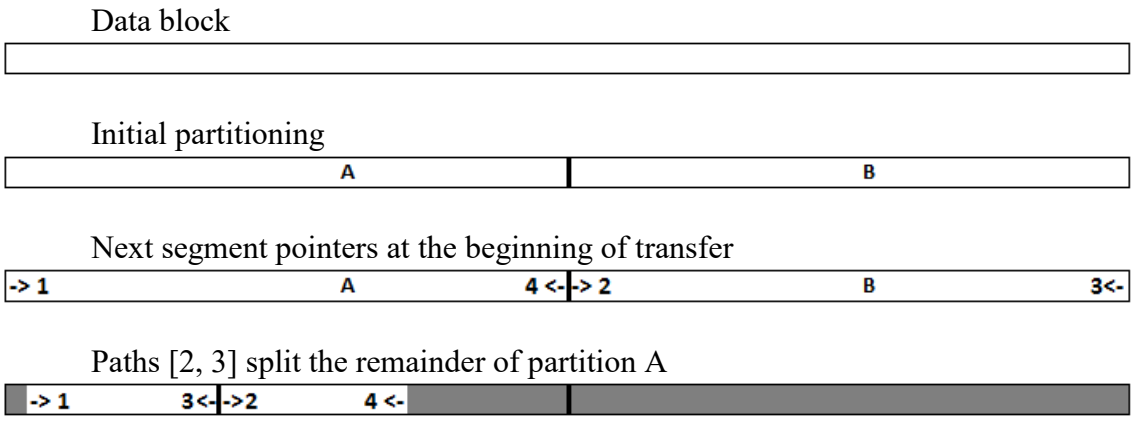


Figure 4.5: An illustration of the *N*-path *Front-Back Algorithm*'s operation.

We conjecture that Theorem 4.1, which we proved for the 2-path *Front-Back Algorithm*, also holds for *N*-paths.

Conjecture 4.1. With respect to both minimizing finish time, and minimizing packet disorder (as measured by context switches), FBN is an optimal *N*-path partitioning algorithm. □

www.manaraa.com

The intuition underlying this conjecture goes as follows: The *N*-path *Front-Back Algorithm* starts by forming pairs (groups) of paths such that largest difference among the net throughputs of all path groups is smallest. When *N* is odd one group contains only one path. Then it partitions the data in proportion to these pair group throughputs. These two steps minimize the probability that any partition will be emptied early, thereby minimizing the expected number of context switches. Data is then transferred simultaneously within all partitions, using the 2-path *Front-Back Algorithm* which, by Theorem 4.1 minimizes both the packet disorder (as measured by context switches) and finish time.

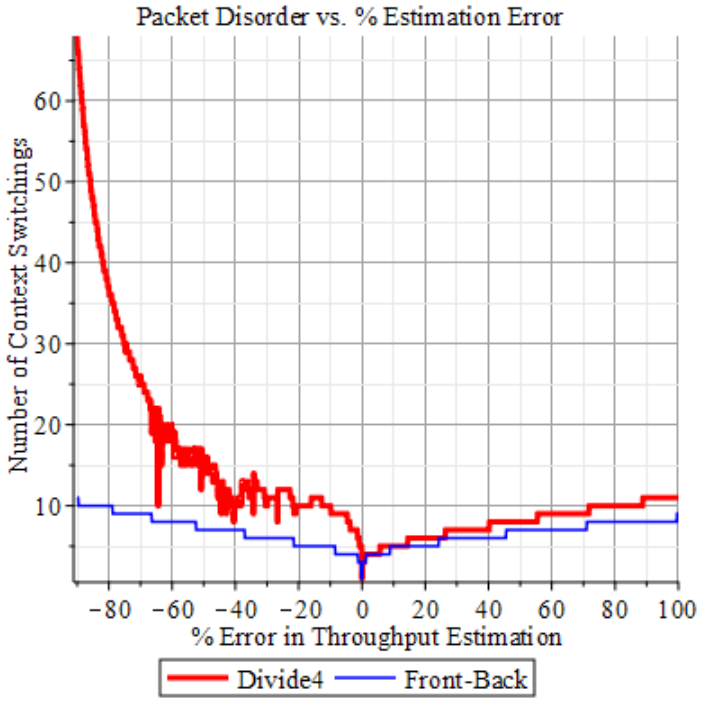
4.4.2 Comparison with Existing Algorithms

In this section we compare the Packet Disorder (as measured by context switches) of the 4-path *Front-Back Algorithm* and *Divide4*. *Divide4* is a 4-path version of the *Divide2* algorithm described in Section 4.3.

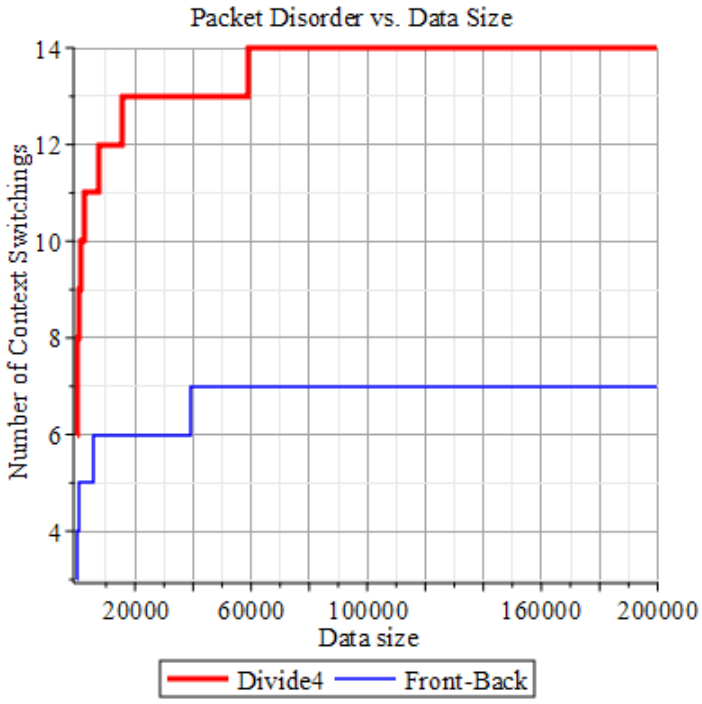
Example 2: 200000 segments of data are to be transferred over four paths. Based on the available estimates, the sending node believes that all paths throughputs are identical with rate = 10 segment/s.

Two partition algorithms are considered for assigning segments to paths, *Divide4* and 4-path *Front-Back*. In each round *Divide4* assigns segments to the paths in proportion to the paths' estimated throughputs. When a path completes its assigned segments, *Divide4* splits the largest unfinished portion of the other paths' assigned segments with the unfinished portion's path. As such it can be viewed as an adaptive 4-path version of the history-based TCP parallel access algorithm in [90].

www.manaraa.com



(a)



(b)

Figure 4.6: Packet Disorder as a function of (a) throughput estimation error and (b) data size for *Front-Back* and *Divide4*.

www.manaraa.com

Algorithm performance is evaluated by the metric of Packet Disorder. Once again, Packet Disorder is measured by counting the number of context switches (reassembly buffer pointer "jumps") required to reassemble the received segments, the more work required, the more disordered the segments.

Figure 4.6 (a) plots the two algorithms' Packet Disorder versus the percentage error of the algorithms' estimates of Path 1's throughput relative to the other paths' throughputs. Figure 4.6 (b) plots the two algorithms' Packet Disorder as a function of the data transferred, given that the algorithms' estimates of Path 1's throughput relative to the other paths' throughputs have a 50% error. These plots, once again, highlight the superiority of the *Front-Back Algorithm* and the danger of relying on channel estimates to ensure in-order delivery of flow segments traversing multiple paths.

4.5 Conclusions

In this chapter, we investigated a novel "front-back" approach to minimizing the packet reordering introduced by multipath routing. We established the optimality of an algorithm implementing our front-back approach for 2 paths with respect to the minimization of packet disorder (as measured by context switches) and finish time, and then briefly discussed how this *Front-Back Algorithm* could be integrated into existing protocols. The *Front-Back Algorithm*'s performance was then contrasted to other proposed algorithms for combating multipath reordering, and examples highlighting its advantages with respect to minimizing packet disorder and transfer finish time, were presented. We concluded the chapter with a discussion of the *Front-Back Algorithm*'s N-path extension ($N > 2$) and examples illustrating the extension's potential advantages.

www.manaraa.com

CHAPTER 5

PERFORMANCE ANALYSIS

We begin this chapter with an analysis of the proposed routing schemes. We focus on their worst-case loading of certain network resources – expressed as oblivious performance ratios (OPRs). We then explore typical Incast traffic patterns in data center networks, and describe a novel method of traffic matrix decomposition to help visually illustrate and classify traffic patterns. Potential benefits of our schemes are assessed through ns-3 simulations on fat-trees under a variety of traffic conditions. Results indicate that over a variety of experimental conditions, the proposed schemes reduce the incidence of TCP Incast compared to standard routing schemes.

5.1 The Oblivious Performance Ratio (OPR)

5.1.1 Definitions of the OPR

In Chapter 3, we defined key fat-tree routing terminology for $FT(m,n)$. In this section, we extend [84]'s approach to assessing worst case link loads – formally defined as oblivious performance ratios (OPR) – to that of worst case switch loads.

We begin by reviewing definitions of performance ratios from [84]. Given traffic matrix TM and routing r, the **performance ratio (link)** measures how far r is from being optimal on TM, with respect to link load. It is defined as the maximum link load of r divided by the min-max optimal link load on TM [92]:

$$PERF(r,TM) = \frac{MLOAD(r,TM)}{OPTU(TM)}. \quad (5.1)$$

www.manaraa.com

Intuitively, $PERF(r, TM) \geq 1$. It equals 1 if and only if the routing r achieves the min-max optimal link load on TM.

Definition 5.1 [84]**.** Given routing r, the maximum performance ratio (link) over all possible traffic matrices is defined as the **oblivious performance ratio (link)** [92]:

$$PERF(r) = \max_{For\ all\ TM} \{PERF(r, TM)\}. \quad (5.2)$$

Similarly, given traffic matrix TM and routing r, the **performance ratio (switch)** measures how far r is from being optimal on TM, with respect to switch load. It is defined as the maximum switch load of r divided by the min-max optimal switch load on TM:

$$PERFSW(r, TM) = \frac{MLOADSW(r, TM)}{OPTUSW(TM)}. \quad (5.3)$$

Intuitively, $PERFSW(r, TM) \geq 1$. It equals 1 if and only if the routing r achieves the min-max optimal switch load on TM.

Definition 5.2. Given routing r, the maximum performance ratio (switch) over all possible traffic matrices is defined as the **oblivious performance ratio (switch)**:

$$PERFSW(r) = \max_{For\ all\ TM} \{PERFSW(r, TM)\}. \quad (5.4)$$

A routing r that achieves the min-max optimal switch load over all TM, i.e. $PERFSW(r) = 1$, is an optimal routing scheme for the network.

Next, we extend the definition of oblivious performance ratio (switch) to different levels of switches in the fat-tree network.

Definition 5.3. Given routing r on $FT(m, n)$, with m a power of 2 and $n \geq 1$, for all TM, the oblivious performance ratio of $level\ t$ switches, $0 \leq t \leq n - 1$, is defined as

www.manaraa.com

$$PERFSW_t(r) = \max_{For\ all\ TM} \left\{ \frac{MLOADSW_t(r, TM)}{OPTUSW_t(TM)} \right\}. \quad (5.5)$$

By construction, traffic load on the "edge" switches ($level\ n-1$) of $FT(m,n)$ is not affected by the routing scheme, Therefore, this ratio is always 1:

$$PERFSW_{n-1}(r) = 1 \text{ for all } r. \quad (5.6)$$

5.1.2 Analysis on the OPR

We now analyze the oblivious performance ratios of different routing schemes. Consider first *Multipath Routing via Dynamic NIx-Vectors (using all paths)* ($MDNVR$). In Corollary 3.2 and Theorem 3.5, we showed that given $FT(m,n)$, for all traffic matrices TM, $MLOAD(MDNVR, TM) = OPTU(TM)$ and $MLOADSW(MDNVR, TM) = OPTUSW(TM)$. It follows from the definition of oblivious performance ratio that:

Theorem 5.1. *Given Multipath Routing via Dynamic NIx-Vectors (using all paths) (MDNVR) on $FT(m,n)$, with m a power of 2 and $n \geq 1$, for all TM,*

$$PERF(MDNVR) = PERFSW(MDNVR) = 1. \square \quad (5.7)$$

Similarly, in other theorems and corollaries from Chapter 3, we derived bounds for *Normal Routing, Dynamic NIx-Vector Routing* and *Multipath Routing via Dynamic NIx-Vectors (using $\lceil X/k \rceil$ paths)*, respectively. In addition, we showed that those bounds are tight. Thus, from the definitions of oblivious performance ratios, we have:

Theorem 5.2. *Given Normal Routing (NR) on $FT(m,n)$, with m a power of 2 and $n \geq 1$, for all TM, the oblivious performance ratio of level t switches, $0 \leq t \leq n-1$, satisfies*

$$PERFSW_t(NR) = \left(\frac{m}{2}\right)^{n-1-t}. \square \quad (5.8)$$

www.manaraa.com

Theorem 5.3. *Given Dynamic NIx-Vector Routing (DNVR) on* $FT(m,n)$*, with* m *a power of 2 and* $n \geq 1$*, for all TM, the oblivious performance ratio (link) satisfies*

$$PERF(DNVR) = \begin{cases} \left\lceil \sqrt{\frac{m}{2}} \right\rceil, & \text{if } n = 2 \\ \frac{m}{2}, & \text{if } n = 3 \end{cases}; \tag{5.9}$$

and the oblivious performance ratio of level t *switches,* $0 \leq t \leq n-1$*, satisfies*

$$PERFSW_t(DNVR) = \begin{cases} \min\left\{ \left\lceil \sqrt{\frac{m}{2}} \right\rceil, \left(\frac{m}{2}\right)^{n-1-t} \right\}, & \text{if } n = 2 \\ \min\left\{ \frac{m}{2}, \left(\frac{m}{2}\right)^{n-1-t} \right\}, & \text{if } n = 3 \end{cases}. \square \tag{5.10}$$

