# THE LOAD-BALANCED ROUTER

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Isaac Keslassy
June 2004

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Nick McKeown<br>
(Principal Adviser)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Balaji Prabhakar

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mark Horowitz

Approved for the University Committee on Graduate Studies.

# Abstract

The function of a router is to switch arriving packets to their correct output destination. A router is built to achieve a specified capacity (the sum of the rates of its interfaces), and users expect a router to consistently achieve this capacity. However, no commercial high-speed router can guarantee today that it will achieve its full capacity for all arrival traffic patterns. This is due to the difficulty of scheduling its switch fabric, and it will become even more difficult in the future as the number of interfaces and the interface speeds increase.

In this thesis, we advocate the use of a load-balanced router, a router architecture that is scalable and can guarantee a full capacity. A load-balanced router consists of two stages. First, a load-balancing stage spreads arriving packets equally among linecards. Then, a forwarding stage transfers packets from the linecards to their final destination. A load-balanced router does not use any centralized scheduler. Therefore, it can scale while providing the throughput guarantees needed by network operators.

In this thesis, we first explain how to simplify the load-balanced router architecture. While current routers commonly need switch fabrics with fast reconfiguration times, we show how to implement the load-balancing and forwarding stages of a load-balanced router using a single passive optical switch fabric with no reconfigurations. We also prove that among all possible switch fabrics with no reconfigurations, a specific load-balanced switch fabric uniquely achieves the maximum possible guaranteed capacity.

A problem with the load-balanced router is that different packets of the same flow can take different paths, possibly leading to packet reordering. In this thesis, we introduce a simple distributed algorithm that can avoid packet reordering while providing delay and capacity guarantees.

Finally, we present a practical switch fabric architecture that would enable load-balanced routers to scale to higher numbers of interfaces, and we prove that this architecture can adapt to arbitrary removals and additions of interfaces. We conclude by showing that the load-balanced router can help provide the scalability and capacity guarantees needed in the Internet.

# Acknowledgements

> *He who learns from a colleague a single chapter, a single law, a single verse, a single statement, or even a single letter, ought to pay him honor.*
>
> *— Pirkei Avot (6:3)*

How could I acknowledge adequately all those who taught and inspired me during the past five years at Stanford?

I could not have asked for a better adviser than Nick McKeown. Nick was always here to inspire, push, guide, ask, answer, motivate, support, compliment and complement whenever needed. Nick, you really were for me a source of both *unbelievable* wisdom and *unbelievable* knowledge, and I am deeply grateful to you.

I will always remember with passion my inspiring discussions with Balaji Prabhakar, my co-adviser, one of very few people in the world who teach you something new every time you speak with them.

Let me thank Mark Horowitz, David Miller and Olav Solgaard. Like my advisers, they are among the smartest and most famous professors in the world, and I am quite proud to have worked with them. They are why Stanford is so unique and why Stanford attracted me from France.

I am immensely grateful to Kate and John Wakerly for the Stanford Graduate Fellowship. I will deeply miss Kate's generosity, kindness and happiness. The support of the SGF, SNRC, DARPA/MARCO and Cisco provided me with the invaluable freedom of choosing the research topics I wanted.

I would like to acknowledge the great time spent with Da (Shang-Tse) Chuang, who collaborated on most of the work presented in this dissertation. Discussing and hurling ideas at each other has been incredibly enriching.

My thanks go to Rui Zhang-Shen and Nandita Dukkipati for the great fun we had in Gates 356, as well as to the other group members: Gireesh, Greg, Guido, Martin, Masayoshi, Matthew, Mingjie, Pablo, Sundar, Theresa and Yashar.

For all the good time during the Ph.D. years, I am grateful to my colleagues and friends from Stanford and around the world, including in Balaji's group, the Coop, Bell Labs and Taiwan's National Tsing Hua University. Huge thanks to Abtin, Alan, Allen, Amalia, Amelia, Anamaya, Ananthan, Arjun, Athina, Bill, Brian, Cheng-Shang, Chandra, Changhua, Chao-Kai, Chao-Lin, Christine, Christophe, Damon, Dana, Daniel, Danny, David, Denise, Derek, Devavrat, Dimitri, Duan-Shin, Elif, Emilio, Eric, Flavio, Giulio, Hanna, In-Sung, Ingrid, Joachim, Jonathan, Ken, Kevin, Kostas, Kyoungsik, Lakshman, Laurence, Lizzi, Marcy, Marissa, Mark, Maureen, Max-David, Mayank, Milind, Mina, Mohsen, Murali, Myles, Nathan, Neda, Neha, Nick, Ofer, Paolo, Pascal, Paul, Peter, Prashanth, Rivi, Rong, Ruben, Ryan, Sam, Sarah, Sylvia, Tali, Tsachy, Vinayak, Vincent, Vishal, Ygal and Yoav.

Finally, I am quite grateful to those who volunteered to review either the whole dissertation or parts of it, including Da Chuang, Mary McDevitt, Sundar Iyer, Bill Lin and Jonathan Sidi.

*This dissertation is dedicated...*

To the memory of my departed grandparents, z"l.

And to my family.

Aux oncles et cousins. A Mamie. A Papa, Maman et Michael.

*Merci ve-toda.*

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background

The Internet consists of end-hosts interconnected by links and routers. Packets sent by a source end-host transit through different links before arriving at their destination end-host. Routers form the junction between links. The role of a router is to switch each arriving packet from its input link to its output link. Therefore, a router logically consists of two consecutive stages. First, a lookup stage determines the appropriate output link of each packet based on the address of its destination. Then, a switching stage transfers the packet from its input link to its output link.

This thesis focuses on the switching stage of high-capacity routers. Let a flow be the set of all packets with the same input and output links. Today, in the most common router architectures, packets from the same flow take the same path through the router. However, these architectures typically have several problems: they require a complex centralized scheduler; they do not provide the throughput guarantees that network operators need to make efficient use of their expensive long-haul links; and they are sensitive to single-point failures, because all packets from the same flow take the same path. These considerations have led to a recent interest in introducing parallelism inside routers, that is, in making the packets take different paths. Examples of multi-path routers include Parallel Packet Switch (PPS) routers [43, 45, 46], Parallel Shared Memory (PSM) and Distributed Shared Memory (DSM) routers [47], Memory-Space-Memory (MSM) routers [24, 25], buffered Clos networks [17, 23, 65], and ring, torus, and hypercube interconnection networks [30, 31, 38, 69, 75, 76].

However, these multi-path routers commonly share the very same problems they were supposed to solve in single-path routers. In particular, they either require complex, centralized schedulers, or rely on simple distributed scheduling algorithms that lack throughput guarantees. More recently, C.-S. Chang *et al.* introduced the Load-Balanced Router architecture [20, 21]. This architecture is based on load-balancing packets uniformly inside the router before forwarding them to their correct destination, an idea first introduced by Valiant *et al.* [83, 84]. As developed later in the Introduction, C.-S. Chang *et al.* show that a load-balanced router does not require any scheduler and that it can guarantee 100% throughput for a broad class of traffic. Therefore, the load-balanced router is not subject to the two main problems commonly present in former architectures: centralized scheduling and the lack of throughput guarantees needed by network operators. This makes the load-balanced router an appealing architecture to study. However, as detailed later, the load-balanced router suffers from several problems, such as packet reordering and the need for frequent switch fabric reconfigurations. In this thesis we will show how to solve these problems and demonstrate that the load-balanced router can help improve router performance.

The remainder of the introduction expands on the load-balanced router. Section 1.2 first introduces the basic load-balancer router architecture and its main properties. Section 1.3 then presents several problems associated with this architecture. These problems prevent the implementation of the basic load-balanced router architecture in high-speed routers. This thesis provides solutions to these problems and explains how a load-balanced router can be practically implemented, as outlined in Section 1.4.

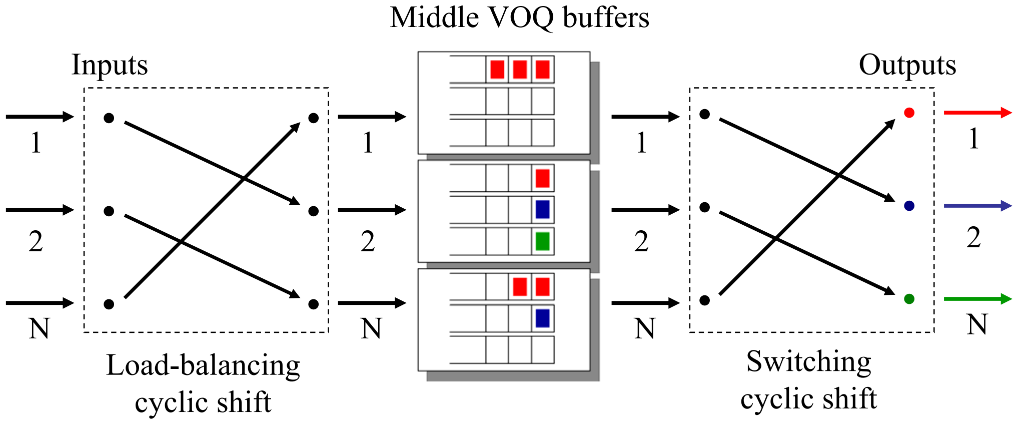


Figure 1.1: Basic load-balanced router.

## 1.2 The Load-Balanced Router

### 1.2.1 Assumptions

Throughout this thesis, we assume for simplicity that all incoming variable-size packets are segmented into *fixed-size packets*, or simply *packets*, and reassembled when leaving the router. We say that links are connected to routers through *linecards*, and denote by $N$ the number of input (and output) linecards. We also assume that all linecards have the same line rate and that time is slotted, so that at most one packet can arrive at any input port and at most one packet can depart from any output port in each time-slot. Finally, we assume that initially there is no packet in the router.

### 1.2.2 Basic Load-Balanced Router Architecture

A basic load-balanced router is shown in Figure 1.1. It consists of a single stage of $N$ buffers, sandwiched by two identical stages of switching. Each buffer is partitioned into $N$ separate FIFO (First-In-First-Out) queues, one per output (hence we call them virtual output queues, VOQs). Each of the two switching stages goes through the same pre-determined cyclic shift configuration. That is, at time $t$, input $i$ of each switching stage is connected to output $[(i + t - 1) \bmod N] + 1$. Therefore, the configuration is a cyclic shift, and each input is connected to each output exactly

$\frac{1}{N}$-th of the time, regardless of the arriving traffic. We will call each switching stage a *fixed, equal-rate switch*.

Although they are identical, it helps to think of the two stages as performing different functions. The first stage is a load-balancer that spreads traffic over all the VOQs. The second stage is an input-queued crossbar switch in which each VOQ is served at a fixed rate. When a packet arrives at the first stage, it is immediately transferred to an intermediate input, which depends on the current configuration of the load-balancer. In the intermediate input, the packet is stored in a VOQ according to its eventual output. Sometime later the VOQ will be served by the second fixed, equal-rate switch. The packet will then be transferred across the second switch to its output, from where it will depart the system.

At first glance, it is not obvious how the load-balanced router can make any throughput guarantees; after all, the sequence of switch configurations is predetermined, regardless of the state of the VOQs or the traffic rates. In a conventional single-stage crossbar switch, throughput guarantees are only possible if a scheduler configures the switch based on a knowledge of the states of all the VOQs or of all the traffic rates. However, C.-S. Chang *et al.* prove in [20] that the load-balanced router, without any scheduler, will still be able to provide 100% throughput for a broad class of arrival traffic. This is an important theorem for the understanding of the load-balanced router, and thus we choose to reproduce it here. But first, we introduce the definitions needed to present the theorem.

### 1.2.3 Definitions

Let $a = \{a(t), t \geq 1\}$ be the sequence of traffic arrivals at the first switching stage, $b = \{b(t), t \geq 1\}$ the sequence of traffic departures from the first switching stage (and arrivals at the buffering stage), $q = \{q(t), t \geq 1\}$ the sequence of queue lengths, and $P = \{P(t), t \geq 1\}$ the sequence of switch permutation matrices. $P$ defines the connections between the inputs (rows) and the outputs (columns) of the first switching stage. As defined above, at time $t$, input $i$ of each switching stage is connected to output $[(i + t - 1) \bmod N] + 1$. Therefore, $P$ is periodic, and $P(t)$ is the $[t \bmod N]$-th diagonal matrix. For instance, $P(0) = P(N) = P(2N) = ... = Id$, the identity matrix. In addition, we have $b = P \cdot a$.

For each sequence $a$, let $\theta_s(a) = \{a(t+s), t \geq 1\}$ represent the time-shifted version of $a$.

**Stationary process:** A stochastic process $a$ is said to be stationary if it has the same joint distribution as its time-shifted version; i.e., for any time-shift $s$ and any set $A$,

$$Pr(a \in A) = Pr(\theta_s(a) \in A).$$

In other words, $a$ is stationary if all its statistical properties are invariant with respect to time-shifts.

**Ergodic process:** A stationary process $a$ is said to be ergodic if for all sets $A, B$,

$$\lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} Pr(\theta_s(a) \in A, a \in B) = Pr(a \in A)Pr(a \in B).$$

Intuitively, $a$ is ergodic if it tends in probability to a limiting form that is independent of the initial conditions. In addition, for a stationary ergodic sequence $a$, time-averages are equal to ensemble-averages [40]; that is,

$$\lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} a(s) = Ea(1), \quad a.s.$$

**Weakly mixing process:** A stationary process $a$ is said to be weakly mixing if for all sets $A, B$,

$$\lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} |Pr(\theta_s(a) \in A, a \in B) - Pr(a \in A)Pr(a \in B)| = 0.$$

Weak mixing is a measure of how fast the past loses influence over the future. Weak mixing implies ergodicity, and therefore is a stronger property. Most Internet traffic models are stationary and weakly mixing, and therefore also ergodic. Since stationarity of traffic models is often implied by the constancy of the traffic generator, we illustrate the weak mixing property of Internet traffic models. Bernoulli traffic is weakly mixing because successive events are independent. On/off Markov bursty traffic is weakly mixing because the influence of the past on the future decreases geometrically. Fractional Gaussian noise, a model for self-similar traffic, is also weakly mixing [62, 86]. However, periodic traffic is not weakly mixing because the influence of the past does not decrease over time.

**Admissible mean rate:** Let $r = Ea(1)$ be the mean rate matrix of a stationary ergodic arrival process $a$. $r_{ij}$ represents the mean arrival rate for packets going from input $i$ to output $j$. Then $r$ is admissible if for all $j$,

$$\sum_{i=1}^{N} r_{ij} < 1,$$

and for all $i$,

$$\sum_{j=1}^{N} r_{ij} < 1.$$

**100% throughput:** A switch has 100% throughput for a given class of arrival traffic if for any arrival process with admissible mean rate, $q(t)$ is upper-bounded by a random matrix that converges in distribution.

Such a property implies that for any arbitrarily small probability $\epsilon > 0$, there is a corresponding queue size such that after some time-slot, the probability that $q$ will exceed this queue size is bounded by $\epsilon$. When $q$ is ergodic, this also implies that the workload will exceed this queue size at most a fraction $\epsilon$ of the time.

**Work-conserving system:** A queueing system is work-conserving if and only if it is non-idling when there are customers in the queue.

### 1.2.4 100% Throughput Guarantee of a Basic Load-Balanced Router

We will now give an intuitive explanation why a basic load-balanced router guarantees 100% throughput for a broad class of arrival traffic. Consider a single fixed, equal-rate crossbar switch, in which each VOQ is served $\frac{1}{N}$-th of the time. Suppose that the arrival traffic is uniform; that is, packets arriving at a switch are equally likely to be destined to any of the $N$ outputs. If the arrival traffic is admissible, then the arrival rate to each input is less than 1, and therefore the arrival rate to each VOQ is less than its service rate $\frac{1}{N}$. For stationary and ergodic arrivals and services, this implies that the single fixed, equal-rate switch has 100% throughput for uniform arrival traffic [7, 18, 59].

Of course, real network traffic is *not* uniform. But an extra load-balancing stage can spread out non-uniform traffic, making it sufficiently uniform to achieve 100% throughput. This is the basic idea of the two-stage load-balanced router. A load-balancing device spreads packets evenly to all the inputs of a second, fixed, equal-rate switch. We will now prove the following theorem:

**Theorem 1** *A basic load-balanced router guarantees 100% throughput for any stationary and weakly mixing arrival traffic with admissible mean rate.*

*Proof:* The proof follows the proof in [20], but with a few changes. In [20], the authors assume that the two crossbars cycle through the same set of $N$ permutations in the same order, but the cycles of the two crossbars are randomly out-of-phase. In particular, the states at time $t = 0$ of both crossbars are chosen independently at random among the $N$ possible permutations. For practical purposes, we assume in this thesis that at each time-slot the two crossbars use the same permutation, and that the initial permutation at $t = 0$ is fixed. However, such an assumption implies that the sequence $P$ is not stationary and ergodic, since $P(0)$ is fixed. This property is needed in the proof. Therefore, without loss of generality, we (theoretically) re-assign the start of time in the router uniformly at random among time-slots $\{-N, ..., -1\}$. As a consequence, the state of the first permutation $P(0)$ is chosen uniformly at random among the $N$ possible values, and we can verify that $P$ is stationary and ergodic. This change of time definition does not affect the arrivals, which are stationary and independent of $P$. The expected value of $P(t)$ is $EP(t) = \frac{1}{N}e$, where $e$ is the $N \times N$ matrix with all 1's.