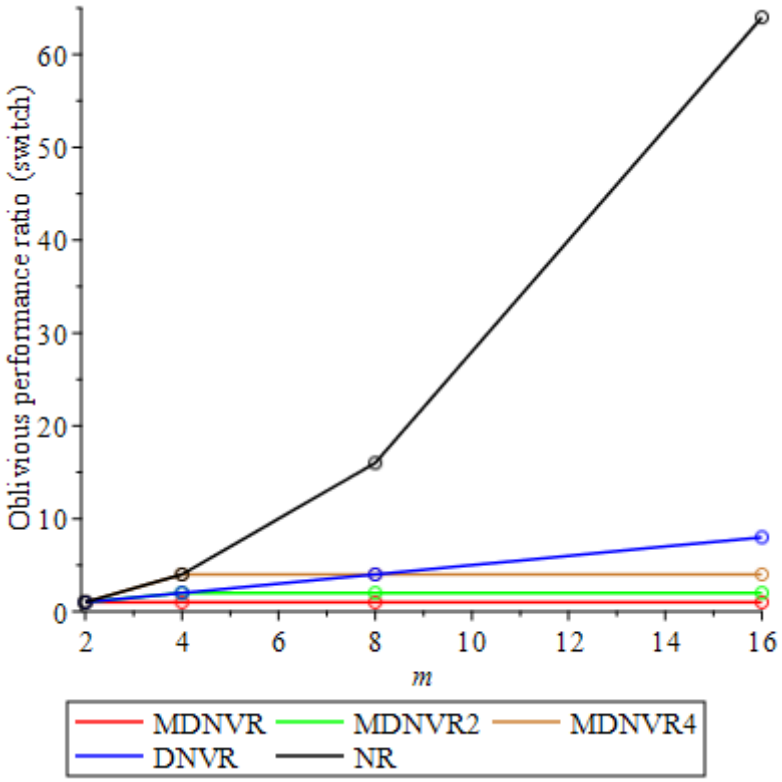


Figure 5.1: Comparison of the oblivious performance ratios (switch) on *FT(m, 3)*, for routing schemes *MDNVR, MDNVRk (k=2 and 4), DNVR* and *NR*.

www.manaraa.com

Theorem 5.4. *Given Multipath Routing via Dynamic NIx-Vectors (using* $\lceil X/k \rceil$ *paths) (*$MDNVRk$*) on* $FT(m,n)$*, with* m *a power of 2 and* $n \geq 1$*, for all* TM*, the oblivious performance ratio of level* t *links,* $0 \leq t \leq n-1$*, satisfies*

$$PERF_t(MDNVRk) = \min\left\{k, \left(\frac{m}{2}\right)^{n-1-t}\right\}; \tag{5.11}$$

and the oblivious performance ratio of level t *switches,* $0 \leq t \leq n-1$*, satisfies*

$$PERFSW_t(MDNVRk) = \min\left\{k, \left(\frac{m}{2}\right)^{n-1-t}\right\}. \square \tag{5.12}$$

In Figure 5.1, we plot and compare the oblivious performance ratios for m-port 3-trees $FT(m,3)$, versus m, for routing schemes $MDNVR$, $MDNVRk$ $(k = 2\ and\ 4)$, $DNVR$ and NR. From this figure, we can clearly see the advantages of our proposed schemes. When $m = 8$, for instance, simply distributing traffic across 2 paths using $MDNVRk$ $(k = 4)$ yields a four-fold reduction in the worst-case switch load compared to *Normal Routing*, thereby reducing the likelihood of Incast.

5.2 Traffic Patterns and Analysis

In this section, we explore typical traffic patterns in data center networks related to TCP Incast, and identify key traffic patterns for comparing routing schemes' performance.

5.2.1 Typical Incast Traffic Patterns

In a typical "barrier-synchronized request workload" Incast pattern, the client sends simultaneous requests to multiple servers, which then send back responses immediately. A distinguishing feature of this type of workload is that the client must wait for all servers' responses to arrive before sending a new batch of requests. Additionally, to meet certain

www.manaraa.com

deadlines, there are typically very tight time constraints, within each "batch", for receipt of the servers' responses. Consequently, servers' responses, within a single batch, are often almost simultaneous, and thus the traffic can be highly synchronized. Should this intense, synchronized traffic overflow one or more network switch buffers, enough servers may time out to initiate Incast.

For example: Suppose that each client sends requests to N servers simultaneously, and that the workload is barrier-synchronized. Suppose additionally, that there are 100 batches of such requests, and that whenever a server receives a request, it responds with $K\ MB$ of data. The data transferred in each batch is then $K \times N$ and, for all batches, the total amount of data transferred is $100 \times K \times N\ MB$.

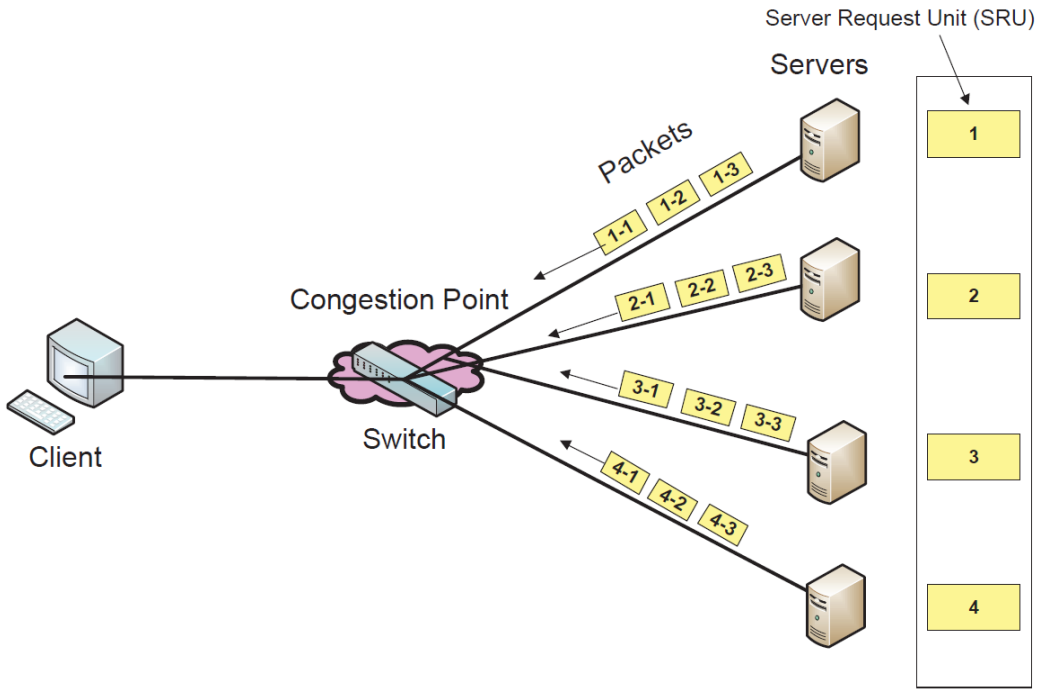


Figure 5.2: A typical TCP Incast network setting from [93], with one client requesting data from multiple servers through synchronized reads.

www.manaraa.com

Figure 5.2, from Zhang and Ansari [93], illustrates a typical network setting under which Incast could occur. They summarize the conditions favorable to the onset of TCP Incast as being:

1. high-bandwidth, low-latency links connected by switches with limited buffers;
2. parallel barrier-synchronized requests from clients;
3. servers returning a fragment of data block for each request.

Examples of previously studied Incast traffic patterns include, from:

(1) "ICTCP: Incast Congestion Control for TCP in Data-Center Networks" [21]:

- Traffic pattern: barrier-synchronized many-to-one
- Link characteristics: 1 Gbps with 100 μs round-trip delay
- Server request unit (SRU) size: 64 KB, 128 KB or 256 KB
- Switch buffer size: 85 KB per port
- Number of senders in parallel: 1 - 46
- Number of experimental rounds: 100

(2) "Preventing TCP Incast Throughput Collapse at the Initiation, Continuation and Termination" [94]:

- Traffic pattern: barrier-synchronized many-to-one
- Link characteristics: 1 Gbps with 100 μs round trip delay
- Server request unit (SRU) size: 10 KB or 100 KB
- Switch buffer size: 128 KB per port
- Number of senders in parallel: 1 – 100
- Number of experimental rounds: 100

www.manaraa.com

(3) "Fast and Cautious: Leveraging Multi-path Diversity for Transport Loss Recovery in Data Centers" [58]:

- Link characteristics: 1 Gbps with ~280 μs round trip delay
- SRU size: smaller than 100 KB for latency sensitive queries, and larger than 100 KB for background requests
- Switch buffer size: 128 KB per port
- Number of senders in parallel: 5
- Number of experimental rounds: 10

5.2.2 Worst Case Patterns

In the previous section, we briefly surveyed the characteristics of several traffic patterns capable of inducing Incast. These "barrier-synchronized many-to-one" request patterns are important in practice, but for the purpose of evaluating routing schemes' effects on switch loading, simpler traffic patterns suffice. We identify a few of these simpler patterns in this section.

We begin by observing that for our purposes – and in practice, given that Incast is worst-case phenomenon triggered by extreme switch loading – traffic patterns that induce the same maximum switch loads can be viewed as equivalent. Formally we say that traffic patterns are **equivalent patterns** for a given routing r if they induce the same baseload normalized maximum switch load $\frac{MLOADSW(r,TM)}{baseload(TM)}$. Two classes of equivalent patterns are of particular interest.

Worst-case patterns are those that, for a given routing r, induce the largest baseload normalized maximum switch load $\frac{MLOADSW(r,TM)}{baseload(TM)}$.

www.manaraa.com

Best-case patterns are those that, for a given routing r, induce the smallest baseload normalized maximum switch load $\frac{MLOADSW(r,TM)}{baseload(TM)}$.

To gain insight into the effects of particular traffic patterns, and to better visually illustrate and classify them, we have found it helpful to decompose their corresponding traffic matrices in a manner that highlights the switch levels loaded by each fat-tree send-receive pair. By definition, an m-port n-tree $FT(m,n)$ contains n switch levels. Send-receive pairs that traverse $level\ 0$ (the core switches), traverse all $n-1$ lower levels. Send-receive pairs that traverse $level\ 1$ (the highest-level aggregation switches), but not $level\ 0$, traverse $n-2$ lower levels, and so on. As the pairs traversing $level\ 0$ are disjoint from the pairs traversing $level\ 1$ but not $level\ 0$, and the pairs traversing $level\ 2$ but not $level\ 1$, and so on, it follows that any traffic matrix for $FT(m,n)$ can be decomposed as

$$TM = TM_0 + TM_1 + \cdots + TM_{n-1} \tag{5.13}$$

where $TM_i, i = 0, 1, \dots, n-1$, indexes those send-receive pairs in TM that traverse $level\ i$, but no higher level.

The utility of this decomposition is best illustrated by example. Consider the 4-port 3-tree $FT(4,3)$ depicted in Figure 5.3. As this tree has 3 levels, all TM on $FT(4,3)$ can be decomposed as $TM = TM_0 + TM_1 + TM_2$. To more conveniently display this decomposition in a single figure we adopt the following convention:

- Source-destination pairs in TM_2 will be labeled E (because they correspond to communications that only traverse $level\ 2$ (edge) switches.
- Source-destination pairs in TM_1 will be labeled A (because they correspond to communications that traverse $level\ 1$ (aggregation) but not $level\ 0$ (core) switches.

www.manaraa.com

- Source-destination pairs in TM_0 will be labeled I (because they correspond to interpod communications that traverse a $level$ 0 (core) switch.

Superimposing the decomposed TMs on top of each other we obtain, for the node labeling in Figure 5.3, the generic traffic matrix representation depicted in Figure 5.4.

To examine the decomposition of a specific traffic matrix one simply superimposes the specific matrix on the generic representation. Consider, for instance, the traffic matrices corresponding to the test patterns *Stride(2)*, and *Stride(4)* in [31]. Let the nodes in $FT(4, 3)$ be labeled from left to right as 0, 1, …, 15. Under *Stride(2)*, every node i sends a unit of traffic to node '$(i + 2)$ mod 16'. Under *Stride(4)*, every node i sends a unit of traffic to node '$(i + 4)$ mod 16'. More concretely, *Stride(2)* specifies that each node in the smallest (1 level) subtree X' send traffic to the node in the same position in the adjacent 1 level subtree $(X' + 1)$ mod 8. Similarly, *Stride(4)* specifies that each node in the 2 level subtree X send traffic to the node in the same position in the adjacent 2 level subtree $(X + 1)$ mod 4. Such patterns are often referred to as "permutation traffic" because each node sends traffic to a distinct destination. Plotting *Stride(2)* and *Stride(4)* on Figure 5.4 we obtain Figures 5.5(a) and 5.5(b). Close examination of these figures, indicates that *Stride(4)* only generates "inter-pod" traffic, while *Stride(2)* generates a mixture of "inter-pod" and "same aggregation switch" traffic.

To use the decomposition to identify worst-case and best-case traffic patterns, observe from Figures 5.3 and 5.4 that:

- "Same edge switch" traffic only loads edge switches.
- "Same aggregation switch" traffic loads both edge and aggregation level switches.
- "Inter-pod" traffic loads edge, aggregation, and core level switches.

www.manaraa.com

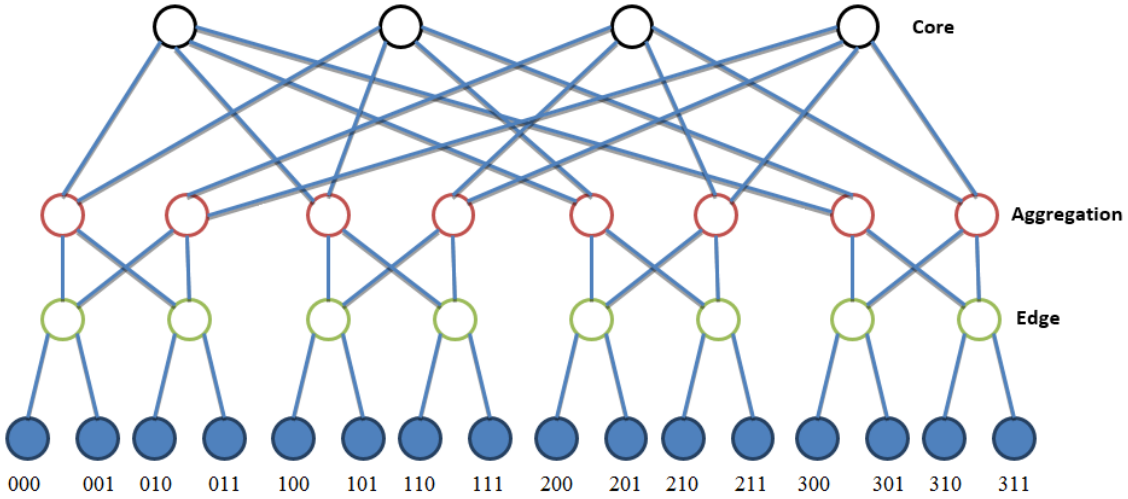
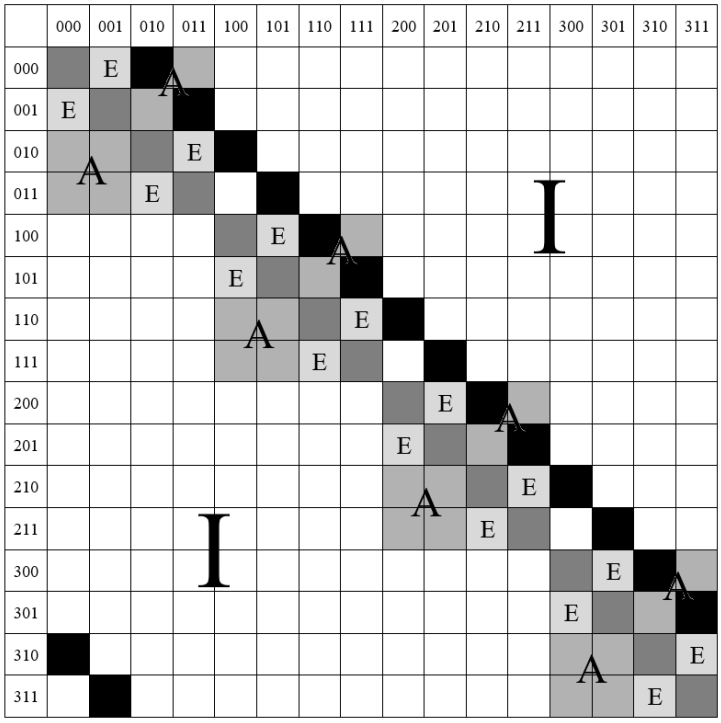


Figure 5.3: An example *FT(4, 3)* fat-tree network.

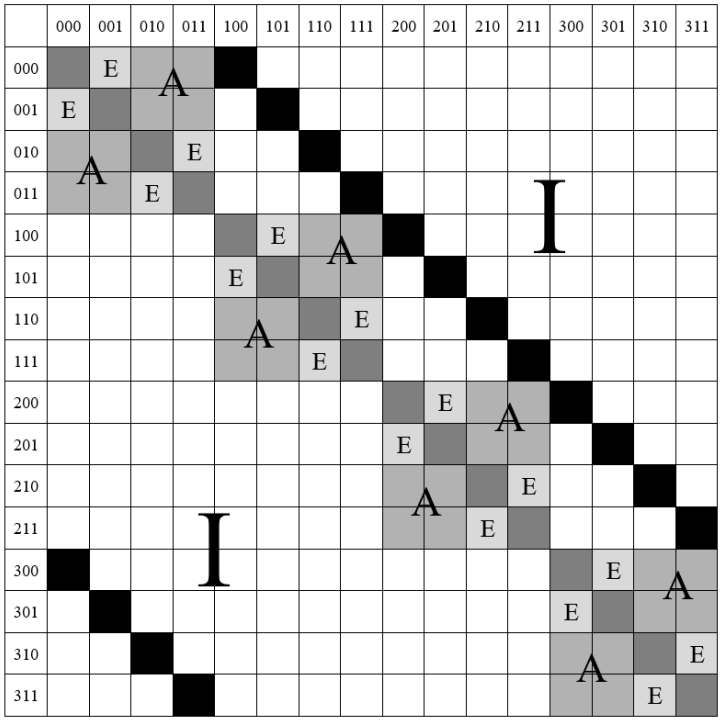
	000	001	010	011	100	101	110	111	200	201	210	211	300	301	310	311
000		E	A	A												
001	E		A	A												
010	A	A		E												
011	A	A	E									I				
100						E	A	A				I				
101					E		A	A								
110					A	A		E								
111					A	A	E									
200										E	A	A				
201									E		A	A				
210									A	A		E				
211				I					A	A	E					
300				I										E	A	A
301													E		A	A
310													A	A		E
311													A	A	E	

Figure 5.4: Traffic matrix decomposition for *FT(4, 3)*.
Symbols: E – same edge switch, A – same aggregation switch, I – inter-pod.

www.manaraa.com



(a) *Stride(2)*



(b) *Stride(4)*

Figure 5.5: Traffic matrix decomposition for (a) *Stride(2)* and (b) *Stride(4).*

www.manaraa.com

Consider first the class of best-case patterns. As discussed in Section 5.1.1, traffic load on the "edge" switches ($level\ n-1$) does not depend on the routing scheme. Therefore, traffic patterns that contain only "same edge switch" traffic ("E" symbols on Figure 5.4), belong to this class and impose the same switch load as optimal routing.

Consider next the class of worst-case patterns. Observe first that worst-case patterns should contain "inter-pod" traffic ("I" symbols on Figure 5.4), because this type of traffic loads switches at all three levels. Additionally, as $baseload(TM)$ is calculated as the maximum of all row and column sums, to achieve the highest baseload normalized maximum switch load, all columns and rows of worst-case patterns should have the same sum. For example, we can place a single block of traffic in each column and row, within the "I" symbol areas.

Clearly, the traffic pattern *Stride(4)*, which specifies that each node in the 2 level subtree X of $FT(4,3)$ send a unit of traffic to the node in the same position in the adjacent 2 level subtree $(X+1) \bmod 4$, satisfies this requirement and is thus a worst-case pattern. The following theorem generalizes this observation to $Stride\left(\left(\frac{m}{2}\right)^{n-1}\right)$, under which every node i sends a unit of traffic to node $\left(i+\left(\frac{m}{2}\right)^{n-1}\right) \bmod \left(2\times\left(\frac{m}{2}\right)^{n}\right)$.

Theorem 5.5. *Given $FT(m,n)$, with m a power of 2 and $n \geq 1$, the traffic pattern $Stride\left(\left(\frac{m}{2}\right)^{n-1}\right)$ achieves the baseload normalized maximum switch load $\frac{MLOADSW(r,TM)}{baseload(TM)}$ upper bounds in Table 3.1, for routing schemes $r = NR, DNVR, MDNVR$ and $MDNVRk$.*

Proof: Given $FT(m,n)$, $Stride\left(\left(\frac{m}{2}\right)^{n-1}\right)$ ("SMN") specifies that each source node in the $n-1$ level subtree X send a unit of traffic to the node in the same position in the adjacent $n-1$ level subtree $(X+1) \bmod m$. All connections are "inter-pod". As each node sends

www.manaraa.com

the same amount of traffic they all send $baseload(SMN)$. Additionally, we observe that the traffic sent by all nodes, which equals $baseload(SMN) \times 2 \times \left(\frac{m}{2}\right)^n$, must pass through $level$ 0 "core" switches.

Consider first *Normal Routing* (NR). In our proof of Theorem 3.1, we showed that the leftmost $level$ 0 switch is maximally loaded by TMs under which all source nodes' traffic is directed towards $n-1$ level sub-fat-trees other than their own. Because SMN is such a TM, we have $MLOADSW(NR, SMN) = baseload(SMN) \times 2 \times \left(\frac{m}{2}\right)^n$.

Consider next *Dynamic NIx-Vector Routing* ($DNVR$). By our proof of Theorem 3.2, because each node sends $baseload(SMN)$, each link in the network carries traffic to or from $\left\lceil \sqrt{\frac{m}{2}} \right\rceil$ source nodes for $n = 2$, and traffic to or from $\frac{m}{2}$ source nodes for $n = 3$. As each switch in $FT(m, n)$ has m links, the maximum switch load $MLOADSW(DNVR, SMN)$ is thus $baseload(SMN) \times \left\lceil \sqrt{\frac{m}{2}} \right\rceil \times m$ for $n = 2$, and $baseload(SMN) \times \frac{m}{2} \times m$ for $n = 3$.

Consider next *Multipath Routing via Dynamic NIx-Vectors (using all paths)* ($MDNVR$). As shown above, all nodes' traffic must pass through $level$ 0 "core" switches. Because $MDNVR$ achieves the min-max optimal switch load (Theorem 3.5), traffic load is uniformly distributed among the $\left(\frac{m}{2}\right)^{n-1}$ core switches. Hence each switch carries traffic $MLOADSW(MDNVR, SMN) = baseload(SMN) \times 2 \times \frac{\left(\frac{m}{2}\right)^n}{\left(\frac{m}{2}\right)^{n-1}} = baseload(SMN) \times m$.

Finally, consider *Multipath Routing via Dynamic NIx-Vectors (using $\lceil X/k \rceil$ paths)* ($MDNVRk$). First, note that there are $\left(\frac{m}{2}\right)^{n-1} \times m$ total links connected to the $level$ 0 "core" switches. In the worst case, all nodes' traffic is distributed over the same $\left(\frac{m}{2}\right)^{n-1} \times$

www.manaraa.com

$\frac{m}{k}$ links. Hence each $level\,0$ link carries $baseload(SMN) \times 2 \times \left(\frac{m}{2}\right)^n / \left(\left(\frac{m}{2}\right)^{n-1} \times \frac{m}{k}\right) = baseload(SMN) \times k$ traffic, and each $level\,0$ switch, with m links attached, carries $MLOADSW(MDNVRk, SMN) = baseload(SMN) \times k \times m$ traffic. □

Theorem 5.5 says that no TM induces higher baseload normalized maximum switch load in $FT(m,n)$ than $Stride\left(\left(\frac{m}{2}\right)^{n-1}\right)$. It is thus a worst case TM for these routings.

Table 5.1: Traffic matrix *Stride(4)* on the 4-port fat-tree *FT(4, 3)*.

Node #	000	001	010	011	100	101	110	111	200	201	210	211	300	301	310	311
000					1											
001						1										
010							1									
011								1								
100									1							
101										1						
110											1					
111												1				
200													1			
201														1		
210															1	
211																1
300	1															
301		1														
310			1													
311				1												

Next, we evaluate the performance ratios of different routing schemes under the worst-case traffic pattern *Stride(4)* ("$S4$") on the 4-port fat-tree $FT(4,3)$. The traffic matrix for *Stride(4)*, with each node sending one unit of traffic, is shown in Table 5.1. We use

www.manaraa.com

Maple, a powerful symbolic and numeric computing platform, for the evaluations. Source codes for the Maple programs can be found in the Appendix.

The routing schemes to be evaluated include: *Multipath Routing via Dynamic NIx-Vectors (using all paths) (MDNVR), Multipath Routing via Dynamic NIx-Vectors (using* $\lceil X/k \rceil$ *paths) (MDNVRk) with* $k = \frac{4}{3}, 2 \text{ and } 4$, *Dynamic NIx-Vector Routing (DNVR)* and *Normal Routing (NR).* Maple results are shown in Table 5.2. They are consistent with the results of Theorems 5.1 through 5.4.

Table 5.2: Maple evaluation results for comparing different routing schemes' oblivious performance ratios

Routing scheme	$MLOADSW_0(\cdot, S4)$	$MLOADSW_1(\cdot, S4)$	$PERFSW(\cdot, S4)$
NR	16	8	4
$DNVR$	8	8	2
$MDNVRk\ (k = 4)$	16	8	4
$MDNVRk\ (k = 2)$	8	8	2
$MDNVRk\ (k = \frac{4}{3})$	$\frac{16}{3}$	4	$\frac{4}{3}$
$MDNVR$	4	4	1

5.3 Validation by Simulations on the ns-3 Platform

In this section, we assess the potential benefits of our proposed routing schemes through ns-3 simulations on fat-trees under a variety of communication patterns. Results indicate that over a variety of experimental conditions, the proposed schemes reduce the incidence of TCP Incast compared to standard routing schemes.

www.manaraa.com

5.3.1 Simulation Setup

We use the ns-3 network simulator [95] (version 3.14.1) to validate our approach. ns-3 is an open, extensible discrete-event network simulation platform, developed primarily for networking research and educational use. It contains a variety of models for network protocols and routing. The simulator itself is written in C++, with optional Python bindings. Both C++ and Python are supported for ns-3 simulation scripts.

5.3.1.1 Simulation Parameters

Table 5.3: Parameters used in our ns-3 simulations.

Parameter	**Values for Random, Stride(2) and Stride(4)**	**Values for 3 Senders, 5 Senders and 7 Senders**
Link Bandwidth	1 Gbps	1 Gbps
Link Delay	500 μs	500 μs
Data to be sent from each node	2 MB (2,000,000 bytes)	262144 bytes × 10 blocks
Switch buffer size	varies from 10~90 packets	varies from 20~200 packets
TCP implementation	TCP Reno	TCP Reno
TCP max segment size	1452 bytes	1452 bytes
Max size of receiver window	65535 bytes	65535 bytes
Fast retransmit threshold	3 duplicate ACKs	3 duplicate ACKs
RTO_{min}	200 ms	200 ms

A ns-3 simulation was created from scratch, using the $FT(4,3)$ fat-tree topology as shown in Figure 1.4. The complete simulation code can be found in the Appendix.

Figure 5.6 shows a snapshot of our simulation topology, as displayed in the ns-3 PyViz [96] simulation animation interface. Table 5.3 summarizes the parameters used in our simulations. All links in the network are 1 Gbps, with a round-trip delay of 500 μs. We

www.manaraa.com

use the TCP Reno implementation with a maximum segment size (MSS) of 1452 bytes, and a minimum TCP re-transmission timeout (RTO) value of 200 ms. The TCP receiver window has a maximum size of 65535 bytes.

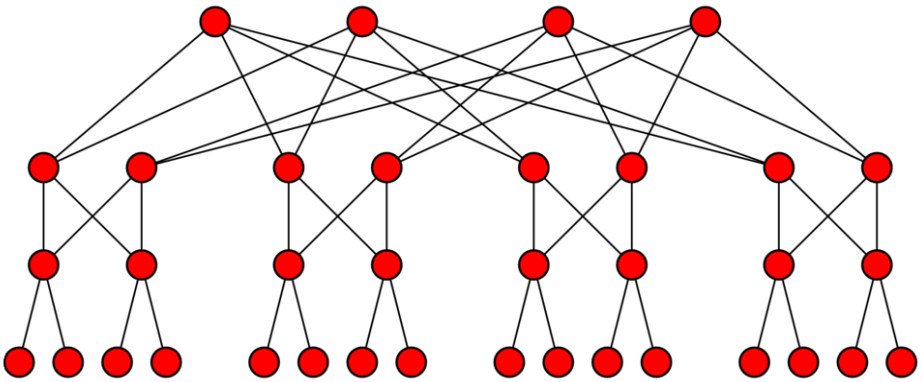


Figure 5.6: Our ns-3 simulation topology.

For traffic patterns *Random, Stride(2)* and *Stride(4)*, the amount of data to be sent from each node is 2 MB (2,000,000 bytes), and switch buffer sizes vary from 10 to 90 packets. For traffic patterns *3 Senders, 5 Senders* and *7 Senders*, each node sends 262144 bytes × 10 blocks of data, and switch buffer sizes vary from 20 to 200 packets.