Let $r$ be the admissible mean rate of $a$. Since $b = P \cdot a$, and $a$ and $P$ are stationary and independent, $b$ is stationary with average rate

$$Eb(t) = EP(t) \cdot Ea(t) = \frac{1}{N} e \cdot r.$$

In addition, $a$ is weakly mixing and $P$ is ergodic, therefore $b$ is ergodic (Theorem 2.6.1 in [68]). Using the fact that $b$ and $P$ are stationary and ergodic, we get:

$$\lim_{t\to\infty} \frac{1}{t} \sum_{s=1}^{t} (b(s) - P(s)) = Eb(t) - EP(t) = \frac{1}{N} e \cdot r - \frac{1}{N} e < 0, \quad a.s.$$

As proved by Loynes [7, 18, 59], if arrivals and services of a work-conserving switch are stationary and ergodic, and at any time the expected number of arrivals is less than the expected number of services, then the switch has 100% throughput. Consider the VOQ of intermediate input $i$ containing all packets destined to output $j$. This VOQ is serviced exactly every $N$ time-slots as long as it is non-empty. We can consider this VOQ every $N$ time-slots, and define the sampled state of the VOQ as *the sampled VOQ*. At every sample, the sampled VOQ is serviced as long as it is non-empty. Therefore, the sampled VOQ is work-conserving. In addition, $b$ and $P$ are stationary and ergodic, and $Eb(t) - EP(t) < 0$. Therefore, the arrivals and services to the sampled VOQ are stationary and ergodic, and the expected number of arrivals is less than the expected number of services. Using the above result by Loynes, we conclude that the switch has 100% throughput. ■

### 1.2.5 Advantages of a Basic Load-Balanced Router

In addition to the 100% throughput property presented above, a basic load-balanced router has several other appealing properties when compared with traditional centralized-scheduler architectures. These properties involve all the main router elements: *scheduling*, *control*, *switch fabric* and *linecards*.

First, a basic load-balanced router uses *no centralized scheduler*. This is unlike the most common current router architectures, which use a centralized arbiter that computes a match between the inputs and the outputs for each time-slot [2, 3, 28, 58, 60, 61, 80]. Indeed, a basic load-balanced router does not need *any* scheduling, since the sequence of crossbar permutations is predetermined.

Second, since there is no centralized scheduler, a basic load-balanced router does not need any back-and-forth exchange of VOQ-state and scheduling-decision information between the linecards and a centralized scheduler. This simplifies *the switch control*. It also removes the loss of bandwidth due to the exchange of information.

Third, a basic load-balanced router simplifies *the switch fabric*. It uses only $N$ states for the switch fabric out of the possible $N!$ states. The state sequence is predetermined, and not computed at each time slot.

Finally, a basic load-balanced router simplifies *the linecards*, and especially the buffering stage. It only uses one stage of buffering, while typical centralized-scheduler switches run the switch fabric faster than the line rate and require two stages of buffering (at the inputs and at the outputs). A basic load-balanced router also typically requires less buffering per linecard because it spreads long bursts across the intermediate inputs, as shown in [20].

All these properties suggest that it might be possible to scale a basic load-balanced router more easily than traditional architectures. The remainder of this thesis consists in solving problems that might occur with the basic load-balanced router architecture and prevent it from being implemented in high-speed routers.

## 1.3 Motivation of the Thesis

### 1.3.1 Optical Switch Fabric

The use of optics in the switch fabric is often desired to scale routers to higher speeds and consume less power than electronic interconnects. However, because of the switch fabric reconfigurations, two factors prevent the use of high-capacity optical switch fabrics in the basic load-balanced router. First, high-capacity optical switch fabrics often have slow reconfiguration times. For instance, MEMS-switch reconfiguration times are in tens of milliseconds [66, 72]. However, with 64-byte packets and speeds of 40-160 Gbps, a reconfiguration time of a few nanoseconds is needed. In

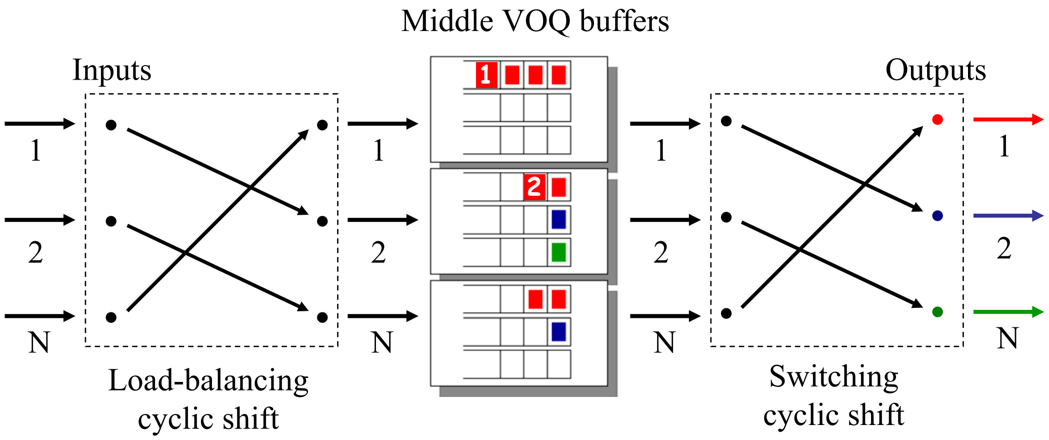


Figure 1.2: Example of reordering in the basic load-balanced router.

addition, tunable optical devices are often significantly more expensive than fixed devices. Consequently, our objective is to find a mechanism that allows for the use of optics in a load-balanced router without any need for reconfiguration.

### 1.3.2 Packet Reordering

Another problem in the load-balanced router is that packets can be reordered. This is because the load-balancer spreads packets without regard to their final destination or to when they will depart from the VOQs. Figure 1.2 illustrates how two packets arriving back to back at the same input, and destined to the same output, are spread to different intermediate linecards. These two packets, numbered 1 and 2, are successively sent to the first two intermediate linecards. Since the intermediate linecard of packet 1 has a higher VOQ occupancy than the intermediate linecard of packet 2, packet 1 will leave the router after packet 2. Therefore, the two packets leave the router out-of-order.

While reordering packets is allowed (and is common [10, 49]) in the Internet,[1] network operators generally insist that routers do not reorder packets belonging to the same application flow. In addition, in its current version, TCP does not perform

[1] Internet RFC 1812 “Requirements for IP Version 4 Routers” [8] does not forbid packet reordering.

well when packets arrive at the destination out of order. Such packets can be perceived as loss indicators and might trigger unnecessary retransmissions, thus reducing the throughput provided to the application [13].

### 1.3.3 Pathological Traffic Patterns

Even though the load-balanced router guarantees 100% throughput for a large class of traffic patterns, it is still vulnerable to some pathological traffic patterns. For instance, if at each time-slot $t$ a packet arrives at input $[t \bmod N] + 1$ and is destined to output 1, it will always be placed in the same first VOQ of the first intermediate linecard. Consequently, the traffic arriving at the forwarding stage is clearly non uniform, and its throughput is only $\frac{1}{N}$. This example illustrates how pathological periodic traffic patterns prevent a basic load-balanced router from providing significant throughput guarantees.

### 1.3.4 Missing Linecards

A final problem with the predetermined configuration of a basic load-balanced router is that the router does not work properly when a linecard is missing. This situation might happen if the operator wants to slowly build up the number of linecards instead of installing all the linecards at once. It might also occur if a linecard fails or if linecards are removed in order to connect some lines to a different router.

The basic load-balanced router architecture assumes that any input linecard equally load-balances its traffic across all intermediate linecards. However, if an intermediate linecard is missing, packets going through this linecard will be lost. Therefore, a missing intermediate linecard will prevent the load-balanced router from working as expected.

We have seen how many problems in a basic load-balanced router architecture prevent its implementation in high-speed routers. This thesis provides solutions for these problems, as outlined in Section 1.4.

## 1.4 Outline of the Thesis

This thesis explains how a load-balanced router architecture can be used to design and implement high-speed routers. The following chapters model and solve the problems described above.

Chapter 2 shows how to use optics in the switch fabric. It proposes an implementation of the load-balanced router based on meshes, and explains why this implementation involves no reconfiguration. It also presents a simple implementation based on a passive AWGR (Arrayed Waveguide Grating Router) optical device. Finally, it proves that among all possible switch fabrics with no reconfigurations, a specific load-balanced switch fabric uniquely achieves the maximum possible guaranteed throughput.

Chapter 3 then shows how a novel algorithm can avoid both the packet reordering and the pathological traffic pattern problems. It shows that this algorithm keeps packets in order, is practical to implement, and prevents pathological traffic patterns. It also proves that for any arbitrary adversarial traffic, this algorithm keeps the average packet delay through the switch within a constant from that of an ideal (output-queued) switch.

In Chapter 4, we explain why the usual implementations of the load-balanced router do not adapt easily to larger port counts and to linecard failures. This leads me to introduce an architecture based on MEMS (MicroElectroMechanical Systems). This architecture provides a practical implementation that could be chosen to build a high-speed router. We then further study the practicality and reliability of this implementation and solve two related problems: how to arrange the MEMS switches and how to find a correct linecard schedule upon linecard failure or addition.

I conclude this thesis in Chapter 5 by explaining how a load-balanced router helps scale the size and capacity of routers while providing a 100% throughput guarantee.

The appendices contain many of the proofs of statements made in the thesis.

# Chapter 2

# Mesh Model

The Introduction described the load-balanced router, a new architecture that requires no centralized scheduler and yet guarantees 100% throughput under a broad class of traffic patterns. In this chapter, we will present how the load-balanced router can be practically implemented using an optical switch fabric with no reconfiguration. In particular, we will show that it can be implemented using a fixed mesh of optical fibers. Then, we will prove that a specific mesh architecture achieves the best throughput among all possible fixed architectures, and can only achieve this throughput by using load-balancing.

## 2.1 From Crossbar to Mesh

### 2.1.1 Mesh Architecture

The load-balanced router architecture relies on two *fixed, equal-rate switches*. Each of these switches connects any input to any output exactly $\frac{1}{N}$-th of the time, regardless of the arriving traffic. We saw in the Introduction that these fixed, equal-rate switches could be implemented using two $N \times N$ switch fabrics that are reconfigured at every time-slot. However, if we were to use an optical switch fabric, this constraint would hinder our ability to scale to higher speeds. For instance, MEMS-switch reconfiguration times are in tens of milliseconds [66, 72], while we would need reconfiguration times of a few nanoseconds. In addition, a system with no reconfigurations

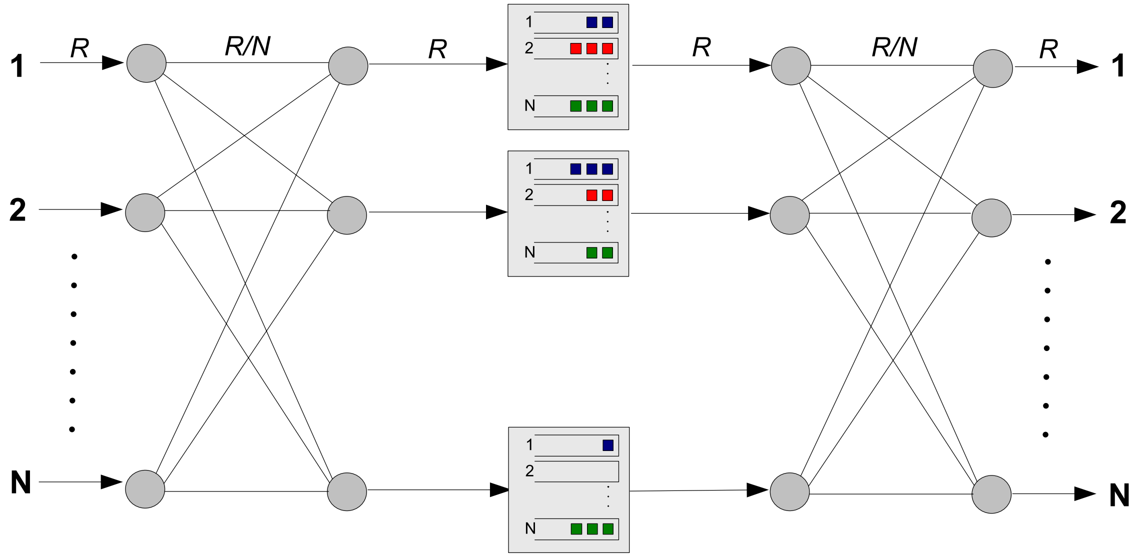


Figure 2.1: Load-balanced router architecture based on a double mesh.

is obviously simpler and more reliable than a system with frequent reconfigurations. Therefore, we would like to replace this fixed, equal-rate switch with a fixed system that does not need any reconfigurations. This is realized with *a fixed mesh of optical channels*.

A first observation is that we can replace each fixed equal-rate switch with $N^2$ fixed channels at rate $R/N$, as illustrated in Figure 2.1. Hence, each switch is replaced by a uniform mesh. The rates provided between any switch input and any switch output will stay the same. Therefore, the switch is still a fixed, equal-rate switch.

A second observation is that we can replace the two meshes with a single mesh running twice as fast, as shown in Figure 2.2(a). This is possible because in a physical implementation, a linecard contains an input, an intermediate input and an output. Every packet traverses the mesh twice, each time at rate $R/N$; therefore, the total channel rate is $2R/N$. After a packet crosses the switch the first time, it is stored in an intermediate linecard; from there, it crosses the switch again to reach the output linecard.[1]

A third observation is that a uniform mesh with optical channels can be replaced

[1] Chapter 4 discusses how to provide more scalability and flexibility by extending this architecture.

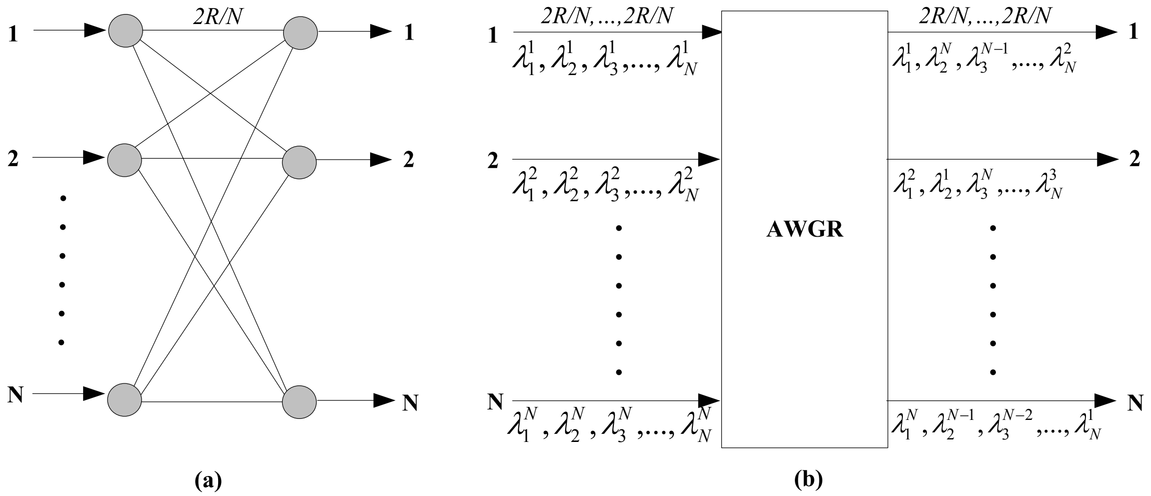

1
2
N
2R/N
1
2
N
(a)
1
2
N
2R/N,...,2R/N
$\lambda_1^1, \lambda_2^1, \lambda_3^1, ..., \lambda_N^1$
$\lambda_1^2, \lambda_2^2, \lambda_3^2, ..., \lambda_N^2$
$\lambda_1^N, \lambda_2^N, \lambda_3^N, ..., \lambda_N^N$
AWGR
2R/N,...,2R/N
$\lambda_1^1, \lambda_2^N, \lambda_3^{N-1}, ..., \lambda_N^2$
$\lambda_1^2, \lambda_2^1, \lambda_3^N, ..., \lambda_N^3$
$\lambda_1^N, \lambda_2^{N-1}, \lambda_3^{N-2}, ..., \lambda_N^1$
1
2
N
(b)

Figure 2.2: Load-balanced router architecture based on (a) a single mesh, and (b) an AWGR.

by an Arrayed Waveguide Grating Router (AWGR), as represented in Figure 2.2(b). Input $i$ transmits $N$ distinct channels on its outgoing fiber. Each different channel $\lambda_w^i$ is transmitted at rate $2R/N$ on a specific wavelength $\lambda_w$. The AWGR, a passive optical device, shuffles the channels such that each channel of a given input is destined to a different output. As a result, the system behaves as a single mesh — indeed, it *is* a mesh, but based on an AWGR instead of the more conventional $N^2$ fibers. Its main advantage is the reduction in the number of fibers needed from $N^2$ to $2N$.

### 2.1.2 Uniform Multiplexing

How is it that we can implement the load-balanced router equivalently using a crossbar, a mesh or an AWGR? Are there any other equivalent architectures?

Actually, the load-balanced router architecture only assumes that the switch fabric is able to spread arriving traffic uniformly across the $N$ outputs, and each output is able to receive traffic uniformly from the $N$ inputs. This simply is a uniform multiplexing (and demultiplexing), a process well-known in the literature. As a consequence, the uniform multiplexing of arriving traffic could be realized by Time Division Multiplexing (crossbar), Space Division Multiplexing (mesh), Wavelength Division Multiplexing (AWGR), and so on, as well as a combination of these methods. Similarly, any other architecture would be sufficient, as long as it can uniformly multiplex and demultiplex traffic.

## 2.2 The Optimal Mesh

### 2.2.1 Motivation

Perhaps the most interesting characteristic of the load-balanced router is that it provably achieves 50% throughput with a single switch fabric (and therefore achieves 100% throughput with two switch fabrics, or at a speedup of two) for a broad class of arrival traffic.