5.3.1.2 Illustrations of the Proposed Schemes

The routing schemes to be investigated include:

1. *Normal Routing*, which corresponds to the single-path IPv4 table-based scheme,
2. *Normal Routing with ECMP,* which utilizes the per-packet ECMP feature in ns-3,
3. *Dynamic NIx-Vector Routing*, a NIx-vector routing variant that uses our modified BFS algorithm to select one path from the multiple available shortest paths,

www.manaraa.com

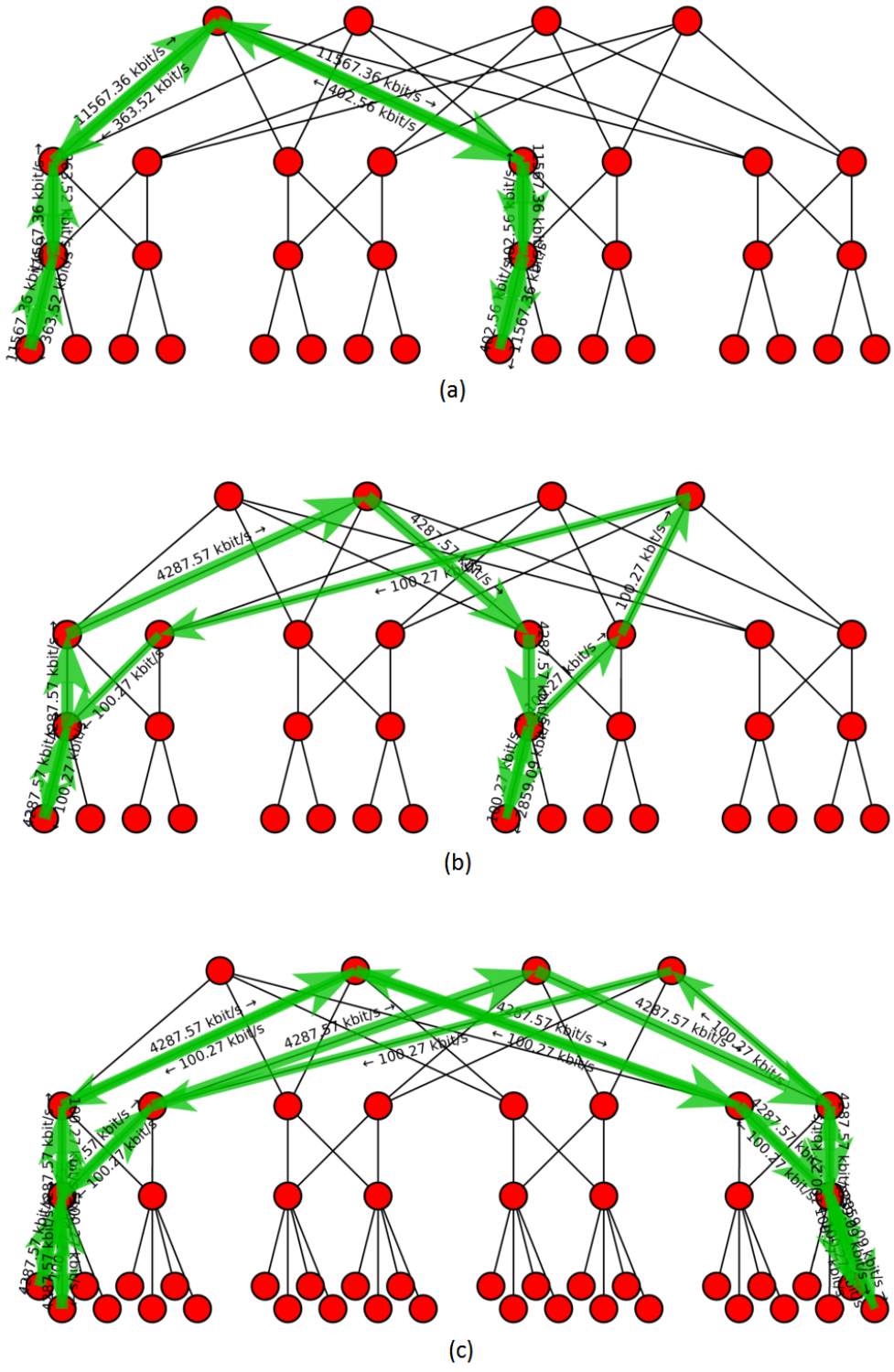


Figure 5.7: Path selection by different schemes in our simulation.
(a) *Normal Routing*, (b) *Dynamic NIx-Vector Routing,*
(c) *Multipath Routing via Dynamic NIx-Vectors.*

www.manaraa.com

4. *Multipath Routing via Dynamic NIx-Vectors*, which extends NIx-vector routing to the multipath case using the *Front-Back Algorithm* (introduced in Chapter 4) to distribute traffic across the shortest paths, and
5. *Dual IPv4/IPv6 Routing with Front-Back* utilizes both IPv4 and IPv6 to set up two different paths, and then use the *Front-Back Algorithm* to send data.

Figure 5.7 shows a "live" snapshot of our ns-3 simulation, with traffic flows between nodes highlighted to illustrate the path selection of the routing schemes.

Normal routing always picks the same shortest path from the four available for each source-destination pair. This can be seen in Figure 5.7 (a). To implement *Dynamic NIx-Vector Routing*, the BFS function in the existing ns-3 NIx-Vector Routing scheme was modified, so that every time it is called it returns a randomly selected shortest path from those discovered by BFS. An example is shown in Figure 5.7 (b). Choosing different random seeds results in different shortest paths being selected. Next, we apply the *Front-Back Algorithm*. The resulting *Multipath Routing via Dynamic NIx-Vectors* will further balance traffic, as illustrated in Figure 5.7 (c).

5.3.1.3 Traffic Patterns Investigated

We investigated the following traffic patterns from [31]. Let the nodes in $FT(4, 3)$ be labeled from left to right as 0, 1, …, 15, these patterns can be described as follows:

- *Random*: Every node i sends the same traffic to any other node in the network with uniform probability.
- *Stride(2)*: Every node i sends the same traffic to node '$(i + 2)$ mod 16'.
- *Stride(4)*: Every node i sends the same traffic to node '$(i + 4)$ mod 16'.

www.manaraa.com

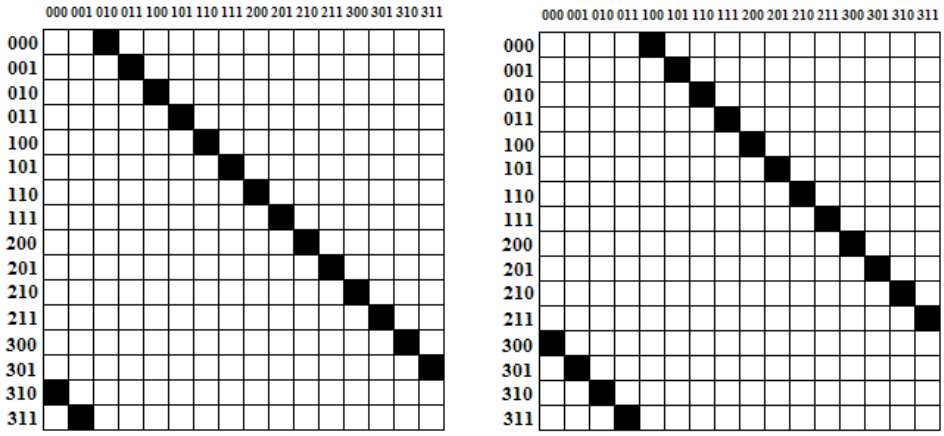


Figure 5.8: Traffic matrices for *Stride(2)* (left) and *Stride(4)* (right).

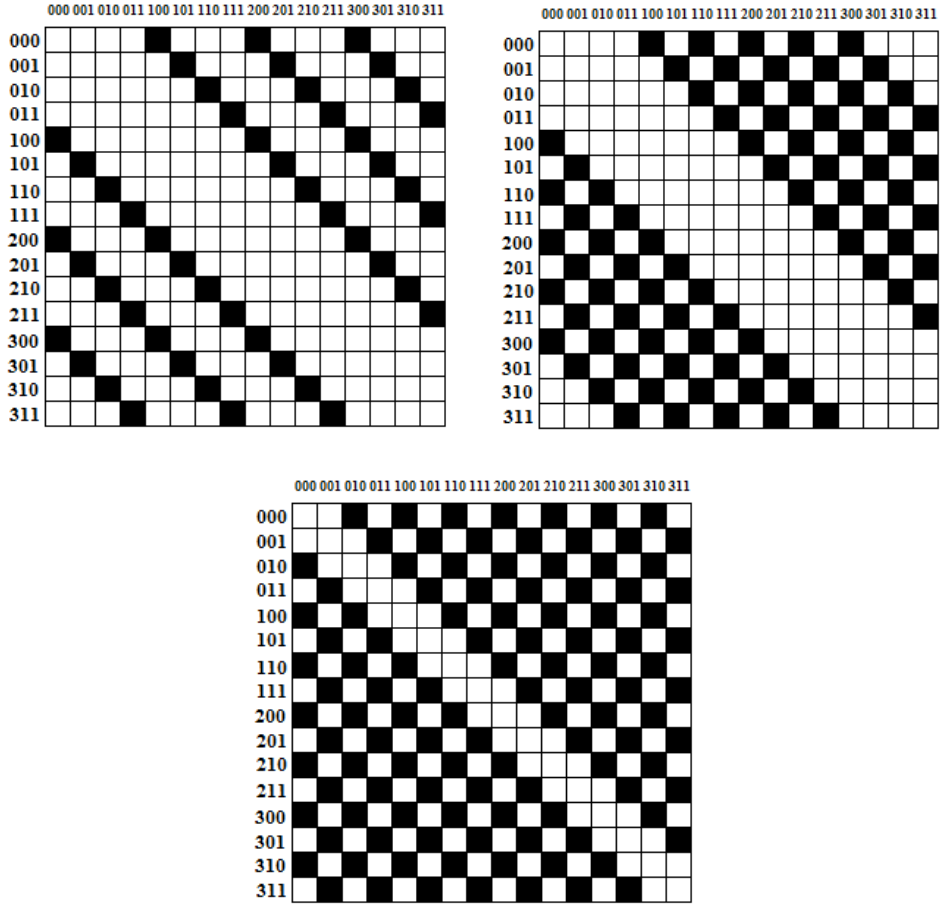


Figure 5.9: Traffic matrices for *3 Senders* (top left), *5 Senders* (top right) *and 7 Senders* (bottom).

www.manaraa.com

The traffic matrices for *Stride(2)* and *Stride(4)* on $FT(4,3)$ are illustrated in Figure 5.8. The amount of data to be sent from each node is 2 MB (2,000,000 bytes). We selected *Stride(4)* because it is an important traffic pattern for comparing routing schemes' performance. Specifically, in Theorem 5.5 we showed that *Stride(4)* is a worst-case TM for the $FT(4,3)$ routing schemes that we study, and that no TM induces higher baseload normalized maximum switch load in $FT(4,3)$ than *Stride(4)*.

Some additional traffic patterns that we investigated include:

- *3 Senders*: Each node receives data from 3 other nodes in the network.
- *5 Senders*: Each node receives data from 5 other nodes in the network.
- *7 Senders:* Each node receives data from 7 other nodes in the network.

Figure 5.9 shows the traffic matrices for *3, 5* and *7 Senders*. These traffic patterns represent the typical many-to-one barrier-synchronized workload in data centers, as discussed in Section 5.2.1. Specifically, each client requests 10 blocks of data, 262144 bytes $\times N$ each, striped over N servers (N = 3, 5, or 7, respectively). Clients request block $k+1$ only after successful receipt of all fragments in block k.

5.3.1.4 Tools for Gathering Statistics

We use FlowMonitor [97] to gather statistics for each flow. To further improve the accuracy of the flow statistics, we enabled packet capture during simulation, and then used a bash script we wrote to automatically process the .pcap files with tshark (part of Wireshark), retrieve statistics and store them in ASCII text files. After that, we ran a MATLAB script which takes these text files as input and automatically generates plots of various statistics for the schemes. These scripts are available in the Appendix. To speed up simulations, we used GNU parallel [98] to run multiple simulations at the same time.

www.manaraa.com

5.3.2 Sample Outputs from Simulation

Table 5.4 shows sample outputs from a run of the ns-3 simulation program for *Normal Routing*. Note that for readability, fewer flows were created in this sample run.

The first part of the output displays various user-controlled parameters, which can be specified from the command-line. "Routing scheme" specifies which routing scheme is being simulated. *QueueSize* sets the buffer size (in packets) in all switches. *SendBytes* is the total amount of data (in bytes) to be transferred from each sender. *SendPattern* specifies which communication pattern (*Random, Stride(2),* or *Stride(4)*) should be used.

The second part of the output shows various flow statistics, gathered by FlowMonitor [97]. These include, for each flow, the time when the last packet is received, total transferred bytes, average throughput, etc. A summary across all flows is then shown.

Table 5.4: Sample output from simulations for *Normal Routing*.

```
*****Parameters Selected*****
Random seed is 81, Routing scheme is Normal Routing
QueueSize is 30, SendBytes is 5000000, SendPattern is STRIDE(4)
****************************
Simulation running, please wait ...

Flow 1: timeLastRx: 1.53 s, txBytes: 5017 KB, rxBytes: 5017 KB, lostPackets: 0, throughput: 9435 KB/s
Flow 2: timeLastRx: 1.87 s, txBytes: 5069 KB, rxBytes: 5032 KB, lostPackets: 26, throughput: 5806 KB/s
Flow 3: timeLastRx: 1.53 s, txBytes: 5017 KB, rxBytes: 5017 KB, lostPackets: 0, throughput: 9435 KB/s
Flow 4: timeLastRx: 1.61 s, txBytes: 5018 KB, rxBytes: 5017 KB, lostPackets: 1, throughput: 8275 KB/s

Average flow completion time: 0.64 s, Average flow throughput: 8238 KB/s
Total bytes transferred: 20123 KB, Total lost packets: 27

Flow 1: (10.1.1.1/49153 --> 10.2.1.1/1), Flow 2: (10.1.1.3/49153 --> 10.2.1.3/2)
Flow 3: (10.1.3.1/49153 --> 10.2.3.1/3), Flow 4: (10.1.3.3/49153 --> 10.2.3.3/4)

Simulation took: 9 seconds
```

www.manaraa.com

The third part of the output displays the connection details of each flow, in the form of (source address/source port → destination address/destination port). Finally, the simulation duration (in wall clock time) is shown.

Another sample output, generated by a run of the ns-3 simulation for *Multipath Routing via Dynamic NIx-Vectors*, is shown in Table 5.5, with the same parameters.

Here, flow 1 and flow 2 are grouped; flow 3 and flow 4 are grouped, etc. The sum of transferred bytes for all flows in each group is *SendBytes*. However, the number of transferred bytes for each flow can vary, as they may encounter different path conditions.

Table 5.5: Sample output from simulations for *Multipath Routing via Dynamic NIx-Vectors*.

*****Parameters Selected*****
Random seed is 81, Routing scheme is **Multipath Routing via Dynamic NIx-Vectors**
QueueSize is 30, SendBytes is 5000000, SendPattern is STRIDE(4)
****
Simulation running, please wait ...

Flow 1: timeLastRx: 1.29 s, txBytes: 2527 KB, rxBytes: 2527 KB, lostPackets: 0, throughput: 8568 KB/s
Flow 2: timeLastRx: 1.3 s, txBytes: 2490 KB, rxBytes: 2490 KB, lostPackets: 0, throughput: 8424 KB/s
Flow 3: timeLastRx: 1.3 s, txBytes: 2527 KB, rxBytes: 2527 KB, lostPackets: 0, throughput: 8549 KB/s
Flow 4: timeLastRx: 1.3 s, txBytes: 2490 KB, rxBytes: 2490 KB, lostPackets: 0, throughput: 8423 KB/s
Flow 5: timeLastRx: 1.29 s, txBytes: 2527 KB, rxBytes: 2527 KB, lostPackets: 0, throughput: 8570 KB/s
Flow 6: timeLastRx: 1.3 s, txBytes: 2490 KB, rxBytes: 2490 KB, lostPackets: 0, throughput: 8412 KB/s
Flow 7: timeLastRx: 1.3 s, txBytes: 2527 KB, rxBytes: 2527 KB, lostPackets: 0, throughput: 8558 KB/s
Flow 8: timeLastRx: 1.3 s, txBytes: 2490 KB, rxBytes: 2490 KB, lostPackets: 0, throughput: 8423 KB/s

Average flow completion time: 0.3 s, Average flow throughput: 16982 KB/s
Total bytes transferred: 20070 KB, Total lost packets: 0

Flow 1: (10.1.1.1/49153 --> 10.2.1.1/1), Flow 2: (10.1.1.5/49153 --> 10.2.1.5/1)
Flow 3: (10.1.1.3/49153 --> 10.2.1.3/2), Flow 4: (10.1.1.7/49153 --> 10.2.1.7/2)
Flow 5: (10.1.3.1/49153 --> 10.2.3.1/3), Flow 6: (10.1.3.5/49153 --> 10.2.3.5/3)
Flow 7: (10.1.3.3/49153 --> 10.2.3.3/4), Flow 8: (10.1.3.7/49153 --> 10.2.3.7/4)

Simulation took: 9 seconds

www.manaraa.com

5.3.3 Addressing the Incast Problem

From our earlier analysis of switch load in Chapter 3, we concluded that compared to standard routing schemes, our proposed schemes do a better job of load balancing and avoiding switch buffer overfills which are the root causes of Incast.