However, it is not obvious why the switch needs to have a fixed, equal rate, i.e., why the mesh needs to be uniform. Do all links need to have a capacity of $R/N$? How would the throughput change if the mesh was not uniform? What arrangement of link capacities maximizes the throughput? If linecards were instead interconnected using a ring, a torus or a hypercube, would throughput be higher?

To make the comparison, we will consider a load-balanced switch with arbitrary link capacities and an arbitrary load-balancing policy. We will allow packets to take any path through this switch, using any number of hops. We will only require that this switch does not use any speed-up, and that each packet goes at least once through the switch. Then, we will determine the architecture that has the highest guaranteed throughput.

We will now first define the problem and provide some examples. Then, we will prove that a given biased mesh reaches the maximum throughput and is unique in doing so. Finally, we will provide some intuition on the results.

### 2.2.2 Problem Formulation

**Notations and Assumptions**

We define a *doubly stochastic* matrix to be a non-negative square matrix with all row and column sums equal to 1. Similarly, we define an *admissible* (or doubly sub-stochastic) matrix to be a non-negative square matrix with all row and column sums upper-bounded by 1. Finally, we define the time unit such that each node can send and receive at most one bit per second (if the maximum node speed is $R$, scale the time unit by a factor $\frac{1}{R}$).

A link of fixed capacity $C_{ij}$ connects node $i$ to node $j$, where $1 \leq i, j \leq N$. The matrix $C = [C_{ij}]_{1 \leq i,j \leq N}$ is the capacity matrix, and any node $l$ can send up to $\sum_{j=1}^{N} C_{lj}$ (and likewise receive at most $\sum_{i=1}^{N} C_{il}$) bits per time unit to and from the N nodes (including itself). Since every node $l$ can send and receive at most one bit per time unit, $\sum_{i=1}^{N} C_{il} \leq 1$ and $\sum_{j=1}^{N} C_{lj} \leq 1$; therefore, the matrix $C$ is admissible. The capacity matrix $C$ defines the architecture; for example, the *uniform mesh architecture* (in which nodes are connected to each other with equal-capacity links), corresponds to the uniform matrix $C$ where $C_{ij} = 1/N$. Similarly, a ring could be defined by $C_{ij} = \mathbf{1}_{\{j=i+1 \bmod N\}}$.

Denote by $T$ the arrival traffic rate matrix, with $T_{ij}$ being the arrival rate at node $i$ of packets destined for node $j$. We will assume that $T$ is admissible, since it cannot be supported otherwise. Suppose we want to load-balance these packets across multiple paths, each path having an arbitrary number of hops. If $P(i, j)$ is the set of paths between nodes $i$ and $j$, then any path $p \in P(i, j)$ can be represented as $(i \to node_1 \to node_2 \to \ldots \to j)$. Let $T_{ij}^{p}$ be the rate of the flow carried by $p$. If the arrival traffic rate matrix $T$ is feasible (i.e., the network has 100% throughput for $T$), it is possible to decompose $T$ into several paths $p$, and therefore for all $i, j$,

$$T_{ij} = \sum_{p \in P(i,j)} T_{ij}^{p}. \tag{2.1}$$

Similarly, we will define the effective load matrix $L$ using for all $i, j$:

$$L_{ij} = \sum_{\{p:(i \to j) \in p\}} T_{ij}^{p}. \tag{2.2}$$

The effective load of a link is the sum of the loads of the paths sharing the link. A solution is feasible if and only if we can find a decomposition of $T$ such that $L \leq C$, i.e., no link is over-booked.

### Problem Intuition

Suppose that $N = 2$ and that we use a uniform mesh architecture, with capacity matrix

$$C = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}.$$

We will use this example to gain some intuition about the throughput of interconnection networks.

If the arrival rate matrix is

$$T_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0 \end{pmatrix}$$

then we cannot send traffic at rate 0.9 on the path 1 → 1, because the capacity is limited by $C_{11} = 0.5$. Therefore, *we need to load-balance the traffic by using the spare capacity of other links*. We will send 0.5 on the direct path 1 → 1, and the remaining 0.4 on the alternative path 1 → 2 → 1. The resulting load matrix is

$$L_1 = \begin{pmatrix} 0.5 & 0.4 \\ 0.4 & 0 \end{pmatrix},$$

and $L_1 \leq C$. Clearly, the direct path is not always sufficient to carry the required rate matrix, but in this case it is possible to use a load-balanced path in order to carry it.

Not all rate matrices are feasible, i.e., the throughput is not always 100%. Consider the arrival rate matrix

$$T_2 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.9 \end{pmatrix}.$$

Sending 0.5 on $1 \to 1$, 0.4 on $1 \to 2 \to 1$, 0.5 on $2 \to 2$ and 0.4 on $2 \to 1 \to 2$, the load matrix is

$$L_2 = \begin{pmatrix} 0.5 & 0.8 \\ 0.8 & 0.5 \end{pmatrix},$$

and so $L_2 \nleq C$. In this particular case, we need to scale down $T_2$ to

$$\begin{pmatrix} 0.75 & 0 \\ 0 & 0.75 \end{pmatrix}$$

for the solution to be feasible.

Finally, load-balancing does not always help, particularly in small matrices when there are not many paths to divert traffic away from congested links. And it is always useless to divert traffic to oneself. For example, consider the rate matrix

$$T_3 = \begin{pmatrix} 0 & 0.5 + \epsilon \\ 0.5 & 0 \end{pmatrix},$$

where $\epsilon > 0$. Sending traffic on the path $1 \to 1 \to 2$ does not divert traffic from the congested link $1 \to 2$; therefore, $T_3$ is not feasible. This teaches us that when sending traffic from node $i$ to node $j \neq i$, it is clearly useless to use the link $i \to i$, because traffic is transferred across the network with no benefit. By comparing $T_1$, $T_2$ and $T_3$, this example also shows that finding the maximum throughput of a given rate matrix is not straightforward, even when $N = 2$. Moreover, since the number of cases to consider increases with $N$, such a problem is increasingly difficult to solve as $N$ grows.

### Problem Definition

Our objective is to find the load-balanced network with the largest throughput guarantee. In other words, we want to find a network with a guaranteed throughput

$\theta^*$, where $\theta^*$ satisfies two properties. First, given any admissible arrival traffic, the network guarantees a throughput $\theta^*$, i.e., it will switch a fraction $\theta^*$ of the traffic for any input-output flow. And second, no other network can have a better guaranteed throughput than $\theta^*$. We will define the problem by decomposing it into three successive optimization problems. First, we will find the throughput for a given network and a given rate matrix. Then, we will obtain the worst-case throughput of a network, which can be achieved for any rate matrix. Finally, we will provide $\theta^*$, which is the best guaranteed throughput among all networks.

In the first optimization, we want to find the maximum throughput for a given network and a given rate matrix. In other words, given capacity matrix $C$ and rate matrix $T$, we want to find the best possible throughput $\theta(C, T)$, such that the scaled-down rate demand matrix $\theta(C, T) \times T$ is feasible. Put mathematically,

$$\theta(C,T) \equiv \max_{\theta}(\theta), \text{ subject to:}$$

$$\begin{array}{lll} (i) & \sum_{p=1}^{P(i,j)} T_{ij}^p = \theta \times T_{ij} & \forall i,j \\ (ii) & L(i,j) \equiv \sum_{\{p:(i \to j) \in p\}} T_{ij}^p \le C_{ij} & \forall i,j \\ (iii) & T_{ij}^p \ge 0 & \forall i,j,p \end{array}$$

In words, the throughput $\theta(C, T)$ is the maximum of the set of throughputs $\theta$ that satisfy three feasibility conditions. First, the arriving traffic is a scaled-down version of $T$ by a factor $\theta$, such that it can be decomposed into several paths $p$. The second condition is that the sum of the loads of the paths must be less than $C$, i.e., that the load matrix is feasible. The last condition is that the rate on each path must be nonnegative.

The second optimization finds the guaranteed maximum throughput $\theta(C)$ for the network. This is the throughput that is achievable by any rate matrix in the network, and, therefore,

$$\theta(C) \equiv \min_{T \text{ admissible}} (\theta(C,T)). \tag{2.3}$$

Finally, we find the maximum possible guaranteed throughput for any network, yielding a guaranteed throughput $\theta^*$, where

$$\theta^* \equiv \max_{C \text{ admissible}} (\theta(C)). \tag{2.4}$$

### 2.2.3 Examples of Guaranteed Throughput

#### Guaranteed Throughput of the Uniform Mesh

The uniform mesh is an architecture in which all links have the same capacity, i.e., $C_{ij} = 1/N$ for all $i, j$. We will show that the maximum guaranteed throughput of the uniform mesh is 50%.

We saw already in the Introduction why the uniform mesh guarantees at least 50% throughput, although the proof was based on slightly different assumptions. In short, each packet goes through both the load-balancing stage and the forwarding stage, and therefore through two hops. Consequently, the link between node $i$ and node $j$ can receive load in two possible ways. Either node $i$ is sending traffic to some node $k$ and spreads it using the intermediate node $j$, or some node $l$ sends traffic to node $j$ and spreads it using the intermediate node $i$. Mathematically, $L_{ij} = \sum_k T_{ik} + \sum_l T_{lj} \leq 2$ with an admissible $T$. Therefore, $\theta(C) \geq 50\%$.

The following example shows that it is not possible to do better using a different load-balanced routing algorithm.

Assume that

$$T = \begin{pmatrix} 0 & x & 0 & \dots & 0 \\ 0 & 0 & x & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & & & 0 & x \\ x & 0 & \dots & 0 & 0 \end{pmatrix},$$

where $x \geq 1/2$. A node $i$ can send at most $C_{i(i+1 \bmod N)} = 1/N$ amount of traffic directly.[2] It also needs to send the remaining $x - 1/N$ amount of traffic to load-balanced paths, with each of these paths using at least two links. Hence, the total traffic load contributed by each node to the system is at least $(1/N) + 2(x - 1/N)$,

[2] The modulo function takes values in $\{1, ..., N\}$ when nodes are numbered $\{1, ..., N\}$.

which implies that the total traffic load contributed by the $N$ nodes is $N(1/N + 2(x - 1/N)) = 2Nx - 1$. As we saw earlier, diagonal elements do not help load-balancing, and with this rate matrix they are also useless for direct paths. Hence, the total useful traffic capacity is the sum of all non-diagonal elements of $C$, i.e., $N \cdot (1 - 1/N) = N - 1$. For the solution to be feasible, we need $2Nx - 1 \leq N - 1$, which translates into $x \leq 1/2$. And so there exists a traffic rate matrix that is only feasible with a throughput of at most 50%. This implies $\theta(C) \leq 50\%$. Since we found that the two-hop algorithm provides a throughput of 50%, it follows that

$$\theta(C) = 50\%. \tag{2.5}$$

Further, it is not possible to improve on the two-hop algorithm.

### Guaranteed Throughput of a Ring

As a second example, consider a network in which the nodes are connected in a uni-directional ring, i.e., node $i$ is connected to node $(i + 1)$ mod $N$. Recall that we assumed that each packet needs to go at least once through the network. In the worst case, $T$ is the identity matrix so that nodes only send traffic to themselves through the ring. Therefore, all packets cross $N$ links, and the throughput $\theta(C_{ring}, T)$ is equal to $1/N$. This $T$ is the worst case, since packets do not need to use more than $N$ links to reach their destination. Therefore,

$$\theta(C_{ring}) = 1/N, \tag{2.6}$$

which — as expected — is much lower than for the uniform mesh.

### Guaranteed Throughput of a Permutation Matrix

The ring is a special case of a permutation matrix $\sigma$ of the set $\{1, ..., N\}$, where $\sigma$ is the capacity matrix of a network. The matrix $\sigma$ can be represented as a $0 - 1$ matrix with exactly one 1 in each row and column; i.e., $\sigma_{ij} = 1$ if $\sigma(i) = j$, and

$\sigma_{ij} = 0$ otherwise. Since $\sigma$ is a permutation, it can be decomposed as a product of disjoint cycles (the decomposition is unique up to the order of the cycles).

If $\sigma$ can be written as a single cycle of length $N$, we can assume without loss of generality that $\sigma(1) = 2$, $\sigma(2) = 3$,..., $\sigma(N) = 1$, and so $\sigma$ is the capacity matrix of a ring, with $\theta(\sigma) = 1/N$.

Alternatively, if $\sigma$ can be written as the product of two or more cycles, then there are two nodes $i$ and $j$ such that node $i$ is in the first cycle and node $j$ is in the second one. It is then impossible to reach node $j$ from node $i$ (the capacity graph is not connected), hence the throughput for any matrix $T$ such that $T_{ij} = 1$ is zero, and $\theta(\sigma) = 0$.

This example illustrates that the throughput of a capacity matrix is sensitive to its coefficients; and that the throughput of a disconnected graph is zero.

### 2.2.4 Properties of the Guaranteed Throughput

In the above examples, we computed the throughputs of several capacity matrices, but found that it is not straightforward in general to compute throughput directly. Since we want to find the capacity matrix with the largest guaranteed throughput, we will use general properties of the throughput function. We will start by showing that it is concave in $C$, scales linearly, and is strictly increasing.

**Concavity**

First, we show that throughput is concave in $C$. Assume that two capacity matrices $C_1$ and $C_2$ achieve throughputs of $\theta(C_1, T)$ and $\theta(C_2, T)$ for a rate matrix $T$. Then, applying the definition of throughput, for any $\lambda \in [0, 1]$, the matrix $C = \lambda C_1 + (1-\lambda)C_2$ will achieve a throughput of $\theta(C, T) \geq \lambda\theta(C_1, T) + (1-\lambda)\theta(C_2, T)$. This can be seen by using the paths from $C_1$ for a fraction $\lambda$ of the traffic, and the paths from $C_2$ for a fraction $1 - \lambda$. As a consequence, we also have $\theta(C) \geq \lambda\theta(C_1) + (1 - \lambda)\theta(C_2)$. This leads to the following proposition.

**Proposition 2** *The guaranteed throughput function $\theta(C)$ is concave in $C$.*

### Linear Scaling

Given any positive $\lambda$, we can find a feasible rate allocation for $\lambda C$ from the rate allocation for $C$ (and vice versa) by scaling the rate assigned to each path by a factor $\lambda$ (respectively by $\frac{1}{\lambda}$). Therefore, we get the following proposition:

**Proposition 3** *The guaranteed throughput function $\theta$ is linear with respect to scaling, i.e.,*

$$\theta(\lambda \cdot C) = \lambda \cdot \theta(C).$$

### Strictly Increasing

Clearly $\theta$ is a non-decreasing function in the space of admissible capacity matrices. In other words, having more capacity cannot decrease the throughput. If $C$ and $D$ are two admissible capacity matrices, where $C \leq D$ (i.e., for all $i, j$, $C_{ij} \leq D_{ij}$, defining a partial order relation), then from the definition of $\theta$: $\theta(C) \leq \theta(D)$.

Now, if $D > C$, there exists $\epsilon$ such that

$$D \geq C + \epsilon C_{\text{uniform}},$$

where $C_{\text{uniform}}$ is the capacity matrix of the uniform mesh. Hence

$$\begin{aligned}
\theta(D) &\overset{(a)}{\geq} \theta((1+\epsilon)(\frac{1}{1+\epsilon}C + \frac{\epsilon}{1+\epsilon}C_{\text{uniform}})) \\
&\overset{(b)}{=} (1+\epsilon) \times \theta(\frac{1}{1+\epsilon}C + \frac{\epsilon}{1+\epsilon}C_{\text{uniform}})) \\
&\overset{(c)}{\geq} (1+\epsilon)(\frac{1}{1+\epsilon}\theta(C) + \frac{\epsilon}{1+\epsilon}\theta(C_{\text{uniform}})) \\
&\overset{(d)}{=} (1+\epsilon)(\frac{1}{1+\epsilon}\theta(C) + \frac{\epsilon}{1+\epsilon}\frac{1}{2})) \\
&> \theta(C),
\end{aligned}$$

where (a) uses the fact that $\theta$ is non-decreasing, (b) uses the equality $\theta(\lambda \cdot C) = \lambda\theta(C)$, (c) uses the concavity of $\theta$ and (d) uses the value of $\theta(C_{\text{uniform}})$. Therefore, we obtain:

**Proposition 4** *The guaranteed throughput function $\theta$ is strictly increasing, i.e., if $C < D$ then $\theta(C) < \theta(D)$.*

### 2.2.5 The Biased Mesh

#### Definition

We have already seen that the uniform mesh has a throughput of 50%, even though a node potentially spreads traffic over the useless links to itself. We can therefore expect a modified mesh — i.e., a mesh that does not spread traffic to itself — to have higher throughput. This is indeed the case; in fact, it is the network with the highest guaranteed throughput.

In this modified mesh, a link from a node to itself is only used to send traffic directly, and not for spreading. However, a link from a node to another one is used for sending traffic directly as well as for spreading. Therefore, intuitively, a link from a node to another one should have twice as much capacity as a link from a node to itself, because it will be used for two functions instead of one. We will call such a modified mesh the *biased mesh*. Its capacity matrix $\hat{C}$ is given by

$$\hat{C} = \begin{pmatrix} c & 2c & \ldots & \ldots & 2c \\ 2c & c & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & c & 2c \\ 2c & \ldots & \ldots & 2c & c \end{pmatrix},$$

where $c = 1/(2N-1)$.