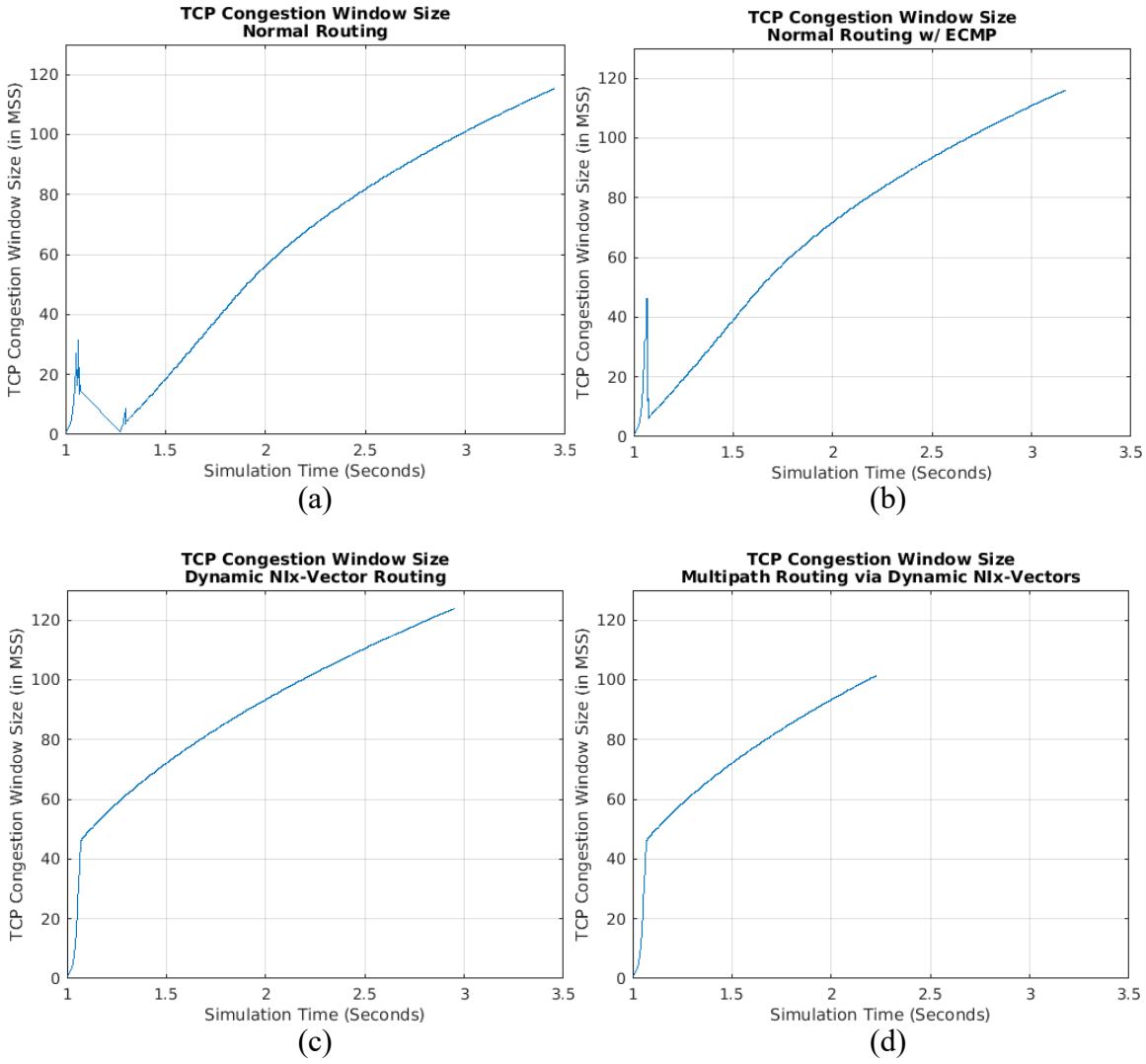


Figure 5.10: Illustration of the effectiveness of proposed schemes on alleviating Incast. (a) *Normal Routing,* (b) *Normal Routing w/ ECMP*, (c) *Dynamic NIx-Vector Routing*, (d) *Multipath Routing via Dynamic NIx-Vectors.*

www.manaraa.com

In our ns-3 simulations, by tracing the TCP congestion window size, we demonstrated the effectiveness of our proposed schemes on alleviating Incast, which agreed with our analysis. The results are plotted in Figure 5.10. We see that both *Normal Routing* and *Normal Routing with ECMP* suffer drastic reductions of TCP congestion window size shortly after data transfer begins. This corresponds to significant drops in perceived application-level throughput (goodput) and is due to Incast. For our proposed schemes, *Dynamic NIx-Vector Routing* and *Multipath Routing via Dynamic NIx-Vectors*, we do not see any TCP congestion window size reduction at similar time intervals. Therefore, we conclude that our schemes are indeed effective in alleviating Incast.

Additionally, we observe from Figure 5.10 that by avoiding Incast, our proposed routing schemes complete data transfer earlier than standard schemes.

5.3.4 Performance Study

To compare routing schemes' performance, we use the following metrics:

1. **Average flow completion time** [58, 99], defined as the average finish time across all flows. The time duration begins with TCP handshake, and ends when an ACK for the last data segment is received.
2. **Average flow throughput**, defined as the average throughput across all flows. The throughput for each flow is calculated by dividing the total bytes transferred by the time duration of the transfer.

The first metric, average flow completion time, is of particular interest for distributed data center applications, such as web search and analytics [100]. As shown in [101], network flows that fail to return their partial results on time can severely affect the responsiveness of real-time applications or degrade their results.

www.manaraa.com

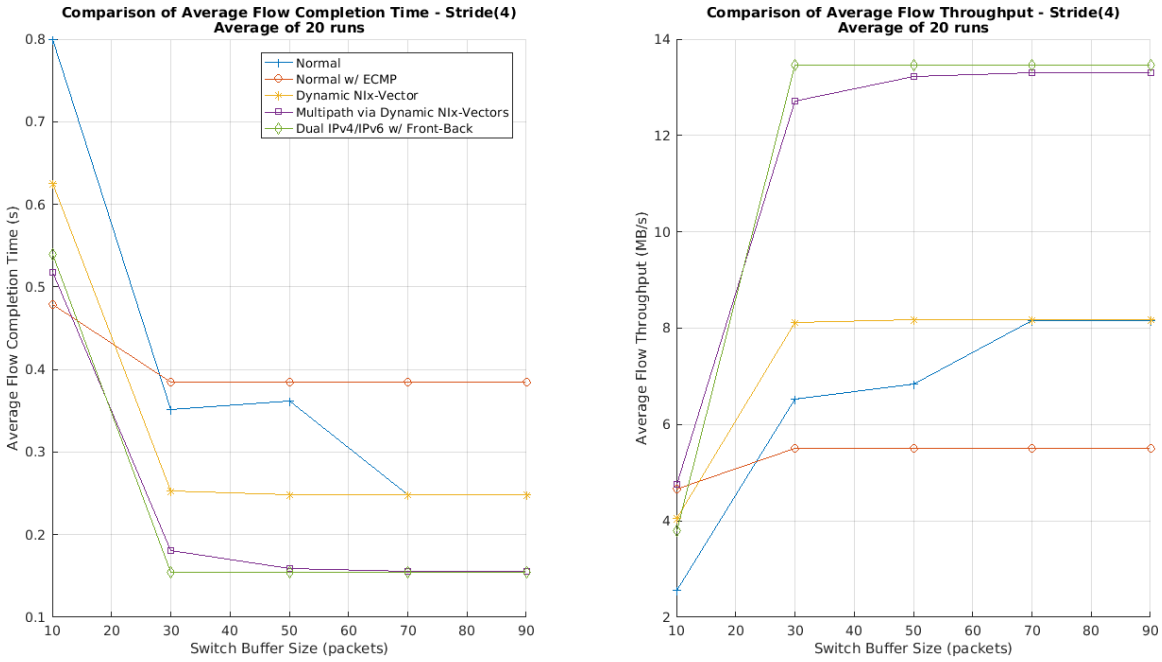


Figure 5.11: Comparison of average flow completion time and throughput for *Stride(4)*.

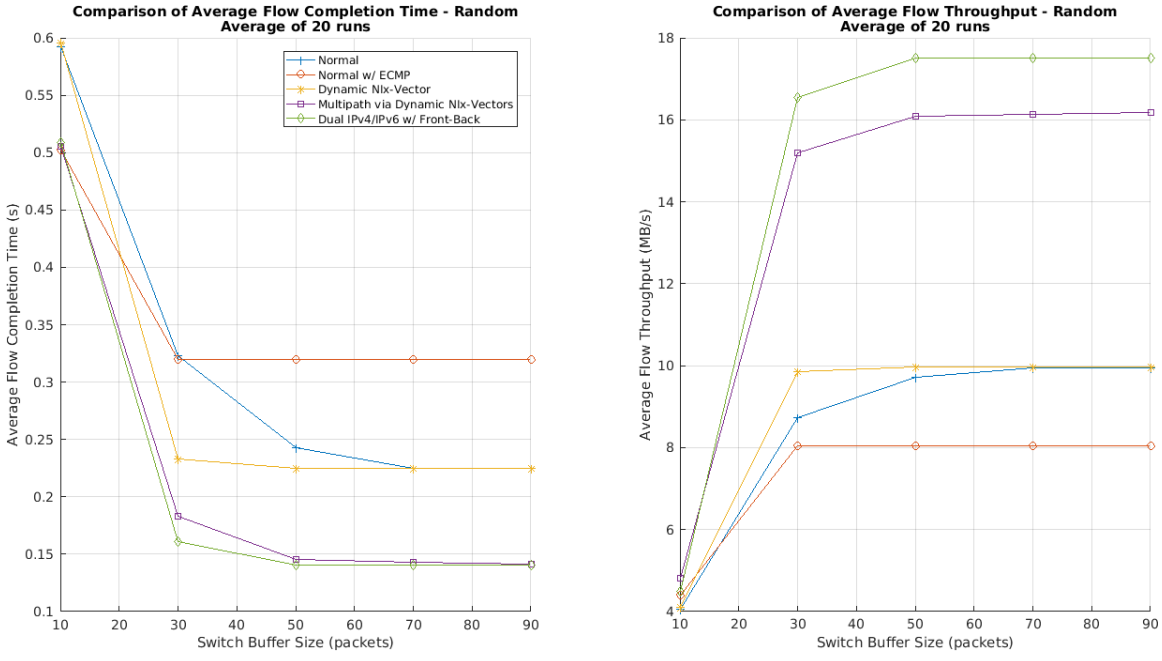


Figure 5.12: Comparison of average flow completion time and throughput for *Random*.

www.manaraa.com

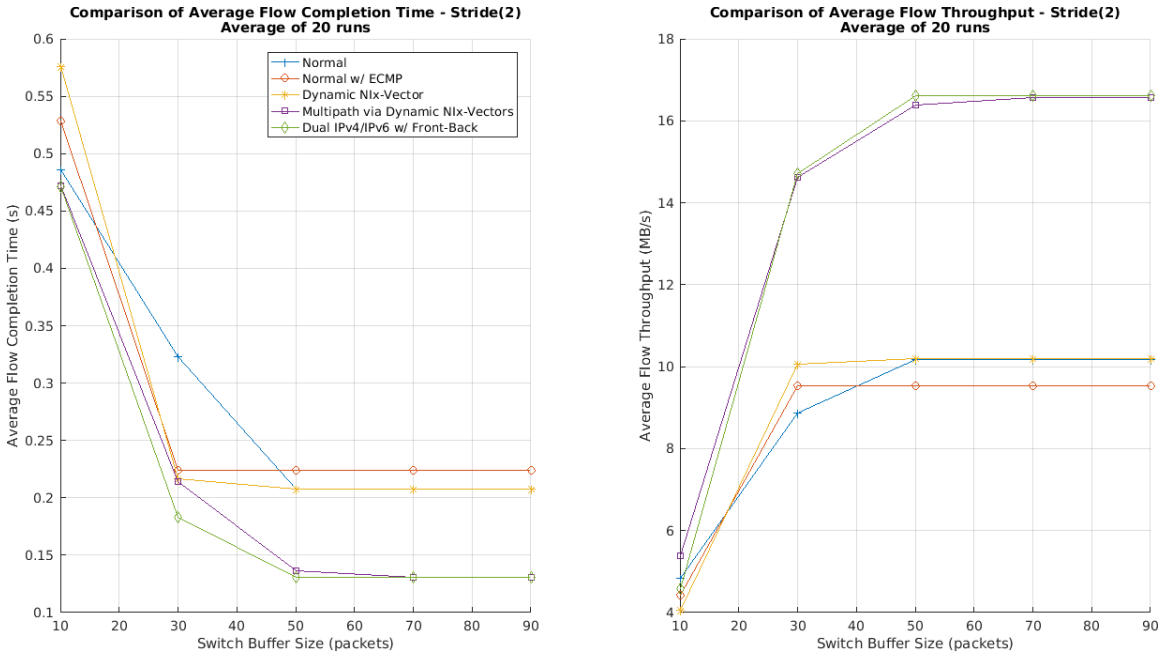


Figure 5.13: Comparison of average flow completion time and throughput for *Stride(2)*.

We first compare the performance of the proposed routing schemes under traffic patterns *Random, Stride(2)* and *Stride(4)*. The simulation results (averaged over 20 runs) are shown in Figures 5.11-5.13. The left subfigures show the average flow completion time, i.e. the average time duration for each node's data transfer to complete (lower is better); the right subfigures show average flow throughput (higher is better).

In Section 5.3.3, we showed, by tracing the TCP congestion window size, that our proposed schemes are effective at alleviating Incast. As observed in Figure 5.10, whenever Incast occurs, there is a steep decline in TCP throughput followed by a slow recovery. This will affect the overall average flow throughput, which can be seen from our results.

We now analyze the results in Figures 5.11-5.13. First, we compare *Dynamic NIx-Vector Routing* with *Normal Routing* and *Normal Routing with ECMP* (which uses the per-packet ECMP feature in ns-3). In the figures, these schemes' performances are plotted,

www.manaraa.com

respectively, as gold, blue and red lines. From the figures, we can see that *Dynamic NIx-Vector Routing* outperforms *Normal Routing* for all three communication patterns. In addition, *Stride(4)* results in the most performance difference between the two schemes. This is because the core switches experience heavy traffic in *Stride(4),* so any improvement in traffic balancing leads to more significant results.