In the remainder (Propositions 6, 7 and 8), we will show that $\hat{C}$ uniquely achieves the highest guaranteed throughput, using three consecutive steps. First, we will show that $\hat{C}$ achieves a throughput of $N/(2N-1)$. Then, we will prove that this is the largest achievable throughput for any network. Finally, we will demonstrate that the biased mesh is the only network to achieve this throughput.

### Guaranteed Throughput of the Biased Mesh

Our first objective is to show that the guaranteed throughput of the biased mesh with the capacity matrix $\hat{C}$ is at least $N/(2N-1)$. Using the definition of the guaranteed throughput, we need to consider all admissible rate matrices $T$. The following proposition significantly restricts the number of rate matrices $T$ we need to consider.

**Proposition 5** *The guaranteed throughput $\theta(C)$ defined in (2.3) can be found by considering the set of permutation matrices, i.e.,*

$$\theta(C) = \min_{T\ \textit{permutation}} (\theta(C,T)). \tag{2.7}$$

*Proof:* For any admissible matrix $T$, there is at least one doubly stochastic matrix $\overline{T}$ such that $T \leq \overline{T}$ [19, 85]. Clearly $\theta(C,\overline{T}) \leq \theta(C,T)$, and so we only need to consider the doubly stochastic rate matrices.

Birkhoff's theorem states that the set of doubly stochastic matrices equals the convex hull of the permutation matrices [12]. The claimed result follows from the definition of throughput. ■

Proposition 5 restricts to the set of permutation matrices the set of rate matrices we need to consider. To show that the throughput of $\hat{C}$ is at least $N/(2N-1)$, we just need to show that a throughput of $N/(2N-1)$ can be achieved for all the permutation matrices. It leads to the following proposition.

**Proposition 6** *The guaranteed throughput of the biased mesh with capacity matrix $\hat{C}$ is at least $N/(2N-1)$.*

*Proof:* We will prove that $\hat{C}$ achieves a throughput of $N/(2N-1)$ when $T = \sigma$, with $\sigma$ a permutation. Let $c = 1/(2N-1)$. We consider a node $i$, and prove that $i$ can always send at rate $Nc$ to $\sigma(i)$. Our objective is to send as much flow as we can directly, and to uniformly load-balance the remainder among the non-diagonal elements. We distinguish two cases: either $\sigma(i) = i$ or $\sigma(i) \neq i$.

If $\sigma(i) = i$, node $i$ needs to send $Nc$ to itself. Therefore, node $i$ can send $c$ directly to itself, and load-balance the remaining rate of $(N-1)c$ among the other $(N-1)$ nodes, then sending $c$ to each node.

If $\sigma(i) \neq i$, node $i$ needs to send $Nc$ to node $\sigma(i) \neq i$. Therefore, node $i$ can send $2c$ directly to $\sigma(i)$, and load-balance the remaining rate of $(N-2)c$ among the $(N-2)$ nodes different from $i$ and $\sigma(i)$; and each such node then sends $c$ again to node $\sigma(i)$.

Let us examine the load on each link. Each diagonal element $\hat{C}_{ii}$ only receives traffic if it is destined from node $i$ to node $i$, and in this case it receives exactly $c$, its capacity.

Moreover, each non-diagonal element $\hat{C}_{ij}$ can only receive traffic in two distinct cases, which cannot happen at the same time. If $j = \sigma(i)$, $\hat{C}_{ij}$ receives exactly $2c$, its capacity. Otherwise $j \neq \sigma(i)$, and $\hat{C}_{ij}$ receives $c$ from the load-balanced path $i \to j \to \sigma(i)$, and $c$ from the load-balanced path $\sigma^{-1}(j) \to i \to j$, summing to $2c$, its capacity.

The load on each link is therefore always bounded by its capacity; hence, this solution is feasible and the guaranteed throughput of $\hat{C}$ is at least $Nc = N/(2N-1)$.

■

### 2.2.6 Optimality of the Biased Mesh

We have just found that the biased mesh guarantees a throughput of at least $\frac{N}{2N-1}$. The following proposition shows that the biased mesh achieves the maximum possible guaranteed throughput for any admissible capacity matrix.

**Proposition 7** *If the capacity matrix $C$ is admissible, then the guaranteed throughput $\theta(C) \leq \frac{N}{2N-1}$.*

The proof for Proposition 7 is in Appendix A.

### 2.2.7 Uniqueness of the Optimal Capacity Matrix

Since we proved that the biased mesh achieves the optimal throughput $N/(2N-1)$, we will now demonstrate that it is the only capacity matrix to do so. This is done in Proposition 8, proved in Appendix B.

**Proposition 8** *The only capacity matrix $C$ that can achieve the optimal throughput $N/(2N-1)$ is the capacity matrix $\hat{C}$ of the biased mesh.*

In conjunction with Propositions 6, 7 and 8, we have, therefore, established the following theorem.

**Theorem 9** *The biased mesh satisfies the following three properties:*

*(i) The guaranteed throughput of the biased mesh is equal to $\hat{\theta} = N/(2N-1)$.*

*(ii) The biased mesh achieves the maximum possible guaranteed throughput for any network, i.e., $\theta(\hat{C}) = N/(2N-1)$.*

*(iii) The biased mesh is the only network to achieve this guaranteed throughput, i.e., $\theta(C') < \theta(\hat{C})$ for any admissible capacity matrix $C' \neq \hat{C}$.*

### 2.2.8 Conclusions and Intuition

Theorem 9 shows that the biased mesh architecture achieves the best throughput among all possible architectures, and is the only one to do so. Therefore, it performs better than many alternative architectures, including the ring, torus and hypercube architectures.

However, this result assumes that if packets arrive at their destination after a single hop, they can immediately leave the switch. Therefore, the architecture will need to allow for one-hop as well as two-hop paths through the switch. This might prove difficult to implement, and might be a reason for a designer to prefer the uniform mesh. Using the fact that the uniform mesh achieves 50% throughput (Equation 2.5), we know that the uniform mesh is

$$\frac{\theta(\hat{C})}{\theta(C_{\text{uniform}})} = \frac{\frac{N}{2N-1}}{\frac{1}{2}} = \frac{1}{1-\frac{1}{2N}} = 1 + o(1) \text{ — optimal} \tag{2.8}$$

for its guaranteed throughput. Therefore, the load-balanced router with a uniform mesh is *asymptotically optimal*. Asymptotically with $N$, it guarantees at least as much throughput as any other fixed interconnection with an admissible capacity matrix. This is an additional argument in favor of the uniform mesh.

# Chapter 3

# Packet Reordering

Chapters 1 and 2 introduced the load-balanced router and illustrated how it can be practically implemented using optics. In this chapter, we will see how the load-balanced router can reorder packets. Packet reordering is a widespread property among load-balanced systems and can be detrimental to Internet traffic. Therefore, we will provide and analyze different possible algorithms to prevent packet reordering. We will finally focus on the FOFF algorithm, and prove that it prevents reordering while providing throughput and delay guarantees.

## 3.1 Presentation of Packet Reordering

### 3.1.1 Example of Reordering in the Load-Balanced Router

We first define packet reordering. As defined in the Introduction, a flow is the set of all packets having the same input and output destination. Packet reordering occurs in a router when packets from a same flow depart from the router in an order different from the one in which they arrived.

Packet reordering could occur in a load-balanced router. This is because the load-balancer spreads packets as they arrive without regard to their final destination or departure time. Therefore, packets from the same flow can take different paths of different delays within the router. Consequently, as illustrated below, this load-balancing among paths of different delays could incur packet reordering.

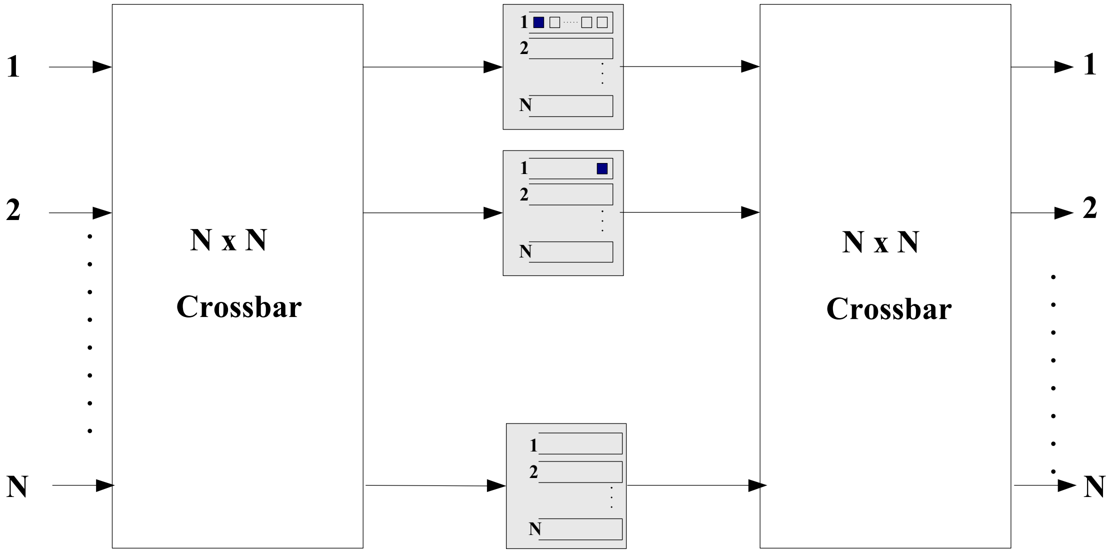


Figure 3.1: Example of packet reordering in a load-balanced router.

Figure 3.1 illustrates this possibility. In this example, all packets considered are destined to the output 1. By definition, in the first $N$ time-slots, each input is connected to the first intermediate input exactly once. Assume that each input except the first one receives a packet and transfers it to the first intermediate input when they are connected together. After $N$ time-slots, the first intermediate input will have received $N-1$ packets from inputs $2, ..., N$, and only one of these packets will have been serviced. Therefore, $N-2$ packets are left in the first intermediate input (they are represented as transparent in Figure 3.1). Assume that the first input then receives two packets back-to-back when it is successively connected to intermediate inputs 1 and 2. Therefore, the first input consecutively transfers these two packets to intermediate inputs 1 and 2 (packets from input 1 are represented as filled in Figure 3.1). The second packet is alone in its queue, while the first one has $N-2$ packets in front of it. Therefore, the second packet is serviced earlier. It arrives at output 1 earlier, and leaves the router earlier. Consequently, the two packets are *reordered*. In other words, they belong to the same flow, and the order in which they arrive at the input is different from the order in which they are leaving the output.

The example shows that reordering occurs when VOQs in different intermediate inputs and destined to the same output have different lengths. Due to the fixed-rate service, the amount of reordering depends on the difference between the lengths of the VOQs. For instance, assume that VOQs are modeled as having an infinite buffer capacity. Then, when a VOQ length increases while the other stays constant, the VOQ length difference grows unbounded, and therefore the possible amount of reordering also grows unbounded. Therefore, the load-balanced router does not provide any guarantee on the amount of reordering.

### 3.1.2 Consequences of Packet Reordering for Internet Traffic

The load-balanced router exhibits reordering — but is reordering really a problem? RFC 1812, the most common standard defining router requirements, does not forbid reordering in routers [8]. In addition, reordering is not uncommon in the Internet [10, 49]. Therefore, we might believe that reordering is not necessarily a problem.

However, in its current version, TCP (Transmission Control Protocol) does not perform well when out-of-order packets arrive at their destination. Out-of-order packets can be perceived as loss indicators, and trigger unnecessary retransmissions and TCP timeouts [13]. These retransmissions and timeouts cause a decrease in TCP throughput and an increase in packet delay. Consequently, since TCP traffic constitutes the vast majority of Internet traffic [35, 37], network operators generally insist that routers do not reorder packets belonging to the same application flow, i.e., sharing the same (source, destination) pair. Our goal in this chapter is to provide this guarantee. In particular, if we assume that all packets from an application flow take the same path in the Internet and therefore belong to the same router flow in the router, it is sufficient to guarantee that the router does not reorder packets belonging to the same router flow.

### 3.1.3 Preventing Reordering

There are two methods of preventing packet reordering. The first method consists in *bounding the amount of reordering* and using a finite reordering buffer at the output.

The second method is to make sure that *packets arrive in order* to the output, thus keeping packets in order throughout the router.

We saw in the example above that reordering can occur when VOQ lengths are different, and that the amount of reordering typically increases as the difference of VOQ lengths increases. Therefore, most of the algorithms that prevent reordering will try to bound or prevent any VOQ length difference. To do so, these algorithms will typically rely on a small input-stage coordination buffer, which spreads packets uniformly among the intermediate inputs.

In [21], the authors propose two schemes based on the first method of bounding reordering. Both schemes rely on algorithms found in the PPS (Parallel Packet Switch) router [45] and use a small input-stage coordination buffer. The first scheme, called FCFS (First Come First Served), uses a jitter control buffer in each intermediate input to ensure proper ordering of the traffic leaving the intermediate inputs. The second scheme, EDF (Earliest Deadline First), schedules packets according to their departure times in an ideal (output-buffered) router. However, both schemes do not seem practical. The jitter control mechanism in FCFS might require up to $N$ memory-write accesses per time slot. And EDF needs to retrieve the packet with the smallest time stamp from a queue, making it hard to implement in a high performance router.

The second method of preventing reordering, which keeps packets ordered throughout the router, is used in [54] and [22]. [54] presents an algorithm that uses a coordination buffer in the input stage, and then queues packets in VOQs in the intermediate inputs based on their input, intermediate input, and output. Using this fine-grained queueing, the algorithm guarantees that packets arrive at the outputs in order. However, this algorithm requires a more complex queueing management system, and communication of state information between the intermediate inputs and the outputs. In [22], the authors introduce an algorithm in which the buffers in the intermediate inputs are finite and packets are guaranteed to leave the router in order. However, the algorithm assumes that in each frame period, arrivals of packets destined to a given output are constrained. While this might be satisfied in frame-based traffic such as SONET (Synchronous Optical Network) traffic, this property is not satisfied in general in the Internet.

Therefore, all these algorithms present significant problems in router implementations. Our objective is to find a different scheme in order to guarantee that packets leave the router in order. We will first consider Application Flow-Based Routing (AFBR) and Uniform Frame Spreading (UFS), two algorithms that attempt to maintain packets ordered throughout the router. Then, we will present Full Ordered Frames First (FOFF), an algorithm that bounds reordering, and we will prove that FOFF also provides throughput and delay guarantees.

## 3.2 Application Flow-Based Routing (AFBR)

### 3.2.1 How AFBR Works

We saw above that reordering is due to the uncontrolled use of parallelism in the load-balanced router. Reordering occurs when packets from the same router flow, and in particular from the same application flow, take different paths with different delays. Therefore, a simple idea is to make all packets from the same application flow take the same path inside the router. We will call this an *Application Flow-Based Routing* (AFBR).

AFBR can be implemented by hashing packet header fields (e.g., source and destination IP addresses and protocol identification) into $N$ different hash values. Packets from the same application flow will return the same hash value. Then, all packets having the same hash value will be transferred by a given input to the same intermediate input. Therefore, all packets from the same application flow take the same path inside the router. Consequently, AFBR prevents reordering throughout the router for packets from the same application flow.

Note that hashing flows is common in load-balanced systems. It is used for example in backbone links [15], address lookups [14, 48], packet processing [16], web servers [70, 71], network processors [32, 51] and flow demultiplexing [33].

### 3.2.2 Properties of AFBR

AFBR is more efficient when there is a guarantee that the amount of traffic corresponding to each (input, intermediate input) pair and each (intermediate input, output) pair does not exceed a rate of $R/N$, especially over short periods of time. As a consequence, the rate between each input and each intermediate input will not exceed the link capacity $R/N$, and the rate between each intermediate input and each output will also not exceed the link capacity $R/N$. In this case, AFBR provides an easy implementation that prevents packet reordering throughout the router.

However, AFBR is more difficult to implement than at first glance. Without traffic guarantees over each frame, as in SONET traffic, the coordination buffer size at the inputs might grow unbounded. Moreover, if all traffic consists of a single application flow, AFBR will not be able to provide any non-trivial throughput guarantee (it will only provide a throughput of $R/N$). AFBR also assumes a fixed hashing scheme, and loses reordering guarantees if hashing becomes dynamic. Finally, by having all packets from the same application flow take the same path inside the router, AFBR might be more prone to single-point failures for the end applications.

## 3.3 Uniform Frame Spreading (UFS)

### 3.3.1 Presentation of UFS

In the basic load-balanced router, reordering occurs when two VOQs destined to the same output in different internal inputs have different lengths. The objective of the *Uniform Frame Spreading* (UFS) algorithm is to avoid reordering throughout the router by assigning the same number of packets to all the VOQs destined to the same output.

In UFS, at each input, there are $N$ FIFO (First In First Out) queues — one per output. Arriving fixed-size packets destined to a given output are buffered in the corresponding queue, until there are $N$ of them. A group of $N$ such packets from a given flow is called *a full frame*. When a frame is full, it is spread uniformly across the intermediate inputs, starting with intermediate input 1, and ending with

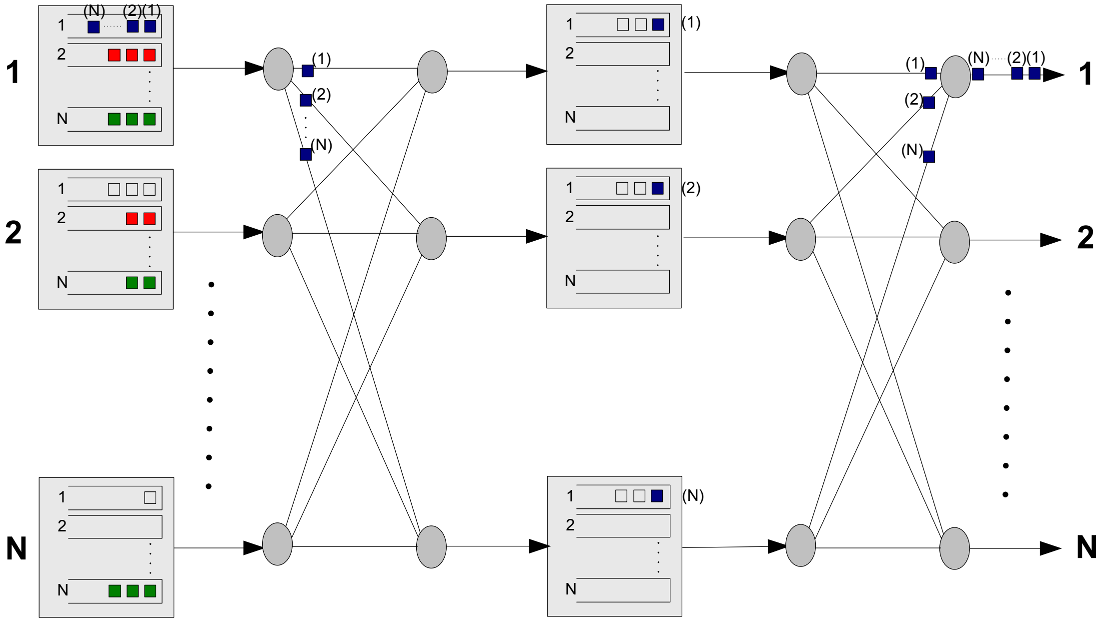


Figure 3.2: Illustration of the UFS algorithm.

intermediate input $N$. When each packet arrives at the intermediate input, it is immediately directed to the appropriate VOQ that the packet is destined to. All packets from the same full frame then consecutively become head-of-line of their respective VOQs and are transferred in order to the output.

Figure 3.2 illustrates the successive steps in the UFS algorithm. In order to explain the intuition behind UFS, we will only consider packets destined to output 1. In this example, we will also assume for simplicity that up to $N$ packets can leave and arrive simultaneously in the mesh structure, even though this is not needed in practice. First, packets arriving at the input are queued in VOQs arranged by outputs, until at least one full frame of $N$ packets is formed. Full frames leave the inputs in order, and the $j$-th packet of each full frame is transferred to the $j$-th intermediate input. The $N$ packets of the same full frame are then queued simultaneously in the intermediate inputs. For instance, Figure 3.2 illustrates three buffered full frames, the first from the first input (filled packets), and the two others from other inputs (transparent packets). Since all packets from a given full frame arrive simultaneously, each VOQ size is the same, and therefore all packets also depart simultaneously. Once the full frame leaves the intermediate inputs, it is transferred to the correct output, from which it then departs in order.

It is interesting to note that all the VOQs destined to the same output behave in the same way, and each packet of the same full frame sees the same behavior. As a consequence, each linecard knows the state of all other intermediate inputs, since all intermediate inputs have the same state. In addition, we can model the behavior of the whole load-balanced router by considering only any given intermediate input. Finally, since each full frame is written as well as read in parallel, this algorithm can be seen as an instance of *bit-slicing*. Therefore, the whole set of $N$ intermediate inputs looks like a single shared-memory pool.

### 3.3.2 Advantages of UFS

We have shown above how the UFS algorithm maintains packets ordered throughout the router. UFS presents the following benefits:

- *UFS keeps packets in order throughout the router.*

  Packets of the same full frame are kept in order because they are transmitted consecutively to the intermediate inputs and received consecutively by the outputs.

- *Linecards can operate independently.*

  The UFS algorithm is distributed and can operate independently in each input, intermediate input and output. It needs no additional communication of information among linecards.

- *No pathological traffic patterns.*

  The 100% throughput proof for the basic architecture relies on the traffic being weakly mixing. While this might be a reasonable assumption for heavily aggregated backbone traffic, it is not guaranteed. In fact, it is easy to create a periodic adversarial traffic pattern that inverts the spreading sequence, and causes packets for one output to pile up at the same intermediate linecard. This can lead to a throughput of only $R/N$ for each linecard.

UFS prevents such pathological traffic patterns. UFS provably always has the same throughput as an ideal output-queued router, irrespective of the arrival process. (An *output-queued router* is an ideal router in which each output contains the packets destined to it. Each output services packets as long as there is as at least one packet to service, a property known as *work-conserving*. No router can have a better throughput or average delay than an output-queued router.) This result is proved in Theorem 21 of Appendix C.

- *VOQ states are known by all linecards.*

  In order to know the amount of congestion in the router (for example to implement a drop-policy), we need to know how many packets are in each VOQ. But because each VOQ has the same occupancy (one packet from each full frame), it is sufficient to look at just one linecard.

### 3.3.3 Filling a Frame

A significant disadvantage of the UFS algorithm is that it requires a frame to contain $N$ packets. If a frame contains fewer than $N$ packets, one can choose whether to send it or not. If the frame is sent with less than $N$ packets, it needs to insert empty idle packets in order to become a full frame, and thus incurs a loss of throughput. However, if the frame is kept in the input and waits for additional packets, it can incur a significant delay if not starvation. Therefore, sending the frame reduces throughput, while waiting for additional packets increases delay and potentially creates starvation. There is a clear dilemma: send or wait? Is there even something to wait for?[1]

There are a few ways of dealing with such a problem, involving a trade-off between throughput, starvation and practicality. A first approach is to wait for frames to fill, and hence keep the throughput and average-delay guarantees, while possibly starving some flows. A second possibility is to make packets much smaller, for instance

[1] *"Waiting for Godot"* [9] provides a good analogy to the problem.

$1/N$-th of the original packet size. This solution would be equivalent to bit-slicing. However, although it would solve the frame-filling problem, the granularity of the resulting packets might cause problems in the intermediate inputs. In particular, smaller chunks might translate into a higher chunk arrival frequency and exceed buffering speed limits. In addition, the overhead caused by headers and control mechanisms will increase. A third solution is to use a timeout mechanism in the inputs. The timeout will trigger the frame to be completed with empty idle packets and forwarded to the intermediate inputs. As noted in [50], this can be arranged to prevent an excessive reduction in throughput. However, a low timeout value will cause a significant loss of throughput. Conversely, a negligible loss in throughput guarantee would need a significant timeout value. A fourth possible solution would be to change timeout values based on state information. As seen above, each linecard knows when each output is congested, and therefore can change the timeout value depending on the congestion state (using a smaller timeout when there is no congestion and therefore no throughput capacity constraints). However, while interesting practically, such an approach does not necessarily provide any better theoretical throughput guarantee. Therefore, none of these schemes appears satisfactory.

## 3.4 Full Ordered Frames First (FOFF)

### 3.4.1 Presentation of FOFF

AFBR and UFS prevent packets from being reordered throughout the router by precisely regulating packet transfer times. However, this strict regulation is not flexible enough to adapt to changing traffic conditions, and results in a loss of throughput and possible starvation. Therefore, the following scheme tries a different approach. The Full Ordered Frames First (FOFF) algorithm allows some bounded reordering inside the router, and relies on a reordering buffer at the output.

FOFF runs independently on each linecard using information locally available. As in UFS, each input linecard keeps a separate FIFO queue for each output. When a packet arrives, it is placed at the tail of the queue corresponding to its eventual output. Ideally, FOFF behaves as UFS, serving a queue only when it contains at least one full frame. However, contrary to the UFS algorithm, FOFF allows for a non-empty queue to send packets when no queue has any full frame. Therefore, FOFF avoids starvation but allows for different VOQ sizes and reordering.

### 3.4.2 Implementation

We will now define how the FOFF algorithm works in each input, intermediate input and output.

#### Input Implementation

In each input, FOFF will send full frames whenever possible, by giving a higher priority to full frames. However, when there are no full frames available, FOFF will service the remaining packets, by picking non empty queues in a round-robin order.

In particular, in each input, FOFF uses at most $N^2$ packets arranged in $N$ FIFO queues labelled $Q_1 \dots Q_N$. FOFF operates as follows:

1. An arriving packet destined to output $j$ is placed in $Q_j$.

2. Every $N$ time-slots, the input selects a queue to serve for the next $N$ time-slots. First, it picks round-robin from among the queues holding more than $N$ packets, i.e., holding at least one full frame. If there are no such outputs, then it picks round-robin from among the non-empty queues. Up to $N$ packets from the same queue (and hence destined to the same output) are transferred to different intermediate linecards in the next $N$ time-slots. For each flow, a pointer keeps track of the last intermediate linecard to which a packet was transferred, and the next packet is always sent to the next intermediate linecard.

Clearly, if there is always at least one full frame, FOFF behaves as UFS and there is no reordering. Reordering occurs only when no queue has a full frame; but as shown in Theorem 25 (Appendix D), the amount of reordering is bounded, and is corrected in the output with a fixed length reordering buffer.

### Intermediate Input Implementation

FOFF uses $N$ VOQs per intermediate input, and behaves exactly like the basic load-balanced router.

### Output Implementation

In each output, FOFF services packets in order. In order to do so, for each flow, FOFF keeps track of the last intermediate input from which the last ordered packet arrived. This enables FOFF to know that the next ordered packet will come from the next intermediate input, because packets from each flow are spread in round-robin order among intermediate inputs. Therefore, FOFF can send packets in order from each output without using sequence numbers, without exchanging state information between linecards, and without knowing which packets belong to a full frame.

More precisely, in each output, FOFF uses the following structures:

- $N$ FIFO queues corresponding to the $N$ intermediate inputs. When a packet arrives from an intermediate input, it is buffered in the queue corresponding to this intermediate input. As proved in Theorem 25 (Appendix D), these queues contain a total of at most $N^2 + 1$ packets (including the one being serviced).

  We will call *head of flow* the first packet of a given router flow that has not yet left the router, and *head of line* the first packet of a given queue. A packet can leave in order if it is both head of flow and head of line of one of these $N$ queues.

- A pointer vector of $N$ pointers corresponding to the $N$ inputs, indicating in what queue to expect the next in-order packet of each flow. These pointers are only incremented cyclically (modulo $N$), since packets are spread among intermediate inputs in a round-robin order.

- A pointer queue of maximum size $N$, indicating from which FIFO queues a packet is ready to depart in order.

At each time-slot and in each output, FOFF implements the steps defined below. In these steps, FOFF looks for packets that are both head of flow and head of line packets, since these are the packets that can leave in order. Let us examine how a packet can become head of flow *and* head of line. If it is head of flow, it can become head of line either because there is no packet in its queue when it arrives (case 1 in the algorithm), or because it is the second packet in its queue and the head of line departs (case 2a in the algorithm). Otherwise, if it is not head of flow but only head of line, its head of flow needs to leave (case 2b in the algorithm). Therefore, the algorithm below ensures that the output is work-conserving whenever there is at least one packet that is both head of line and head of flow, and such a packet always exists whenever the reordering buffer contains $N^2 + 1$ packets (Theorem 25, Appendix D). Note that there is a bounded number of steps at each time-slot, independently of $N$. Also note that the algorithm at each output operates independently of other outputs.

1. *Arrival:* When a packet arrives at the output, it is first buffered in the queue corresponding to its intermediate input. Assume the packet comes from input $i$ and intermediate input $j$. If queue $j$ in the output is empty upon the arrival of the packet, and the pointer for $i$ designates queue $j$, the packet is both head of flow and head of line. It is ready to depart in order. Therefore FOFF adds a pointer to queue $j$ to the pointer queue, and increments (modulo $N$) the pointer for input $i$.

2. *Departure:* At each time-slot, if the pointer queue is not empty, its first pointer designates some queue $j$ from which the first packet can depart in order. Assume this packet comes from input $i$. FOFF services this first packet, increments (modulo $N$) the pointer for input $i$, and removes the pointer from the pointer queue. Then, FOFF examines the two following packets, and adds them to the pointer queue if needed:

    (a) First, FOFF examines the new head-of-line packet of queue $j$, coming from some input $i'$. If the pointer for $i'$ points to $j$, then this packet is both head of line and head of flow. Therefore FOFF adds it to the pointer queue and increments (modulo $N$) the pointer for input $i'$.

(b) Then, FOFF also examines the head-of-line packet of the queue designated by the pointer for input $i$, which is just $j+1$ (modulo $N$). If this head-of-line packet indeed comes from input $i$, then it is both head of line and head of flow. Therefore, FOFF adds it to the pointer queue and increments (modulo $N$) the pointer for input $i$.

As explained above, these steps ensure that FOFF will service packets in order. We will now show that FOFF can also provide throughput and delay guarantees.

### 3.4.3 Properties of FOFF

FOFF has the following properties, which are proved in Appendix D.

- *Packets leave the router in order.*

  FOFF bounds the amount of reordering inside the router, and requires a reordering buffer that holds at most $N^2+1$ packets (proof in Appendix D).

- *FOFF is practical to implement.*

  As explained above, FOFF has an $O(1)$ complexity, since it only needs a bounded number of steps per time-slots, independently of $N$. In addition, FOFF is distributed and operates independently in each input, intermediate input and output. It needs no additional communication of information among linecards.

- *No pathological traffic patterns.*

  Like UFS, FOFF manages to avoid pathological traffic patterns that can reduce the throughput in the basic load-balanced router. It does so by spreading each flow evenly across the intermediate linecards. FOFF guarantees that the cumulative number of packets sent to each intermediate linecard for a given flow differs by at most one. This even spreading prevents a traffic pattern from concentrating packets to any individual intermediate linecard. As a result, FOFF guarantees 100% throughput for *any* arriving traffic pattern; there are provably no adversarial traffic patterns that reduce throughput, and the router has the same throughput as an ideal output-queued router.

In fact, Appendix E proves an even stronger result: the average packet delay through the router is within a constant from that of an ideal output-queued router. This result is valid for any adversarial infinite traffic, and therefore independent of the traffic pattern and the arrival rate. Of course, a packet might experience a very long — or infinite — delay using FOFF. For instance, this occurs at the input because full frames have higher priority than non-full frames; therefore, a persistent flow with continuous full frames can prevent a packet from leaving the input. However, Appendix E proves that the number of such packets is bounded for any adversarial traffic. In fact, under any arrival traffic, the number of packets that leave the load-balanced router is within a constant of the number of packets that would leave an ideal output-queued router.

- *FOFF is practical to implement.* Each input, each intermediate input and each output require $N$ queues. Each input and each output hold at most $N^2 + 1$ packets per linecard (the intermediate inputs hold the congestion buffer, and their memory size is determined by the same factors as in a shared-memory work-conserving router). The FOFF scheme is decentralized, uses only local information, and does not require complex scheduling.

- *Priorities in FOFF are practical to implement.* It is simple to extend FOFF to support $k$ priorities with $k \cdot N$ queues in each input, intermediate input and output. These queues could be used to distinguish different service levels or could correspond to sub-ports.

These advantages of FOFF make it more practical to implement than the schemes listed above. It will be the algorithm chosen in the implementations described in the next chapter.

# Chapter 4

# Implementation of the Load-Balanced Router Using Optics

In this chapter, we will design and architect a high-performance router. We will use the example of a 100Tb/s Internet router, arranged as 640 linecards operating at 160Gb/s. First, we will show that the simple mesh architecture introduced in Chapter 2 does not scale to such a high number of linecards. Therefore, we will introduce a hierarchical mesh architecture to allow for scaling. Second, we will illustrate why a hierarchical mesh architecture is not able to operate correctly when linecards are introduced incrementally or are removed, as might happen in practice. Therefore, we will introduce a new architecture based on electronic linecards and an optical switch fabric with static MEMS. We will prove that this architecture operates without packet conflicts. Last, we will demonstrate why this architecture is a practical choice today for scaling routers while providing throughput and fault-tolerance guarantees.

## 4.1 Architecture Requirements

### 4.1.1 A 100Tb/s Router Example

The load-balanced router seems to be an appealing architecture for scalable routers that need performance guarantees. In what follows we will study the architecture in more detail. To focus our study, we will assume that we are designing a 100Tb/s

Internet router that implements the requirements of RFC 1812 [8]. The router will be arranged as 640 linecards operating at 160Gb/s (OC-3072). We pick 100Tb/s because it is challenging to design, is probably beyond the reach of a purely electronic implementation, but seems possible with optical links between racks of distributed linecards and switch fabrics. It is roughly two orders of magnitude larger than Internet routers currently deployed, and seems feasible to build with technology available today. We pick 160Gb/s for each linecard because 40Gb/s linecards are feasible now, and 160Gb/s is the next logical generation.

### 4.1.2 Architecture Requirements

Our objective is to design an architecture for implementing a 100Tb/s router, arranged as 640 linecards operating at 160Gb/s. We will require this architecture to provide a 100% throughput guarantee.

Since current centralized schedulers can hardly scale to this capacity while providing throughput guarantees, we will require the router to not have any centralized scheduler, and assume that the router has a load-balanced router architecture.

In addition, we require the router to avoid packet reordering while providing 100% throughput. Therefore, the router will implement the FOFF algorithm introduced in Chapter 3.

Finally, we want the router to operate correctly when populated with any number of linecards connected to any ports. This will enable us to add linecards incrementally, as well as provide fault-tolerance if a linecard fails.

### 4.1.3 Assumptions

We will assume that the router occupies multiple racks, with up to 16 linecards per rack. In terms of optical technology, we will assume that it is possible to multiplex and demultiplex 64 WDM (Wavelength Division Multiplexing) channels onto a single optical fiber. We will also assume that each of these WDM channels can operate at up to 10Gb/s.