The reason that *Normal Routing with ECMP* has bad performance is due to packet reordering induced by per-packet ECMP, which confuses TCP as packet loss and significantly degrades its performance. The scheme improves slightly in *Stride(2)*, because of the lower number of hops (4 versus 6) between source-destination pairs.

Next, we compare *Multipath Routing via Dynamic NIx-Vectors* and *Dual IPv4/IPv6 Routing with Front-Back* and *Normal Routing*, plotted as the magenta, green and blue lines, respectively. We observe that the first two, which are our proposed schemes, significantly outperform the last one in the simulation. Looking at the "average flow throughput" plot, we see that the most improvement occurs at approx. buffer size 50, with a percentage of around 80% under *Stride(4)*. The performance difference between *Multipath Routing via Dynamic NIx-Vectors* and *Dual IPv4/IPv6 Routing with Front-Back* is generally small. However, under the *Random* communication pattern, the latter has better performance.

In conclusion, from the figures we can see that our proposed scheme *Multipath Routing via Dynamic NIx-Vectors* outperforms both *Normal Routing* and *Normal Routing with ECMP* for all three communication patterns *Random, Stride(2) and Stride(4).*

From the "average flow completion time" plot, we see that the most improvement under *Stride(4)* is observed at approx. buffer size 50, reducing the completion time by 50% or more.

www.manaraa.com

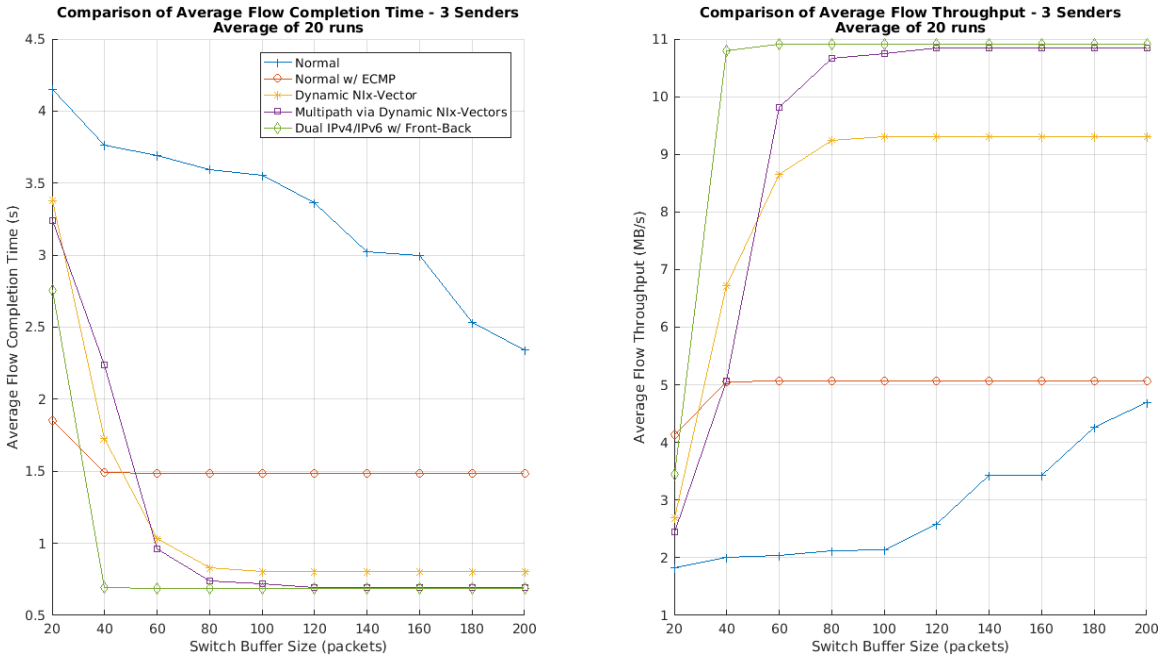


Figure 5.14: Comparison of average flow completion time and throughput for *3 Senders*.

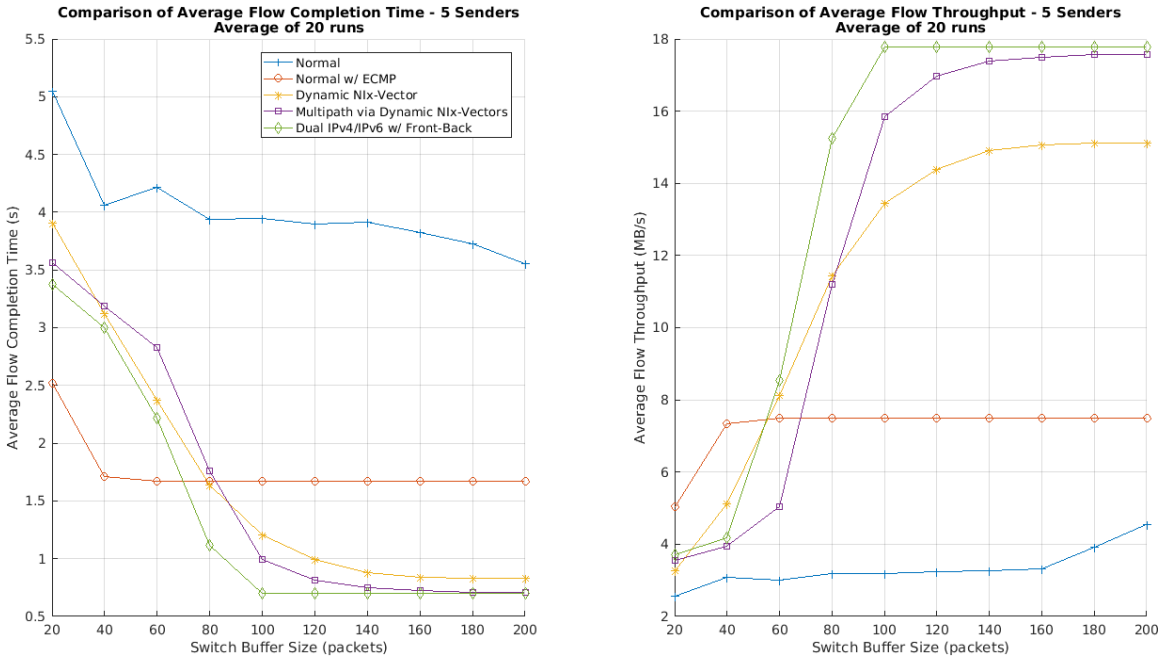


Figure 5.15: Comparison of average flow completion time and throughput for *5 Senders*.

www.manaraa.com

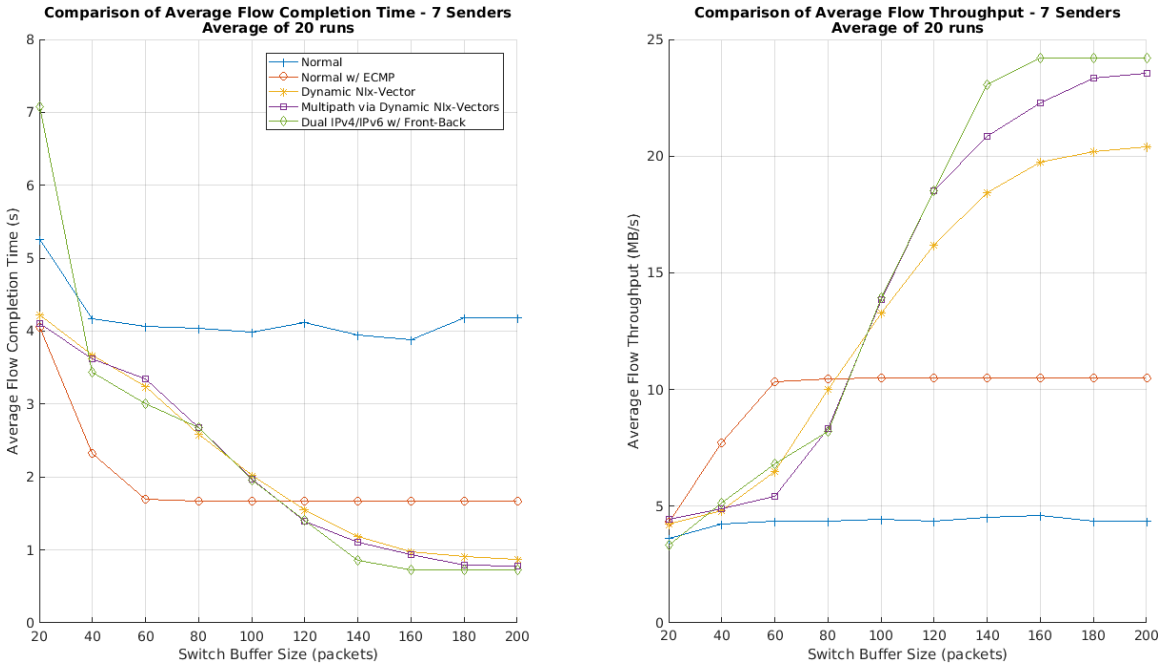


Figure 5.16: Comparison of average flow completion time and throughput for *7 Senders*.

Next, we compare performance of the proposed routing schemes under the additional traffic patterns *3 Senders, 5 Senders and 7 Senders,* as illustrated earlier in this section. The results are plotted in Figures 5.14-5.16. Again, the left subfigures show the average flow completion time, i.e. the average time duration for each node's data transfer to complete (lower is better); the right subfigures show average flow throughput (higher is better). We can see that our proposed schemes *Multipath Routing via Dynamic NIx-Vectors* and *Dual IPv4/IPv6 Routing with Front-Back* outperform *Normal Routing* for all three communication patterns.

Specifically, looking at the "average flow completion time" subfigures, we see that:

For *3 Senders*, the most improvement is observed at approx. buffer size 80, reducing the completion time by 75% or more.

www.manaraa.com

For 5 *Senders*, the most improvement is observed at approx. buffer size 140, reducing the completion time by 80% or more.

For 7 *Senders*, the most improvement is observed at approx. buffer size 180, reducing the completion time by 80% or more.

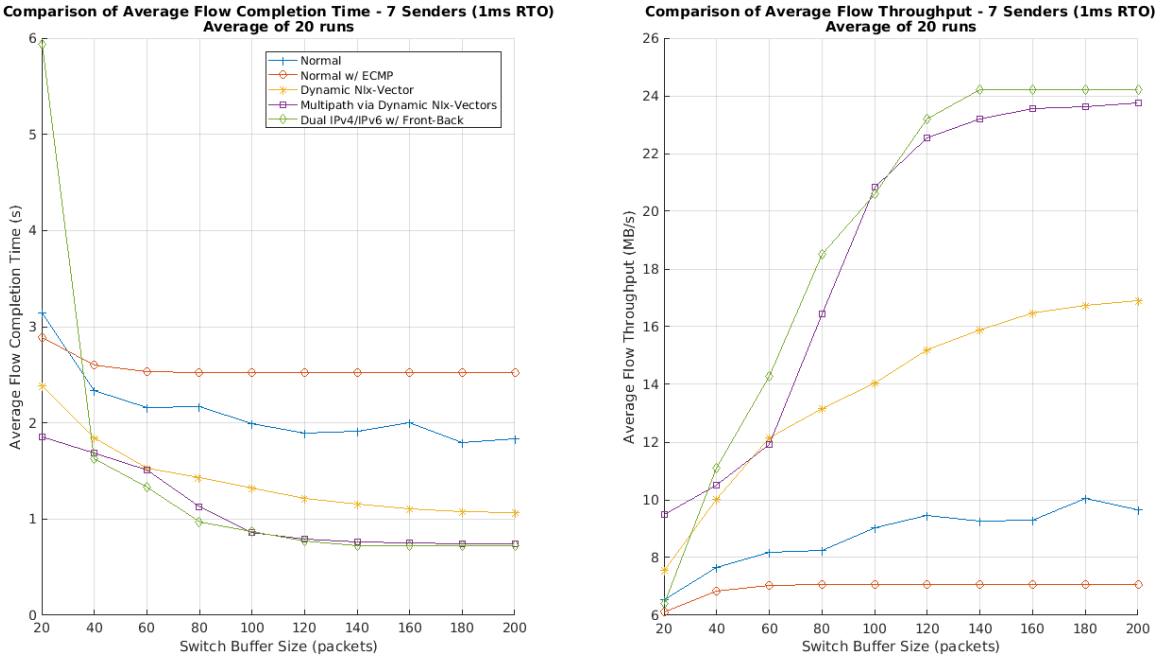


Figure 5.17: Comparison of average flow completion time and throughput for *7 Senders*, with 1ms RTO instead of the normal 200ms RTO.

Finally, we investigate our proposed schemes' performance in conjunction with other researchers' method of reducing the TCP re-transmission timeout (RTO) [4]. Figure 5.17 plots the schemes' performance under traffic pattern *7 Senders,* with 1ms RTO instead of the normal 200ms RTO. We observe that, under the reduced TCP RTO of 1ms, our proposed scheme *Multipath Routing via Dynamic NIx-Vectors* still outperforms *Normal Routing* by both reducing average flow completion time and improving throughput. More

www.manaraa.com

importantly, this means our approach can be "complementary" to the method of reducing TCP RTO, and both methods could be applied simultaneously to further improve performance.

5.4 Conclusions

In this chapter, we first defined and analyzed the oblivious performance ratios (OPRs) under the proposed routing schemes. We then explored typical Incast traffic patterns in data center networks, i.e. the "barrier-synchronized request workload", and summarized the conditions favorable to the onset of TCP Incast. Next, we identified important traffic patterns using our new method of traffic matrix decomposition, and illustrated different routing schemes' performance ratios.

Next, we assessed the potential benefits of our proposed routing schemes through ns-3 simulations on fat-trees under a variety of communication patterns. By tracing the TCP congestion window size, we demonstrated the effectiveness of our schemes on alleviating TCP Incast. In the detailed performance study that followed, we focused on two important metrics, namely the average flow completion time and average flow throughput, to help compare routing schemes' performance. Results indicate that the proposed schemes outperform standard routing schemes over a variety of experimental conditions.

www.manaraa.com

CHAPTER 6

CONCLUSIONS AND POSSIBLE EXTENSIONS

6.1 Summary of Research

In this dissertation, we used network regularity to overcome TCP Incast through multipath routing. First, we developed new oblivious, multi-path, routing schemes for fat-tree networks, which provide a low overhead means to reduce the likelihood of Incast. Next, we established that these schemes delay the onset of Incast, by deriving tight worst-case loading bounds for fat-tree switches and proving that our schemes lower these bounds. We then investigated a novel "front-back" approach to avoid multipath reordering, proved its optimality for two paths, and extended the algorithm to N paths ($N > 2$).

Our performance analysis began with an investigation of different routing schemes' oblivious performance ratios (OPRs). We then explored typical Incast traffic patterns in data center networks, and described a novel method for traffic matrix decomposition to help visually illustrate and classify traffic patterns. Finally, we assessed the potential benefits of our schemes through ns-3 simulations on fat-trees under a variety of traffic conditions. Results indicate that over a variety of experimental conditions, the proposed schemes reduced the incidence of TCP Incast compared to standard routing schemes.