## 4.2 The Hierarchical Mesh Architecture

### 4.2.1 Scaling the Number of Linecards

As seen in Chapter 2, it is possible to implement a load-balanced router with a single mesh of link capacity $2R/N$. This mesh can also be replaced by an AWGR or a round-robin crossbar, as long as each two linecards are connected at rate $2R/N$.

However, a mesh between $N$ linecards requires $N^2$ links, and therefore it does not scale easily with $N$. In practice, using $N^2$ optical fibers or electrical links is impractical or too expensive. For example, a 100Tb/s router with $N = 640$ linecards would need 409,600 links, each carrying data at $2R/N = 0.5$ Gbps.

We could use an AWGR instead of a mesh. This would reduce the number of fibers, and increase the data-rate carried by each. Instead of connecting to $N$ fibers, each linecard multiplexes onto one fiber $N$ WDM channels, each operating at $2R/N$. However, current state-of-the-art AWGRs hardly scale beyond one hundred inputs and outputs [11]. In addition, having 640 distinct WDM channels on the same fiber is beyond what is practical today.

We could also try to use a round-robin active switch fabric instead. For instance, we could use an electric crossbar, or an optical switch such as a MEMS switch [72], an electro-optic [66] or an electro-holographic waveguide [67]. However, we would need to reconfigure frequently such an active switch (each time a packet is transferred), and we would run into scalability issues, since this active device would have to connect up to 640 inputs and outputs.

Therefore, it does not seem practical to manufacture a 640-port switch from any of these technologies. And so, to scale the router to this number of linecards, we need to decompose the switch fabric into several smaller switch fabrics.

### 4.2.2 The Hierarchical Mesh

The load-balanced router needs a path to connect each input port to each output port at a fixed uniform rate. The switch fabric could be a uniform mesh, but this $640 \times 640$ uniform mesh would have too many ports to be practical. Therefore, we will

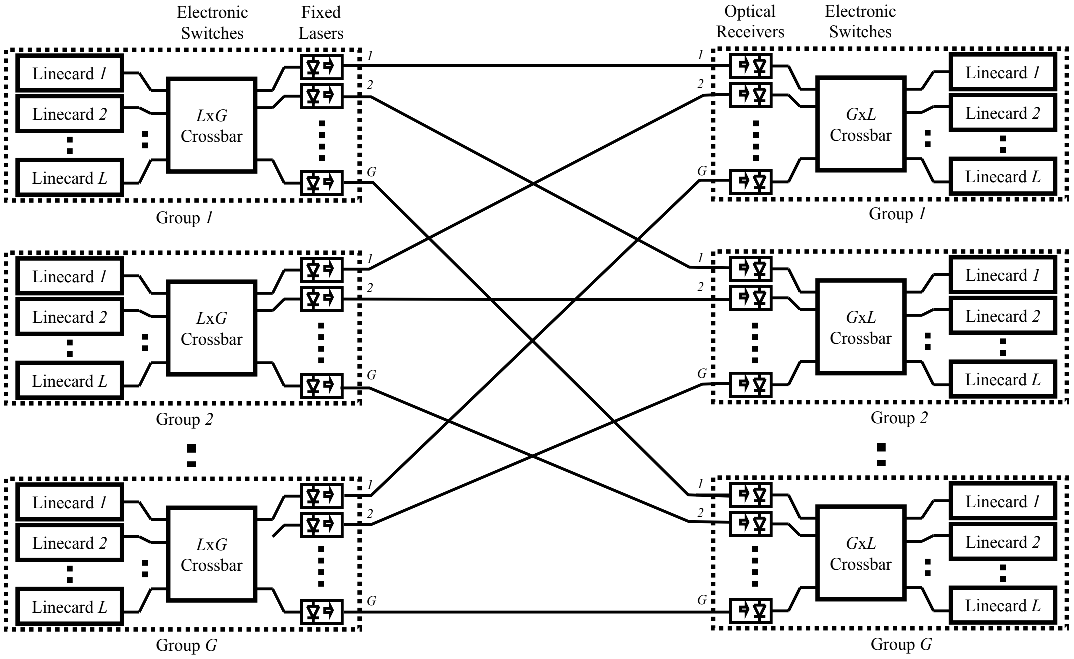


Figure 4.1: Hierarchical mesh architecture.

reduce this uniform mesh into a uniform mesh connecting a smaller number of ports of larger bandwidth. In order to do so, we will group together many linecards. Then, instead of establishing a mesh between linecards, we will establish a mesh between groups.

Figure 4.1 illustrates such a hierarchical approach, called the *hierarchical mesh architecture*. The $N$ linecards are grouped together into $G$ groups (or racks) of $L$ linecards each. A uniform mesh connects together the $G$ groups. Within each group, traffic from the linecards is multiplexed at the input side and demultiplexed at the output side, for instance with a round-robin crossbar. The hierarchical mesh uses electronics in the groups (racks), and could use optical links between the racks for the mesh. Therefore, it needs $G$ fixed lasers per input-side crossbar, and $G$ optical receivers per output-side crossbar. Since each transmitting group multiplexes traffic from $L$ linecards sending at rate $2R$ and spreads it among $G$ groups, the capacity of each link is $\frac{2RL}{G}$.

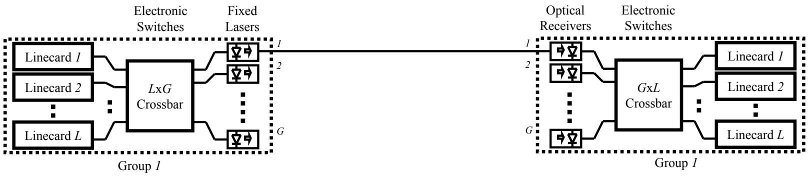


Figure 4.2: Hierarchical mesh architecture populated with only the first group.

## 4.3 The MEMS-Based Architecture

### 4.3.1 Mesh Decomposition as a Sum of Matches

The hierarchical mesh architecture satisfies all the requirements but one: it is not able to operate correctly when populated with any number of linecards that can be connected to any ports.

Assume for instance that the first group is populated with linecards, and all other groups are empty, as illustrated in Figure 4.2. This could occur for example when the router is populated incrementally, when multiple linecards need to be removed from the router, or when multiple linecards fail. All the traffic from the first transmitting group will be sent to the first receiving group, and spread uniformly across its receiving linecards. Therefore, the first transmitting group needs to send $2RL$ amount of traffic, because it multiplexes $L$ streams running at rate $2R$. However, the link capacity determined earlier is only $\frac{2RL}{G}$. Therefore, the capacity of the uniform mesh is not sufficient to switch the traffic sent by the first transmitting group. We need to find a way of providing enough capacity for the traffic.

The capacity of the hierarchical mesh is not sufficient to switch traffic because when groups become empty, the links destined to them are not used. Since the mesh is fixed, the capacity of those links cannot be reoriented towards the groups that do have linecards. Therefore, we need to be able to move those links when needed. We do this by replacing the mesh with $M$ matches between the transmitting groups and the receiving groups. Each match implements a one-to-one connection between the

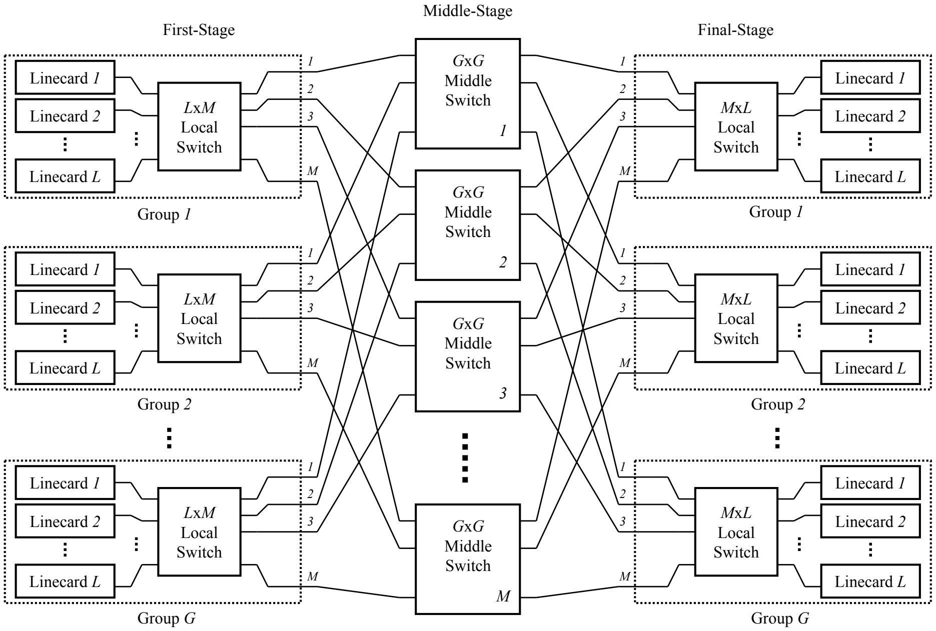
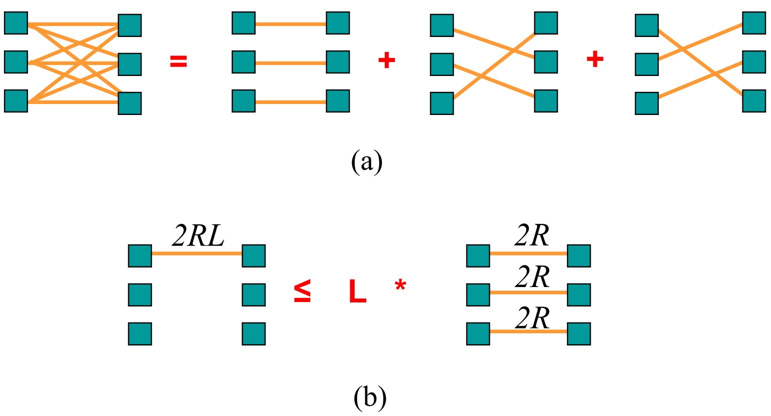


Figure 4.3: Partitioned switch fabric.

transmitting groups and the receiving groups. While the mesh is always fixed, the matches change when the linecard configuration changes, and remain fixed otherwise.

Figure 4.3 illustrates how the matches are implemented by using $M$ middle-stage non-blocking switches. As shown in the figure, each transmitting group in the input side multiplexes traffic from its linecards. It then spreads traffic across the receiving groups using the $M$ middle stages. Each of the $M$ middle stages implements one of the $M$ matches. Finally, the receiving groups in the output side demultiplex the incoming traffic and spread it uniformly among their linecards. We will later determine the value of $M$, prove that the $M$ matches provide sufficient capacity, and provide more implementation details.

As an example, assume that the mesh is uniform. Then we could use $G$ matches, as illustrated in Figure 4.4(a). The first match would be represented by the identity matrix. Implemented in the first middle-stage switch, it statically connects sending group 1 to receiving group 1, sending group 2 to receiving group 2, and so on. Each other match would rotate its configuration by one. For instance, the second match in



(a)

(b)

Figure 4.4: Decomposition of a mesh into matches, with (a) a uniform mesh having all groups, and (b) a mesh with a single group.

the second switch connects sending group 1 to receiving group 2, sending group 2 to receiving group 3, etc. The third match connects sending group 1 to receiving group 3, and so on. Therefore, the matches are the $G$ round-robin permutations, and their sum is the uniform matrix. In this case, $M = G$.

As another example, assume again that all groups but the first one fail. As shown in Figure 4.4(b), we can then use $L$ matches, each connecting the first transmitting group to the first receiving group at rate $2R$. This will then provide the needed capacity of $2RL$ between the transmitting group and the receiving group.

### 4.3.2 Using MEMS Switches

The MEMS-based switch fabric is a straightforward implementation of the design described above and is represented in Figure 4.5. As before, the architecture is arranged as $G$ groups of $L$ linecards. In the center, $M$ statically configured $G \times G$ MEMS switches interconnect the $G$ groups. MEMS switches are the optical equivalent of crossbars. MEMS switches use micromirrors to reflect optical beams from inputs to outputs and are independent of wavelength and data-rate. Typical reconfiguration

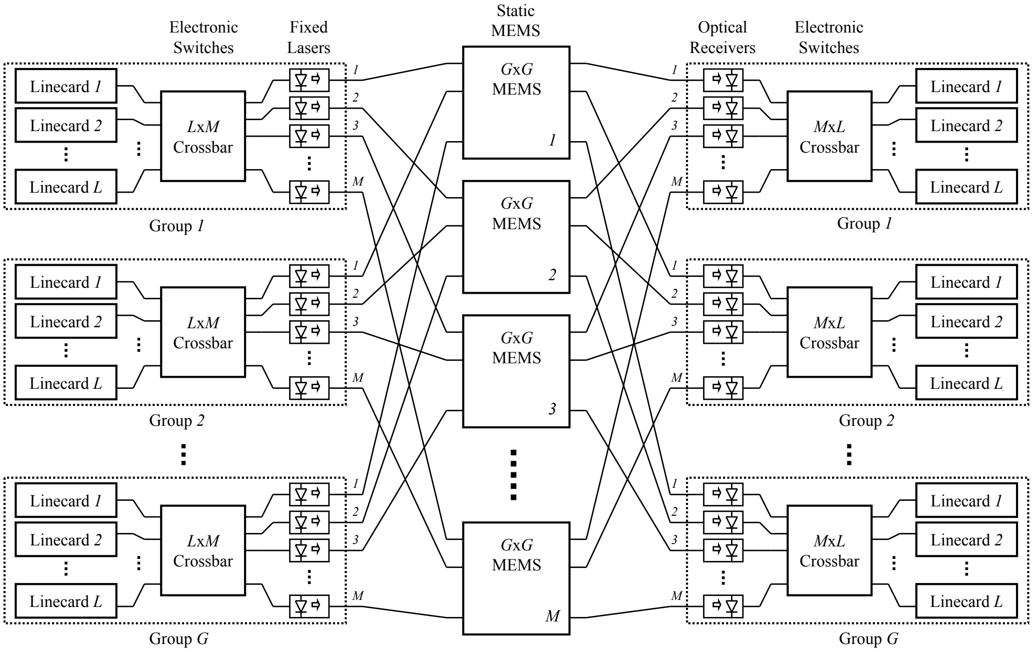


Figure 4.5: MEMS-based architecture.

times of MEMS switches are 1-10ms [66, 72]. Note that the MEMS switches are reconfigured only when a linecard is added or removed and provide the ability to create the needed paths to distribute the data to the linecards that are actually present. Each transmitting group of linecards spreads packets over the MEMS switches using an $L \times M$ electronic crossbar. Each output of the electronic crossbar is connected to a different MEMS switch over a dedicated fiber at a fixed wavelength. Packets from the MEMS switches are spread across the $L$ linecards in a group by an $M \times L$ electronic crossbar.

In the remainder of this chapter, we will assume that all the lines in Figure 4.5 have capacity $2R$. We will explain with more detail in Section 4.5 how to implement these line rates in practice.

## 4.4 Linecard Schedule

While we described the MEMS-based architecture, we did not specify how many MEMS switches are needed, nor how packets could be scheduled without collisions. We will now formally introduce these problems, and then propose a possible solution.

### 4.4.1 Determining the Number of MEMS Switches

Consider a simple load-balanced router with just three linecards. Figure 4.6(a) shows three linecards connected as a full mesh; each linecard sends at rate $2R/3$ to every other linecard. Now partition the linecards into two groups, $A$ and $B$, with two linecards in group $A$ and one linecard in group $B$, as shown in Figure 4.6(b). We will determine how the electronic crossbars and MEMS switches are configured so that each pair of linecards is connected at rate $2R/3$. Group $A$ needs to send at an aggregate rate of $8R/3$ to group $A$, and $4R/3$ to group $B$; group $B$ needs to send at rate $4R/3$ to group $A$ and $2R/3$ to group $B$. Since we assumed that each crossbar output can send at maximum rate $2R$ through a given MEMS switch, we require two MEMS switches to connect group $A$ to group $A$. We therefore need a total of three MEMS switches, two arranged in the straight configuration and one arranged in the cross configuration. The correct configurations of the MEMS switches are shown in Figure 4.6(c).

Assume we pick the static configuration for each MEMS switch. Our objective is to determine the order in which the transmitting linecards will send packets for the receiving linecards. We want each sending linecard to spread packets uniformly over receiving linecards. We also want to avoid that packets conflict by using the same line at the same moment. To do so, we will adopt a *frame-based mechanism*. We will select a frame of packets sent by linecards and a frame of permutations followed by the electronic crossbars. Then, we will instruct each linecard and each crossbar to cycle repeatedly through these frames.