6.2 Possible Extensions

Alternative NIx-like approach by enabling source routing in data centers

Source routing [102] allows the sender of a packet to specify the route the packet takes through the network, either partially or completely. There are IPv4 header options

www.manaraa.com

and IPv6 routing header extensions that can be used for source routing. Unfortunately, these options have been disabled on most Internet switches due to security concerns. For instance, an attacker can spoof his/her IP address to impersonate another user, while still receiving responses by specifying his/her real IP address as one of the hops that the packets must traverse. In recent years, source routing has seen increased application in routing protocols. In [34], the authors proposed BCube Source Routing (BSR), specific to the BCube topology. Notable examples for wireless networks include *Dynamic Source Routing (DSR)* [103] and *Multipath Dynamic Source Routing (MP-DSR)* [104].

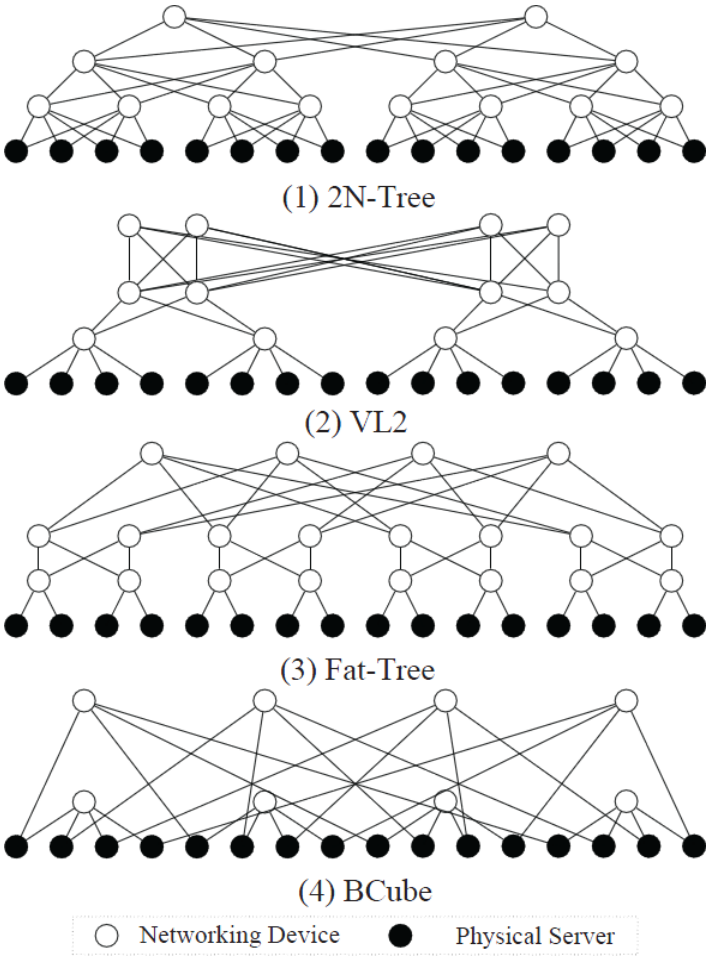


Figure 6.1: Some state-of-the-art data center topologies from [105].

www.manaraa.com

We argue that since an entire data center network is often managed and maintained by a single company or entity, the security issues stated above become less of a concern. Hence, enabling the source routing options for all switches in a data center would give us great flexibility in routing. This would be a viable alternative to the NIx-Vector routing schemes. Given that source routing is enabled, the source node can easily control which path each packet will traverse, by simply adding the appropriate options in packet headers. Therefore, an interesting extension could be to investigate the possibilities of applying source routing to data center networks.

Larger-scale fat-trees and irregular data center topologies

Our proposed schemes could easily be extended to larger-scale topologies, such as fat-trees with 8- or 16-port switches. Figure 6.1 illustrates some of these state-of-the-art topologies, including the VL2 and BCube topologies. *Multipath Routing via Dynamic NIx-Vectors* could be directly applied to these topologies, because the modified BFS path selection process does not depend on the topology used. For *Dual IPv4/IPv6 Routing with Front-Back*, when applied to fat-trees with 8- or 16-port switches, one could re-use the static routing setup in Figure 2.2, scaled to fit the larger topology.

A possible extension of our work is to investigate the performance of our proposed schemes in large topologies. Specifically, we could implement and test our *N-path "Front-Back" Algorithm,* because there can be more shortest paths available between each source-destination pair. In addition, we could see how algorithm performance will be affected in other topologies where the multiple paths have different delays.

www.manaraa.com

Investigate other path selection algorithms for BFS

Another possible extension is to investigate other path selection algorithms in the breadth-first search (BFS) phase of *Dynamic NIx-Vector Routing*, for adapting to different scenarios as appropriate. For different groups of communication patterns in the data center, we could use appropriate path selection policies to improve traffic balance. The first step would be further modifying the BFS algorithm in ns-3's NIx-Vector routing code, so that we can store all the discovered shortest paths for retrieval; alternatively, we can generate NIx-Vectors for all the multiple paths, and store them for retrieval later.

A possible metric for ranking different path selection schemes' performance would be to count the number of simultaneous connections before Incast collapse. The higher the number, the more effectively that scheme works to alleviate Incast.

Experiment on real network testbeds

Besides showing advantages in theoretical calculation and simulations, an interesting extension would be to validate our proposed schemes in real-life scenarios. For example, we can run experiments using the Open Network Laboratory (ONL) [106] or Emulab [107] and gather statistics on how well our proposed schemes perform, under different circumstances. ONL and Emulab are both open to public researchers. We could create different data center topologies over these testbeds to validate our schemes and observe their performance.

Implementation of the proposed approaches

To date our proposed approaches to mitigating the Incast problem show good promise. We conclude here with a brief discussion of their implementation issues.

www.manaraa.com

NIx-Vector/Source routing requires the storage of routing path information in packet headers. For source routing, existing IPv4 Options such as “strict source and record route” (SSRR) and “loose source and record route” (LSRR) could be used. For NIx-Vector routing, in the complete version of the NIx-Vector paper by Riley et al. [68], the authors proposed several new IPv4 Options for this purpose, such as the “Record-NIx-Vector Option” and “Use-NIx-Vector Option”.

IPv6 defines a set of additional headers called “extension headers” to achieve similar functionalities as IPv4 Options. Among these is the routing header, which provides support for source routing and IPv6 mobility. This header allows a source to list one or more intermediate nodes that a packet must “visit” on its way to the destination, so it can be utilized by NIx-Vector/Source routing.

The issues are somewhat different for *Virtual Table-Based Routing* (specifically, *Dual IPv4/IPv6 Routing)*. As mentioned earlier in Chapter 2, the implementation is straightforward when data center switches are dual-stacked, i.e. with both IPv4 and IPv6 enabled. If this is not the case, a more complex alternative would be to install multiple IP stacks on different interfaces of a switch and then set their routing tables so that packets received on different interfaces are routed differently.

The *Front-Back Algorithm* stands on its own and is straightforward to implement in conjunction with any scheme for which the existing multiple paths between each source-destination pair are used simultaneously.

www.manaraa.com

APPENDIX

SIMULATION SOURCE CODE AND SCRIPTS

The following source codes and scripts are available in the **srccode_scripts** directory:

File Name	Description
dnvr.cc	Complete ns-3 simulation source code for Random, Stride(2) and Stride(4)
dnvr2.cc	Complete ns-3 simulation source code for 3 Senders, 5 Senders and 7 Senders
runsim.sh	Bash script to run simulations under different routing schemes, for Random, Stride(2) and Stride(4)
runsim2.sh	Same as above, but for 3 Senders, 5 Senders and 7 Senders
runbatch.sh	Bash script using GNU parallel to run multiple simulations at the same time, for Random, Stride(2) and Stride(4)
runbatch2.sh	Same as above, but for 3 Senders, 5 Senders and 7 Senders
rproc.sh	Bash script to process simulation results and generate statistics, for Random, Stride(2) and Stride(4)
rproc2.sh	Same as above, but for 3 Senders, 5 Senders and 7 Senders
netgoodput.m	MATLAB script to estimate the best-case improvement (Fig. 1.6)
genplots.m	MATLAB script to make plots from simulation output, for Random, Stride(2) and Stride(4) (Figs. 5.11-5.13)
genplots2.m	Same as above, but for 3 Senders, 5 Senders and 7 Senders (Figs. 5.14-5.17)
cwndplot.m	MATLAB script to plot traced TCP congestion window sizes (Fig. 5.10)
avgcalc.m	MATLAB script to calculate average values over 20 simulation runs
drcalc.mw	Maple program to plot transfer finish time for 2 paths (Fig. 4.3)
ncs_2paths.mw	Maple program to plot packet disorder for 2 paths (Figs. 4.3-4.4)
ncs_4paths.mw	Maple program to plot packet disorder for 4 paths (Fig. 4.6)
oprcalc.mw	Maple program to plot the oblivious performance ratios (Fig. 5.1)
ft43load.mw	Maple program to analyze the oblivious performance ratios (Table 5.2)
ns3bfsmod.diff	Diff file showing our changes to ns-3 NIx-Vector BFS code to achieve RLB

www.manaraa.com

REFERENCES

[1] Dean, J. and Ghemawat, S., *MapReduce: simplified data processing on large clusters.* Communications of the ACM, 2008. **51**(1): p. 107-113.

[2] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. *The Hadoop Distributed File System*. in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 2010.

[3] Chen, Y., Griffit, R., Zats, D., and Katz, R.H., *Understanding tcp incast and its implications for big data workloads.* University of California at Berkeley, Tech. Rep, 2012.

[4] Vasudevan, V., Phanishayee, A., Shah, H., Krevat, E., Andersen, D.G., Ganger, G.R., Gibson, G.A., and Mueller, B. *Safe and effective fine-grained TCP retransmissions for datacenter communication*. in *ACM SIGCOMM computer communication review*. 2009. ACM.

[5] *Parallel Data Lab Project: INCAST*. Available from: http://www.pdl.cmu.edu/Incast/.

[6] Phanishayee, A., Krevat, E., Vasudevan, V., Andersen, D.G., Ganger, G.R., Gibson, G.A., and Seshan, S. *Measurement and Analysis of TCP Throughput Collapse in Cluster-based Storage Systems*. in *FAST*. 2008.

[7] Chen, Y., Griffith, R., Liu, J., Katz, R.H., and Joseph, A.D. *Understanding TCP incast throughput collapse in datacenter networks*. in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. 2009. ACM.

[8] Nagle, D., Serenyi, D., and Matthews, A. *The panasas activescale storage cluster: Delivering scalable high bandwidth storage*. in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. 2004. IEEE Computer Society.

[9] Ghemawat, S., Gobioff, H., and Leung, S.-T. *The Google file system*. in *ACM SIGOPS operating systems review*. 2003. ACM.

[10] Alexander, F.J., Hoisie, A., and Szalay, A., *Big data [Guest editorial]*. Computing in Science & Engineering, 2011. **6**(13): p. 10-13.

[11] Cheng, Y., Qin, C., and Rusu, F. *GLADE: big data analytics made easy*. in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 2012. ACM.

[12] Huai, Y., Lee, R., Zhang, S., Xia, C.H., and Zhang, X. *DOT: a matrix model for analyzing, optimizing and deploying software for big data analytics in distributed systems*. in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 2011. ACM.

[13] Bajda-Pawlikowski, K., Abadi, D.J., Silberschatz, A., and Paulson, E. *Efficient processing of data warehousing queries in a split execution environment*. in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 2011. ACM.

www.manaraa.com

[14] Zheng, H. and Qiao, C. *An effective approach to preventing TCP incast throughput collapse for data center networks*. in *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*. 2011. IEEE.

[15] Rajanna, V.S., Shah, S., Jahagirdar, A., Lemoine, C., and Gopalan, K. *Xco: Explicit coordination to prevent network fabric congestion in cloud computing cluster platforms*. in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. 2010. ACM.

[16] Krevat, E., Vasudevan, V., Phanishayee, A., Andersen, D.G., Ganger, G.R., Gibson, G.A., and Seshan, S. *On application-level approaches to avoiding TCP throughput collapse in cluster-based storage systems*. in *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07*. 2007. ACM.

[17] Zhang, P., Wang, H., and Cheng, S. *Shrinking mtu to mitigate tcp incast throughput collapse in data center networks*. in *Communications and Mobile Computing (CMC), 2011 Third International Conference on*. 2011. IEEE.

[18] Kulkarni, S. and Agrawal, P. *A probabilistic approach to address TCP incast in data center networks*. in *2011 31st International Conference on Distributed Computing Systems Workshops*. 2011. IEEE.

[19] Adesanmi, A. and Mhamdi, L. *Controlling TCP Incast congestion in data centre networks*. in *2015 IEEE International Conference on Communication Workshop (ICCW)*. 2015.

[20] Yongmao, R., Jun, L., Guodong, W., Lingling, L., and Shanshan, S. *SA-TCP: A novel approach to mitigate TCP Incast in data center networks*. in *2015 International Conference on Computing and Network Communications (CoCoNet)*. 2015.

[21] Wu, H., Feng, Z., Guo, C., and Zhang, Y., *ICTCP: Incast congestion control for TCP in data-center networks.* IEEE/ACM Transactions on Networking (TON), 2013. **21**(2): p. 345-358.

[22] Ramakrishnan, K., Floyd, S., and Black, D. *The addition of explicit congestion notification (ECN) to IP*. RFC 3168, , DOI 10.17487/RFC3168. 2001; Available from: http://www.rfc-editor.org/info/rfc3168.

[23] Zhang, Y. and Ansari, N. *On mitigating TCP incast in data center networks*. in *INFOCOM, 2011 Proceedings IEEE*. 2011. IEEE.

[24] Wilson, C., Ballani, H., Karagiannis, T., and Rowtron, A. *Better never than late: Meeting deadlines in datacenter networks*. in *ACM SIGCOMM Computer Communication Review*. 2011. ACM.

[25] Alizadeh, M., Greenberg, A., Maltz, D.A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. *Data center tcp (dctcp)*. in *ACM SIGCOMM computer communication review*. 2010. ACM.

[26] Tseng, H.W., Yang, T.T., and Peng, Y.H. *An urgency and congestion control scheme for larger-scale TCP incast problem in data center*. in *2015 IEEE Symposium on Computers and Communication (ISCC)*. 2015.

www.manaraa.com

[27] Hwang, J., Yoo, J., and Choi, N., *Deadline and Incast Aware TCP for cloud data center networks.* Computer Networks, 2014. **68**: p. 20-34.

[28] Pfister, G.F., *An introduction to the infiniband architecture.* High Performance Mass Storage and Parallel I/O, 2001. **42**: p. 617-632.

[29] Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N., and Su, W.-K., *Myrinet: A gigabit-per-second local area network.* IEEE micro, 1995. **15**(1): p. 29-36.

[30] Yu, Y., Aung, K.M.M., Tong, E.K.K., and Foh, C.H. *Dynamic load balancing multipathing in Data Center Ethernet*. in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. 2010. IEEE.

[31] Al-Fares, M., Loukissas, A., and Vahdat, A. *A scalable, commodity data center network architecture*. in *ACM SIGCOMM Computer Communication Review*. 2008. ACM.

[32] Greenberg, A., Hamilton, J.R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D.A., Patel, P., and Sengupta, S., *VL2: a scalable and flexible data center network.* Communications of the ACM, 2011. **54**(3): p. 95-104.

[33] Guo, C., Wu, H., Tan, K., Shi, L., Zhang, Y., and Lu, S. *Dcell: a scalable and fault-tolerant network structure for data centers*. in *ACM SIGCOMM Computer Communication Review*. 2008. ACM.

[34] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S., *BCube: a high performance, server-centric network architecture for modular data centers.* ACM SIGCOMM Computer Communication Review, 2009. **39**(4): p. 63-74.

[35] Greenberg, A., Lahiri, P., Maltz, D.A., Patel, P., and Sengupta, S. *Towards a next generation data center architecture: scalability and commoditization*. in *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*. 2008. ACM.

[36] Abu-Libdeh, H., Costa, P., Rowstron, A., O'Shea, G., and Donnelly, A., *Symbiotic routing in future data centers.* ACM SIGCOMM Computer Communication Review, 2010. **40**(4): p. 51-62.

[37] Leiserson, C.E., *Fat-trees: universal networks for hardware-efficient supercomputing.* IEEE transactions on Computers, 1985. **100**(10): p. 892-901.

[38] Clos, C., *A Study of Non-Blocking Switching Networks.* Bell System Technical Journal, 1953. **32**(2): p. 406-424.

[39] Lin, X.-Y., Chung, Y.-C., and Huang, T.-Y. *A multiple LID routing scheme for fat-tree-based InfiniBand networks*. in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. 2004. IEEE.

[40] Valiant, L.G. and Brebner, G.J. *Universal schemes for parallel communication*. in *Proceedings of the thirteenth annual ACM symposium on Theory of computing*. 1981. ACM.

www.manaraa.com

[41] Zats, D., Das, T., Mohan, P., Borthakur, D., and Katz, R., *DeTail: reducing the flow completion time tail in datacenter networks.* ACM SIGCOMM Computer Communication Review, 2012. **42**(4): p. 139-150.

[42] Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. *Hedera: Dynamic Flow Scheduling for Data Center Networks*. in *NSDI*. 2010.

[43] Schlansker, M., Turner, Y., Tourrilhes, J., and Karp, A. *Ensemble routing for datacenter networks*. in *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. 2010. ACM.

[44] Wu, W., Turner, Y., and Schlansker, M. *Routing optimization for ensemble routing*. in *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*. 2011. IEEE Computer Society.

[45] Miura, S.i., Boku, T., Okamoto, T., and Hanawa, T. *A dynamic routing control system for high-performance PC cluster with multi-path Ethernet connection*. in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. 2008. IEEE.

[46] Mudigonda, J., Yalagandula, P., Al-Fares, M., and Mogul, J.C., *SPAIN: design and algorithms for constructing large data-center ethernets from commodity switches*. 2009, Tech. Rep. HPL-2009-241, HP Labs.

[47] Raiciu, C., Pluntke, C., Barre, S., Greenhalgh, A., Wischik, D., and Handley, M. *Data center networking with multipath TCP*. in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. 2010. ACM.

[48] Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and Handley, M. *Improving datacenter performance and robustness with multipath TCP*. in *ACM SIGCOMM Computer Communication Review*. 2011. ACM.

[49] Ford, A., Raiciu, C., Handley, M., and Bonaventure, O. *TCP extensions for multipath operation with multiple addresses*. RFC 6824, DOI 10.17487/RFC6824. 2013; Available from: http://www.rfc-editor.org/info/rfc6824.

[50] Li, L., Hu, N., Liu, K., Fu, B., Chen, M., and Zhang, L., *AMTCP: an adaptive multi-path transmission control protocol*, in *Proceedings of the 12th ACM International Conference on Computing Frontiers*. 2015, ACM: Ischia, Italy. p. 1-8.

[51] Kheirkhah, M., Wakeman, I., and Parisis, G. *MMPTCP: A multipath transport protocol for data centers*. in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 2016.

[52] Wang, W., Zhou, L., and Sun, Y. *Improving Multipath TCP for Latency Sensitive Flows in the Cloud*. in *2016 5th IEEE International Conference on Cloud Networking (Cloudnet)*. 2016.

[53] Li, M., Lukyanenko, A., Ou, Z., Ylä-Jääski, A., Tarkoma, S., Coudron, M., and Secci, S., *Multipath Transmission for the Internet: A Survey.* IEEE Communications Surveys & Tutorials, 2016. **18**(4): p. 2887-2925.

www.manaraa.com

[54] Habib, S., Qadir, J., Ali, A., Habib, D., Li, M., and Sathiaseelan, A., *The past, present, and future of transport-layer multipath.* Journal of Network and Computer Applications, 2016. **75**: p. 236-258.

[55] Li, M., Lukyanenko, A., Tarkoma, S., and Ylä-Jääski, A., *MPTCP incast in data center networks.* China Communications, 2014. **11**(4): p. 25-37.

[56] Alizadeh, M., Edsall, T., Dharmapurikar, S., Vaidyanathan, R., Chu, K., Fingerhut, A., Lam, V.T., Matus, F., Pan, R., Yadav, N., and Varghese, G., *CONGA: distributed congestion-aware load balancing for datacenters.* SIGCOMM Comput. Commun. Rev., 2014. **44**(4): p. 503-514.

[57] Cao, Y., Xu, M., Fu, X., and Dong, E., *Explicit multipath congestion control for data center networks*, in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. 2013, ACM: Santa Barbara, California, USA. p. 73-84.

[58] Chen, G., Lu, Y., Meng, Y., Li, B., Tan, K., Pei, D., Cheng, P., Luo, L., Xiong, Y., Wang, X., and Zhao, Y., *Fast and cautious: leveraging multi-path diversity for transport loss recovery in data centers*, in *Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference*. 2016, USENIX Association: Denver, CO, USA. p. 29-42.

[59] Lappetelainen, A., *Equal Cost Multipath Routing in IP Networks.* Faculty of Electronics, Communications and Automation, 2011.

[60] Curtis, A.R., Kim, W., and Yalagandula, P. *Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection*. in *INFOCOM, 2011 Proceedings IEEE*. 2011. IEEE.

[61] Estan, C. and Varghese, G., *New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice.* ACM Transactions on Computer Systems (TOCS), 2003. **21**(3): p. 270-313.

[62] Wu, X. and Yang, X. *Dard: Distributed adaptive routing for datacenter networks*. in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. 2012. IEEE.

[63] Curtis, A.R., Mogul, J.C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S., *DevoFlow: scaling flow management for high-performance networks.* ACM SIGCOMM Computer Communication Review, 2011. **41**(4): p. 254-265.

[64] Raiciu, C., Handley, M., and Wischik, D. *Coupled congestion control for multipath transport protocols*. RFC 6356, DOI 10.17487/RFC6356. 2011; Available from: http://www.rfc-editor.org/info/rfc6356.

[65] Farrington, N., *Multipath TCP under Massive Packet Reordering*. 2009, University of California at San Diego.

[66] Leung, K.-C. and Li, V.O. *Generalized load sharing for packet-switching networks*. in *Network Protocols, 2000. Proceedings. 2000 International Conference on*. 2000. IEEE.

www.manaraa.com

[67] Wischik, D., Handley, M., and Braun, M.B., *The resource pooling principle.* ACM SIGCOMM Computer Communication Review, 2008. **38**(5): p. 47-52.

[68] Riley, G.F., Ammar, M.H., and Zegura, E.W. *Efficient routing using nix-vectors*. in *High Performance Switching and Routing, 2001 IEEE Workshop on*. 2001. IEEE.

[69] *ICS 161 Lecture notes by D. Eppstein*. Available from: http://www.ics.uci.edu/~eppstein/161/960215.html.

[70] Ohring, S.R., Ibel, M., Das, S.K., and Kumar, M.J. *On generalized fat trees*. in *Parallel Processing Symposium, 1995. Proceedings., 9th International*. 1995. IEEE.

[71] Leiserson, C.E., Abuhamdeh, Z.S., Douglas, D.C., Feynman, C.R., Ganmukhi, M.N., Hill, J.V., Hillis, D., Kuszmaul, B.C., Pierre, M.A.S., Wells, D.S., Wong, M.C., Yang, S.-W., and Zak, R., *The network architecture of the Connection Machine CM-5 (extended abstract)*, in *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*. 1992, ACM: San Diego, California, USA. p. 272-285.

[72] Gomez, C., Gilabert, F., Gomez, M.E., López, P., and Duato, J. *Deterministic versus adaptive routing in fat-trees*. in *2007 IEEE International Parallel and Distributed Processing Symposium*. 2007. IEEE.

[73] Johnson, G., Kerbyson, D.J., and Lang, M. *Optimization of infiniband for scientific applications*. in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. 2008. IEEE.

[74] Zahavi, E., Johnson, G., Kerbyson, D.J., and Lang, M., *Optimized InfiniBandTM fat-tree routing for shift all-to-all communication patterns.* Concurrency and Computation: Practice and Experience, 2010. **22**(2): p. 217-231.

[75] Rodriguez, G., Minkenberg, C., Beivide, R., Luijten, R.P., Labarta, J., and Valero, M. *Oblivious routing schemes in extended generalized fat tree networks*. in *2009 IEEE International Conference on Cluster Computing and Workshops*. 2009. IEEE.

[76] Dally, W.J. and Towles, B.P., *Oblivious Routing*, in *Principles and Practices of Interconnection Networks*. 2004, Elsevier. p. 180.

[77] Kulkarni, S.B., *Incast-free TCP for Data Center Networks*. 2012, Auburn University.

[78] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J., *Modeling TCP throughput: a simple model and its empirical validation*, in *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*. 1998, ACM: Vancouver, British Columbia, Canada. p. 303-314.

[79] Villamizar, C. and Song, C., *High performance TCP in ANSNET.* ACM SIGCOMM Computer Communication Review, 1994. **24**(5): p. 45-60.

[80] Morris, R. *TCP behavior with many flows*. in *Network Protocols, 1997. Proceedings., 1997 International Conference on*. 1997. IEEE.

www.manaraa.com

[81] Morris, R. *Scalable TCP congestion control*. in *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. 2000. IEEE.

[82] Appenzeller, G., Keslassy, I., and McKeown, N., *Sizing router buffers.* SIGCOMM Comput. Commun. Rev., 2004. **34**(4): p. 281-292.

[83] Dhamdhere, A. and Dovrolis, C., *Open issues in router buffer sizing.* ACM SIGCOMM Computer Communication Review, 2006. **36**(1): p. 87-92.

[84] Yuan, X., Nienaber, W., Duan, Z., and Melhem, R., *Oblivious routing in fat-tree based system area networks with uncertain traffic demands.* IEEE/ACM Transactions on Networking (TON), 2009. **17**(5): p. 1439-1452.

[85] Hsieh, H.-Y. and Sivakumar, R., *A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts.* Wireless Networks, 2005. **11**(1-2): p. 99-114.

[86] Iyengar, J.R., Amer, P.D., and Stewart, R. *Receive buffer blocking in concurrent multipath transfer*. in *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.* 2005. IEEE.

[87] Kandula, S., Katabi, D., Sinha, S., and Berger, A., *Dynamic load balancing without packet reordering.* ACM SIGCOMM Computer Communication Review, 2007. **37**(2): p. 51-62.

[88] Baldini, A., De Carli, L., and Risso, F. *Increasing performances of TCP data transfers through multiple parallel connections*. in *Computers and Communications, 2009. ISCC 2009. IEEE Symposium on*. 2009. IEEE.

[89] Han, H., Shakkottai, S., Hollot, C.V., Srikant, R., and Towsley, D., *Multi-path tcp: a joint congestion control and routing scheme to exploit path diversity in the internet.* IEEE/ACM Transactions on Networking (TON), 2006. **14**(6): p. 1260-1271.

[90] Rodriguez, P. and Biersack, E.W., *Dynamic parallel access to replicated content in the Internet.* IEEE/ACM Transactions on Networking (TON), 2002. **10**(4): p. 455-465.

[91] Hasegawa, Y., Yamaguchi, I., Hama, T., Shimonishi, H., and Murase, T. *Improved data distribution for multipath TCP communication*. in *GLOBECOM'05. IEEE Global Telecommunications Conference, 2005.* 2005. IEEE.

[92] Applegate, D. and Cohen, E., *Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs*, in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. 2003, ACM: Karlsruhe, Germany. p. 313-324.

[93] Zhang, Y. and Ansari, N., *On Architecture Design, Congestion Notification, TCP Incast and Power Consumption in Data Centers.* IEEE Communications Surveys & Tutorials, 2013. **15**(1): p. 39-64.

[94] Tam, A.S.W., Xi, K., Xu, Y., and Chao, H.J. *Preventing TCP incast throughput collapse at the initiation, continuation, and termination*. in *2012 IEEE 20th International Workshop on Quality of Service*. 2012.

www.manaraa.com

[95] *The ns-3 Network Simulator*. Available from: http://www.nsnam.org/.

[96] *PyViz ns-3 Simulation Visualizer*. Available from: http://www.nsnam.org/wiki/index.php/PyViz.

[97] Carneiro, G., Fortuna, P., and Ricardo, M. *FlowMonitor: a network monitoring framework for the network simulator 3 (NS-3)*. in *Proceedings of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools*. 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[98] Tange, O., *Gnu parallel-the command-line power tool.* The USENIX Magazine, 2011. **36**(1): p. 42-47.

[99] Chrysos, N., Neeser, F., Gusat, M., Minkenberg, C., Denzel, W., Basso, C., Rudquist, M., Valk, K., and Vanderpool, B., *Large switches or blocking multi-stage networks? An evaluation of routing strategies for datacenter fabrics.* Computer Networks, 2015. **91**: p. 316-328.

[100] Dukkipati, N. and McKeown, N., *Why flow-completion time is the right metric for congestion control.* SIGCOMM Comput. Commun. Rev., 2006. **36**(1): p. 59-62.

[101] Dean, J. and Barroso, L.A., *The tail at scale.* Commun. ACM, 2013. **56**(2): p. 74-80.

[102] Gummadi, P.K., Madhyastha, H.V., Gribble, S.D., Levy, H.M., and Wetherall, D. *Improving the Reliability of Internet Paths with One-hop Source Routing*. in *OSDI*. 2004.

[103] Johnson, D.B., Maltz, D.A., and Broch, J., *DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks.* Ad hoc networking, 2001. **5**: p. 139-172.

[104] Leung, R., Liu, J., Poon, E., Chan, A.-L., and Li, B. *MP-DSR: a QoS-aware multi-path dynamic source routing protocol for wireless ad-hoc networks*. in *Local Computer Networks, 2001. Proceedings. LCN 2001. 26th Annual IEEE Conference on*. 2001. IEEE.

[105] Fang, W., Liang, X., Sun, Y., and Vasilakos, A.V., *Network element scheduling for achieving energy-aware data center networks.* International Journal of Computers Communications & Control, 2014. **7**(2): p. 241-251.

[106] *Open Network Laboratory (ONL)*. Available from: https://onl.wustl.edu/.

[107] *Emulab - Network Emulation Testbed*. Available from: http://www.emulab.net/.

www.manaraa.com