Let us illustrate what conflicts could occur in a frame. Consider the example in Figure 4.6 again. Since each sending linecard needs to spread its data uniformly over the three receiving linecards, the frame will have three slots. In the frame, each

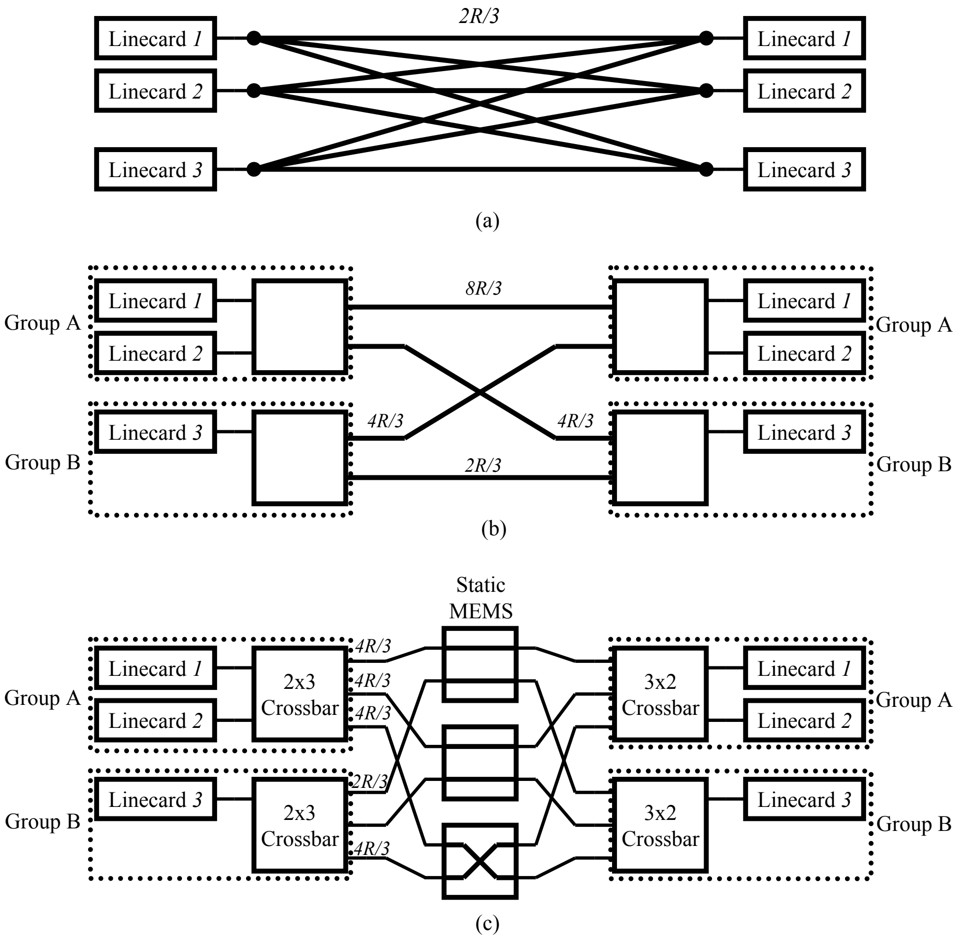


Figure 4.6: Example of a MEMS-based switch architecture with three linecards in two groups. (a) A full linecard mesh logical view. (b) Group $B$ is not fully populated, so the rates between groups are different. (c) The configuration of MEMS switches to achieve the required rates.

sending linecard will send exactly one packet to each of the three receiving linecards. Then, a conflict could occur when two sending linecards send a packet to the same receiving linecard at the same time, since these two packets would collide at the receiving linecard. A conflict could also occur when two sending linecards from the same sending group send packets to two receiving linecards from the same receiving group using the same MEMS switch. In this case, these packets would collide on the line linking the crossbar to the MEMS switch. Therefore, some conflicts are specific to linecards, while others are specific to MEMS switches. The algorithm that determines the frame needs to be aware of these different types of conflicts, which can only make the algorithm more complex.

We will now first formally describe the problem. Then, we will determine the minimum number of MEMS switches needed. Finally, we will introduce an algorithm that will provably construct a correct frame in polynomial time, and we will show some results on its running speed.

### 4.4.2 The Linecard Schedule Problem

We will assume throughout this chapter that there are $G$ groups; group $i$ contains $L_i$ linecards, and the total number of linecards is:

$$N = \sum_{i=1}^{G} L_i.$$

We will assume that $L_1$, $L_2$, ..., $L_G$ are fixed for a given linecard arrangement.

During every frame of $N$ time-slots each sending linecard needs to be connected exactly once to each of the $N$ receiving linecards. Similarly, each receiving linecard needs to be connected exactly once to each of the $N$ sending linecards. Furthermore, in every time-slot, each sending linecard cannot connect to more than one receiving linecard, and vice-versa.

Put mathematically, if sending linecard $i$ is connected to receiving linecard $T_{ij}$ in time-slot $j$, then:

$$\begin{cases} T_{ij'} \neq T_{ij} & \text{for all } j' \neq j \\ T_{i'j} \neq T_{ij} & \text{for all } i' \neq i \\ T_{ij} \in \{1, ..., N\} & \text{for all } i, j \end{cases}$$

We will call $T$ the *linecard schedule.* $T$ is a Latin square, i.e., the numbers from 1 to $N$ appears exactly once in every row and every column. We will refer to a time-slot as a column.

For instance, let us assume that $L_1 = 3$, $L_2 = 2$, and $L_3 = 2$ (i.e., $G = 3, N = 7$). Then the following is a linecard schedule:

$$T = \left(\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 3 & 4 & 5 & 6 & 7 & 1 \\ 3 & 4 & 5 & 6 & 7 & 1 & 2 \\ \hline 4 & 5 & 6 & 7 & 1 & 2 & 3 \\ 5 & 6 & 7 & 1 & 2 & 3 & 4 \\ \hline 6 & 7 & 1 & 2 & 3 & 4 & 5 \\ 7 & 1 & 2 & 3 & 4 & 5 & 6 \end{array}\right)$$

The last constraint arises from the use of MEMS switches in the MEMS-based architecture. Let $L_i$ represent the number of linecards in group $i$. The rate needed between group $i$ and group $j$ is equal to

$$(L_i \cdot 2R) \cdot (L_j/N), \text{ where } 1 \leq i, j \leq G.$$

This is because the incoming traffic is spread uniformly over all $N$ receiving linecards, and group $j$ receives a portion $(L_j/N)$ of this traffic. As assumed above, two groups can only communicate at a rate up to $2R$ through any single MEMS switch. Therefore, the minimum number of MEMS switches between group $i$ and group $j$ is:

$$\left\lceil \frac{L_i \cdot 2R \cdot L_j}{N} \cdot \frac{1}{2R} \right\rceil = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil.$$

We will call this the *MEMS constraint.*

Matrix $T$ above does not meet the MEMS constraint because the maximum number of connections allowed between transmitting group 1 and receiving group 1 at any time-slot is $\lceil \frac{3 \cdot 3}{7} \rceil = 2$. This is not satisfied in the first time-slot, because transmitting group 1 has three connections to receiving group 1 (more specifically to receiving linecards 1, 2 and 3). Similarly the second and third groups do not meet the constraint.

### 4.4.3 Number of MEMS Switches Needed for a Linecard Schedule

The following theorem shows how many MEMS switches are needed in order to build a linecard schedule that satisfies the MEMS constraint.

**Theorem 10** *We need at least*

$$\alpha = \sum_{j=1}^{G} \left\lceil \frac{L \cdot L_j}{N} \right\rceil \leq L + G - 1$$

*static MEMS switches in order to build a linecard schedule that satisfies the MEMS constraint, where* $L = \max_i(L_i)$.

*Proof:* A MEMS switch can connect a sending group to at most one receiving group, and the minimum number of MEMS switches needed to connect sending group $i$ to all receiving groups is:

$$\sum_{j=1}^{G} \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil .$$

In particular, assume that the largest group has $L = \max_i(L_i)$ linecards. Then the total number of MEMS switches needed by the largest group to connect to all receiving groups is at least:

$$\alpha = \sum_{j=1}^{G} \left\lceil \frac{L \cdot L_j}{N} \right\rceil < \sum_{j=1}^{G} \left( \frac{L \cdot L_j}{N} + 1 \right) = L + G.$$

Because $\alpha$, $L$ and $G$ are integers, $\alpha \leq L + G - 1$.

Therefore we need at most $L + G - 1$ static MEMS switches to create a uniform mesh with any linecard arrangement. ■

In our example with $L_1 = 3$, $L_2 = 2$, and $L_3 = 2$,

$$\alpha = \left\lceil \frac{3 \cdot 3}{7} \right\rceil + \left\lceil \frac{3 \cdot 2}{7} \right\rceil + \left\lceil \frac{3 \cdot 2}{7} \right\rceil = 4.$$

It is clear that $\alpha \leq L + G - 1 = 5$. It is also possible to reach the upper bound with a different linecard configuration where $L = 3$ and $G = 3$, for instance with $L_1 = 3$, $L_2 = 3$, and $L_3 = 2$.

### 4.4.4 Valid Schedules

**Linecard Schedule**

Our goal is to find an algorithm that works with exactly $\alpha$ MEMS switches, where $\alpha \leq L + G - 1$. To help illustrate such an algorithm, we will introduce three different types of schedules: (1) linecard-to-linecard, (2) linecard-to-group, and (3) group-to-group schedules. As described in the definitions below, the first part of the schedule name represents whether the schedule determines the specific sending linecards or only the sending groups, and the second part of the name specifies whether the schedule determines the specific receiving linecards or only the receiving groups. For instance, a linecard-to-group schedule will determine which linecard will send to which receiving group in each column.

**Definition 1** *A* linecard-to-linecard *(L-L) schedule $T$ is a matrix with $N$ rows corresponding to the $N$ sending linecards, $N$ columns corresponding to the $N$ time-slots of the frame, and one receiving linecard index per row-column intersection.*

Note that a linecard-to-linecard schedule is the same as a linecard schedule.

**Definition 2** *An L-L schedule $T$ is said to be* valid *iff a receiving linecard appears exactly once in every row and column of $T$, and at most $\left\lceil \frac{L_i \cdot L_j}{N} \right\rceil$ receiving linecards from group $j$ are connected to sending linecards from group $i$ in any column of $T$ (MEMS constraint).*

In other words, $T$ is a valid L-L schedule if it is a Latin square satisfying the MEMS constraints. Here is an example of valid L-L schedule.

$$T = \left(\begin{array}{ccccccc} 1 & 4 & 2 & 6 & 3 & 5 & 7 \\ 2 & 6 & 1 & 5 & 7 & 3 & 4 \\ 4 & 3 & 7 & 1 & 5 & 6 & 2 \\ \hline 5 & 1 & 3 & 4 & 2 & 7 & 6 \\ 7 & 5 & 4 & 3 & 6 & 2 & 1 \\ \hline 3 & 2 & 6 & 7 & 1 & 4 & 5 \\ 6 & 7 & 5 & 2 & 4 & 1 & 3 \end{array}\right)$$

Notice that the MEMS constraint used in Definition 2 applies to groups, not linecards. For instance, the example matrix $T$ is not allowed to have more than two receiving linecards from the first group in the first three rows in any column. Therefore, in order to build L-L schedules, we cannot only consider the constraints on linecards, but also need to take into account the constraints on groups. The MEMS constraint makes the linecard schedule problem non-trivial.

We will show that it is possible to build a valid *group-to-group* schedule that only considers constraints on groups, and then successively build a valid *linecard-to-group* schedule and finally a valid *linecard-to-linecard* schedule that incorporates the constraints on linecards. We will define and provide examples for these schedules below.

### Linecard-to-Group Schedule

**Definition 3** *A* linecard-to-group *(L-G) schedule $U$ is a matrix with $N$ rows corresponding to the $N$ sending linecards, $N$ columns corresponding to the $N$ time-slots of the frame, and one letter per row-column intersection corresponding to the receiving group.*

**Definition 4** *An L-G schedule $U$ is said to be* valid *iff the $i^{th}$ letter appears exactly $L_i$ times in each row and each column, and at most $\left\lceil \frac{L_i \cdot L_j}{N} \right\rceil$ times in the linecards of group $i$ in any column of $U$ (MEMS constraint).*

Here is an example of a valid L-G schedule, applied again to the same linecard configuration.

$$U = \left(\begin{array}{ccccccc} A & B & A & C & A & B & C \\ A & C & A & B & C & A & B \\ B & A & C & A & B & C & A \\ \hline B & A & A & B & A & C & C \\ C & B & B & A & C & A & A \\ \hline A & A & C & C & A & B & B \\ C & C & B & A & B & A & A \end{array}\right)$$

Notice that matrix $U$ is the same as matrix $T$ except that the receiving linecard indices are replaced with the letters corresponding to the receiving linecard group.

### Group-to-Group Schedule

**Definition 5** *A* group-to-group *(G-G) schedule $V$ is a matrix with $G$ rows corresponding to the $G$ sending linecard groups, $N$ columns corresponding to the $N$ time-slots of the frame, and $L_i$ letters per row-column intersection in row $i$.*

**Definition 6** *A G-G schedule $V$ is said to be* valid *iff the $i^{th}$ letter appears exactly $L_i \cdot L_j$ times in each row $j$ (corresponding to sending group $j$), $L_i$ times in each column, and at most $\left\lceil \frac{L_i \cdot L_j}{N} \right\rceil$ times in any row-column intersection in row $i$ (MEMS constraint).*

Here is an example of a valid G-G schedule, applied again to the same linecard configuration.

$$V = \left(\begin{array}{ccccccc} AAB & ABC & AAC & ABC & ABC & ABC & ABC \\ \hline BC & AB & AB & AB & AC & AC & AC \\ \hline AC & AC & BC & AC & AB & AB & AB \end{array}\right)$$

Notice that one can get matrix V by grouping together the rows corresponding to the same group in matrix U.

### Schedule Equivalence Theorem

Given a valid L-L schedule, we can easily deduce a valid L-G schedule, and then a valid G-G schedule. However, it is not obvious how to create a valid L-L schedule from a valid G-G schedule. The following theorem, which is proved in Appendix F.1, shows that we can.

**Theorem 11** *Consider the following three schedules:*

*(i) A valid linecard-to-linecard (L-L) schedule T*

*(ii) A valid linecard-to-group (L-G) schedule U*

*(iii) A valid group-to-group (G-G) schedule V*

*Given one schedule we can create the other two: (L-L)⇔(L-G)⇔(G-G).*

We will now show how to construct a valid G-G schedule, hence proving that it is always possible to obtain a linecard schedule that satisfies the MEMS constraint.

## 4.4.5 Constructing a Valid G-G Schedule

### Algorithm for Constructing a Valid G-G Schedule

We will now construct an algorithm that recursively builds a valid group schedule time-slot after time-slot, for the $N$ time-slots of the frame. We will then show in Appendix F.2 that the algorithm finds a valid solution, and that it has a polynomial complexity.

**At the start:**

Let $t$ be the number of time-slots left to schedule after each iteration. At the start, $t = N$, since all the time-slots are unscheduled. Also, let $M \stackrel{\triangle}{=} M^t = M^N$ be the initial matrix of all the elements that need to be scheduled. Its rows represent the sending groups, its columns the receiving groups (letters "A", "B", ...). At the start, for all $i, j$, $M_{ij} = L_i \cdot L_j$, i.e., there are $L_i \cdot L_j$ connections to schedule from sending group $i$ to receiving group $j$ during the whole frame.

**Iteratively:**

For $t = N, N-1, ..., 1$, proceed as follows.

1. For each $i, j$, do the decomposition of $M^t_{ij}$ in base $t$: $M^t_{ij} = t \times P^t_{ij} + Q^t_{ij}$ (i.e., $P^t = \left\lfloor \frac{1}{t} M^t \right\rfloor$, $Q^t = M^t - t \times P^t$). In this iteration, we will start by scheduling $P^t$, and then consider the remainder $Q^t$ and schedule a part of it such that all the constraints are satisfied.

   Intuitively, our objective is to uniformly spread traffic between any two groups across the different time-slots, so that no MEMS switch receives more traffic than it can handle at any precise time-slot. Since we want to schedule $M^t$ as uniformly as possible among the next $t$ time-slots, we will attempt to schedule $\frac{1}{t}$-th of it in the next time-slot. Therefore, we first decompose $M^t$ as the sum of $t \times P^t$ and $Q^t$. Then, we schedule exactly $P^t$ at this time-slot, and consider what part of $Q^t$ needs to be scheduled to satisfy the scheduling constraints.

2. Define the vectors $a^t$ and $b^t$ such that

   $$\begin{cases} a^t_i = \frac{\sum_{j'=1}^{G} Q^t_{ij'}}{t} & \text{for all } i \\ b^t_j = \frac{\sum_{i'=1}^{G} Q^t_{i'j}}{t} & \text{for all } j \end{cases}$$

   $a^t$ and $b^t$ are integer vectors (proved in Appendix F.2). $a^t$ and $b^t$ represent the amount of traffic that should be sent from (and received by) each group at the next time-slot out of the remainder matrix $Q^t$.

3. Find a 0-1 matrix $R^t \leq Q^t$ such that:

   $$\begin{cases} \sum_{j'=1}^{G} R^t_{ij'} = a^t_i & \text{for all } i \\ \sum_{i'=1}^{G} R^t_{i'j} = b^t_j & \text{for all } j \\ R^t_{ij} \in \{0, 1\} & \text{for all } i, j \end{cases}$$

   The first two conditions state that each group sends and receives the correct amount of traffic in the next time-slot, so that the schedule remains valid. The last condition is a technical condition used in the proof. The proof in Appendix F.2 shows that $R^t$ exists (it uses graph theory for proof of existence, and the Ford-Fulkerson max-flow algorithm for constructing it).

Table 4.1: Example of application of the algorithm for constructing a valid G-G schedule

$$M^7 = \begin{pmatrix} 9 & 6 & 6 \\ 6 & 4 & 4 \\ 6 & 4 & 4 \end{pmatrix}, P^7 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, Q^7 = \begin{pmatrix} 2 & 6 & 6 \\ 6 & 4 & 4 \\ 6 & 4 & 4 \end{pmatrix}, R^7 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}, S^7 = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

$$M^6 = \begin{pmatrix} 7 & 5 & 6 \\ 6 & 3 & 3 \\ 6 & 4 & 3 \end{pmatrix}, P^6 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, Q^6 = \begin{pmatrix} 1 & 5 & 0 \\ 0 & 3 & 3 \\ 6 & 4 & 3 \end{pmatrix}, R^6 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, S^6 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

$$M^5 = \begin{pmatrix} 6 & 4 & 5 \\ 5 & 2 & 3 \\ 4 & 4 & 2 \end{pmatrix}, P^5 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, Q^5 = \begin{pmatrix} 1 & 4 & 0 \\ 0 & 2 & 3 \\ 4 & 4 & 2 \end{pmatrix}, R^5 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}, S^5 = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}.$$

$$M^4 = \begin{pmatrix} 4 & 4 & 4 \\ 4 & 1 & 3 \\ 4 & 3 & 1 \end{pmatrix}, P^4 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, Q^4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 3 \\ 0 & 3 & 1 \end{pmatrix}, R^4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, S^4 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

$$\text{For t=3,2,1: } M^t = \begin{pmatrix} t & t & t \\ t & 0 & t \\ t & t & 0 \end{pmatrix} = t \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, R^t = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{ and } S^t = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

4. Use the schedule $S^t = P^t + R^t$ for this time-slot. Update $M^{t-1} = M^t - S^t$.

   In other words, the amount of traffic left to schedule at the end of the time-slot is the difference between the amount of traffic left to schedule at the start of the time-slot and the amount of traffic scheduled during the time-slot.

### Example

We build the matrix $V$ given the schedules, $S^t$, provided in Table 4.1. More specifically, $S^t_{ij}$ represents the number of occurrences of the $j^{th}$ letter in the $i^{th}$ row in column $N - t + 1$ of matrix $V$. For instance, the schedule

$$S^7 = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

helps us create the first column of $V$, with two $A$'s and one $B$ in the first row, one $B$ and one $C$ in the second row, and one $A$ and one $C$ in the last row. $S^6$ will determine the second column, $S^5$ will determine the third column, and so on. The resulting matrix is

$$V = \left( \begin{array}{ccccccc} AAB & ABC & AAC & ABC & ABC & ABC & ABC \\ \hline BC & AB & AB & AB & AC & AC & AC \\ \hline AC & AC & BC & AC & AB & AB & AB \end{array} \right)$$

### 4.4.6 Valid L-L Schedule

We will now construct an algorithm that successively transforms a valid G-G schedule into a valid L-G schedule, and then a valid L-G schedule into a valid L-L schedule. This algorithm will be used in Appendix F.1 to prove Theorem 11.

**From a Valid G-G Schedule to a Valid L-G Schedule**

Let us start with a valid G-G schedule $V$. For each $1 \leq j \leq G$, consider row $j$ in $V$. In our example, the first row is:

$$\left( \begin{array}{ccccccc} AAB & ABC & AAC & ABC & ABC & ABC & ABC \end{array} \right)$$

We want to subdivide each row $j$ into $L_j$ sub-rows, corresponding to the subdivision of each sending group $j$ into $L_j$ sending linecards, thus forming a valid L-G schedule.

First, each letter has $L_i \cdot L_j$ occurrences in any given row of $V$. Arbitrarily divide them into $L_i$ subscripted letters ("sub-letters") of $L_j$ elements. In our example, we transform the letters of $V$ into $N$ arbitrarily assigned sub-letters $(A_1, A_2, A_3, B_1, B_2, C_1, C_2)$. For instance, since $A$ appears nine times in the first row, we replace the $A's$ arbitrarily with three $A_1$'s, three $A_2$'s and three $A_3$'s:

$$\left( \begin{array}{ccccccc} A_1A_1B_1 & A_1B_1C_1 & A_2A_2C_1 & A_2B_1C_1 & A_3B_2C_2 & A_3B_2C_2 & A_3B_2C_2 \end{array} \right)$$

In row $j$ of matrix $V$, each of the $N$ sub-letters has $L_j$ occurrences, and each of the $N$ columns has $L_j$ elements. We can form a new matrix that has sub-letters as inputs and columns as outputs. In this new matrix, all columns and all rows have $L_j$ elements. In our example, the new matrix for the first row of $V$ is :

$$\left(\begin{array}{c||ccccccc}
 & col.1 & col.2 & col.3 & col.4 & col.5 & col.6 & col.7 \\
\hline\hline
A_1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
A_2 & 0 & 0 & 2 & 1 & 0 & 0 & 0 \\
A_3 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
B_1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
B_2 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
C_1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
C_2 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{array}\right)$$

We can now apply the Birkhoff-von Neumann decomposition theorem to this matrix, by decomposing it into a sum of $L_j$ permutations [12, 19].[1] By reading column after column, each of the $L_j$ permutations gives a sequence of sub-letters that corresponds to a row of the desired L-G schedule. Therefore, the $L_j$ permutations yield the $L_j$ rows of the L-G schedule corresponding to group $j$. In our example, the first permutation could be:

$$\left(\begin{array}{c||ccccccc}
 & col.1 & col.2 & col.3 & col.4 & col.5 & col.6 & col.7 \\
\hline\hline
A_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
A_3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
B_1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
B_2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
C_1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
C_2 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}\right),$$

[1] In this case, since all elements are integers, we can equivalently apply König's edge-coloring theorem, which states that the edge-coloring number of a bipartite graph is equal to its maximum degree [55, 73]. Therefore, we can use graph-coloring algorithms instead [1, 26, 27, 74]. Note that the Slepian-Duguid algorithm is also a practical alternative for fast implementation [34, 41, 77].

yielding the first row of:

$$\begin{pmatrix} A_1 & B_1 & A_2 & C_1 & A_3 & B_2 & C_2 \\ A_1 & C_1 & A_2 & B_1 & C_2 & A_3 & B_2 \\ B_1 & A_1 & C_1 & A_2 & B_2 & C_2 & A_3 \end{pmatrix}$$

We finally replace each sub-letter by the corresponding letter, and get the valid L-G schedule. Upon examination of the algorithm, it is clear that we only permute letters within the same column of the same sending group, thus yielding a valid L-G schedule. In our example, the resulting L-G schedule is:

$$U = \left(\begin{array}{ccccccc} A & B & A & C & A & B & C \\ A & C & A & B & C & A & B \\ B & A & C & A & B & C & A \\ \hline B & A & A & B & A & C & C \\ C & B & B & A & C & A & A \\ \hline A & A & C & C & A & B & B \\ C & C & B & A & B & A & A \end{array}\right)$$

### From a Valid L-G Schedule to a Valid L-L Schedule

We transformed above any valid G-G schedule into a valid L-G schedule. We will now transform the valid L-G schedule into a valid L-L schedule.

We apply the Birkhoff-von Neumann theorem for each letter. First, we replace each $A$ with a "1", and every other letter with a "0". For our example, we get:

$$\left(\begin{array}{ccccccc} A & 0 & A & 0 & A & 0 & 0 \\ A & 0 & A & 0 & 0 & A & 0 \\ 0 & A & 0 & A & 0 & 0 & A \\ \hline 0 & A & A & 0 & A & 0 & 0 \\ 0 & 0 & 0 & A & 0 & A & A \\ \hline A & A & 0 & 0 & A & 0 & 0 \\ 0 & 0 & 0 & A & 0 & A & A \end{array}\right) \rightarrow \left(\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right)$$

We then decompose the above matrix into the sum of $L_i$ different permutations, such that the $l^{th}$ permutation will indicate at which times linecard $l$ is scheduled. Since there are exactly $L_1$ ones (corresponding to the $L_1$ $A$'s) in each row and column, this is possible by Birkhoff-von Neumann. In our example, we can decompose the above matrix into the sum of three permutations:

$$\left(\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right) + \left(\begin{array}{ccccccc} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right) + \left(\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Applying this method to each letter, we then create a valid L-L schedule, with exactly one occurrence of each receiving linecard index in each row and column. In our example, we get the following valid L-L schedule, hereby concluding the construction process:

$$T = \left(\begin{array}{ccccccc} 1 & 4 & 2 & 6 & 3 & 5 & 7 \\ 2 & 6 & 1 & 5 & 7 & 3 & 4 \\ 4 & 3 & 7 & 1 & 5 & 6 & 2 \\ \hline 5 & 1 & 3 & 4 & 2 & 7 & 6 \\ 7 & 5 & 4 & 3 & 6 & 2 & 1 \\ \hline 3 & 2 & 6 & 7 & 1 & 4 & 5 \\ 6 & 7 & 5 & 2 & 4 & 1 & 3 \end{array}\right)$$

### 4.4.7 Practical Considerations

Upon linecard failure, current network protection switching time requirements impose that restoration time be below 50ms in order to provide a fast recovery [4, 42, 81, 82]. A Stanford student, Srikanth Arekapudi, implemented the algorithm in

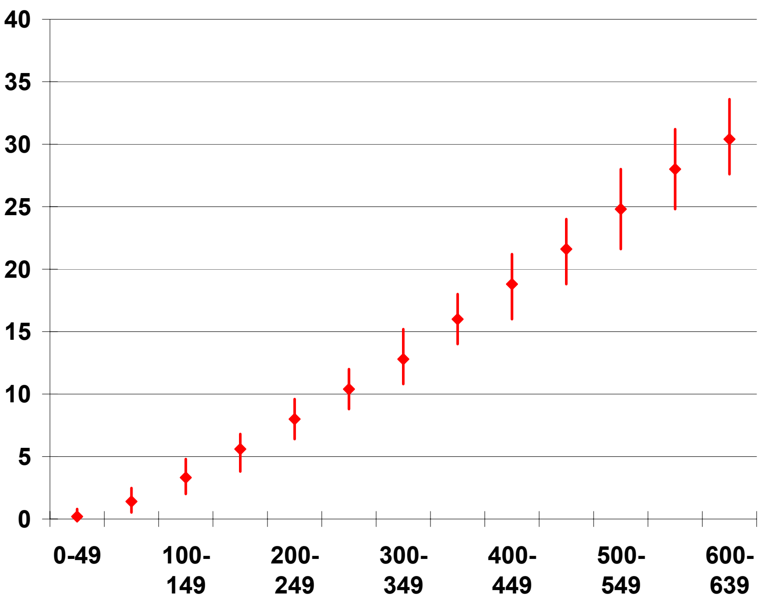


Figure 4.7: Running time in milliseconds of the algorithm implementation, as a function of the number of linecards. The plot represents the worst-case, average and best-case values for each range.

Verilog in order to estimate how long the algorithm takes to run in practice [5, 6]. Because of the Verilog simulation time, he also developed a cycle-accurate C-model and verified its accuracy by comparing it with the Verilog implementation. The C-model reports the number of clock cycles the Verilog simulation would run for. Given an arrangement of linecards and groups, the program finds a valid linecard schedule to configure the MEMS switches and electronic crossbars. The objective is for the implementation to run in less than 50ms.

In the simulation, the program first selects a number of linecards between 0 and 640. Then, it spreads linecards uniformly at random over $G = 40$ groups, with a maximum of 16 linecards per group. This is repeated for different linecard ranges between 0 and 640 (e.g., a range of 50 linecards between 100 and 149). For each range, the worst case times are collected for each run over 1000 runs. The implementation assumes that the clock cycle is 4ns, and that the degree of parallelism can be equal to the number of groups for the L-L and L-G schedules. It does not consider the time required to upload and download the matrix information to/from the chips.

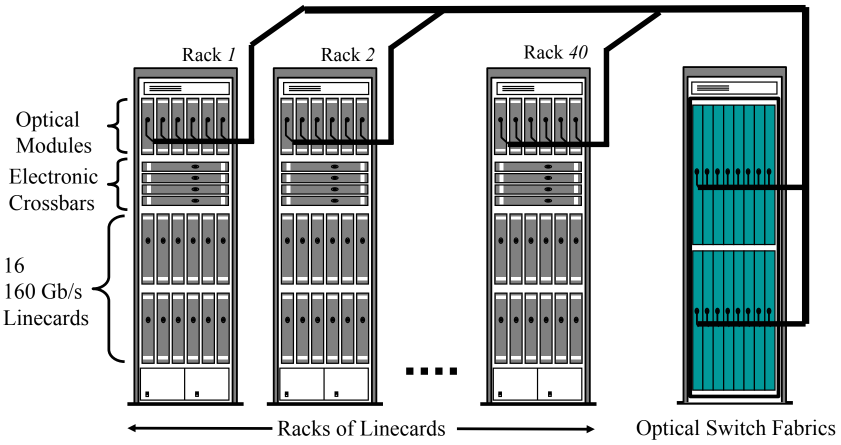


Figure 4.8: Possible system packaging for a 100 Tb/s router with 640 linecards arranged as 40 racks with 16 linecards per rack.

Figure 4.7 illustrates the running time of this implementation in milliseconds, as a function of the number of linecards. The main result is that the worst-case running time for all trials was under the 50ms delay allowed for a router to recover from a failure. Of course, a real-life implementation might make different assumptions about the clock speed, the degree of parallelism, and so on. However, the complexity of the algorithm seems to be roughly right, at least within an order of magnitude. (Note that if the algorithm was too complex, we could also run the algorithm in advance and store the results for all possible single-linecard or single-group failures.)

## 4.5 Practicality and Reliability of the 100Tb/s Router

We have seen above how a load-balanced router can be implemented and scheduled using the MEMS-based architecture. It is now worth asking: Can we build a 100Tb/s router using this architecture, and if so, could we package it in a way that network operators could deploy it in their network?

We believe that it is possible to build the 100Tb/s MEMS-based router with

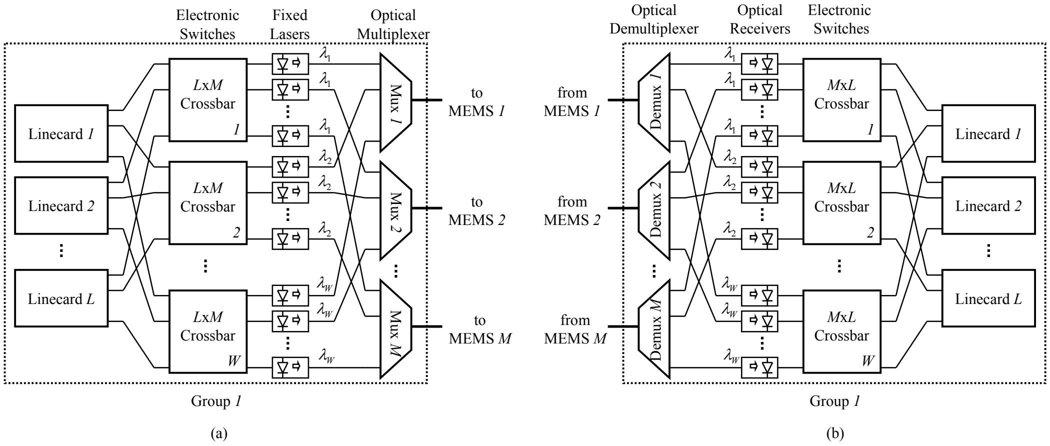


Figure 4.9: Bit-sliced crossbars for the MEMS-based architecture. (a) represents the transmitting side of the switch fabric. (b) represents the receiving side of the switch fabric.

technology available today. The system could be packaged in multiple racks as shown in Figure 4.8, with $G = 40$ racks each containing $L = 16$ linecards, interconnected by $L + G - 1 = 55$ statically configured $40 \times 40$ MEMS switches.

To justify this, we break the question down into a number of smaller questions. Our goal is to address the most critical issues that a system designer would consider when building such a system. Clearly our list cannot be complete. Different systems have different requirements, and must operate in different environments. With this caveat, we consider the following different aspects.

### 4.5.1 The Electronic Crossbars

In the description of the MEMS-based architecture, we assumed that one electronic crossbar interconnects a group of linecards, each at rate $2R = 320$Gb/s. This is too fast for a single crossbar, but we can use bit-slicing. We will assume $W$ crossbar slices, where $W$ is chosen to make the serial link data-rate achievable. For example, with $W = 32$, the serial links operate at a more practical 10Gb/s. Each slice would be a $16 \times 55$ crossbar operating at 10Gb/s. This is less than the capacity of crossbars that have already been reported [29].

Figure 4.9 shows $L$ linecards in a group connected to $W$ crossbar slices, each operating at rate $2R/W$. As before, the outputs of the crossbar slices are connected to lasers. But now, the lasers attached to each slice operate at a different, fixed wavelength, and data from all the slices to the same MEMS switch are multiplexed onto a single fiber. As before, the group is connected to the MEMS switches with $M$ fibers. If a packet is sent on the $n$-th crossbar slice, it will be delivered to the $n$-th crossbar slice of the receiving group. Apart from the use of slices to make a parallel datapath, the operation is the same as before.

Each slice would connect to $M = 55$ lasers or optical receivers. This is probably the most technically challenging, and interesting, design problem for this architecture. One option is to connect the crossbars to external optical modules, but this might lead to prohibitively high power consumption in the electronic serial links. We could reduce power if we could directly connect the optical components to the crossbar chips. The direct attachment (or "solder bumping") of III-V opto-electronic devices onto silicon has been demonstrated [56], but is not yet a mature, manufacturable technology. Another option is to attach optical modulators rather than lasers. An external, high powered continuous wave laser source could illuminate an array of integrated modulators on the crossbar switch. The array of modulators modulate the optical signal and couple it to an outgoing fiber [57].

### 4.5.2 Packaging 100Tb/s of MEMS Switches

We can say with confidence that the power consumption of the optical switch fabric will not limit the router's capacity. Our architecture assumes that a large number of MEMS switches are packaged centrally. Because they are statically configured, MEMS switches consume almost no power, and all 100Tb/s of switching can be easily packaged in one rack using MEMS switches commercially available. Compare this with a 100Tb/s electronic crossbar switch, that connects to the linecards using optical fibers. Using today's serial link technology, the electronic serial links alone would consume approximately 8kW (assume 400mW and 10Gb/s per bidirectional serial link). The crossbar function would take at least 100 chips, requiring multiple extra serial links between them; hence the power would be much higher. Furthermore, the switch needs to terminate over 20,000 optical channels operating at 10Gb/s. Today, with commercially available optical modules, this would consume tens of kilowatts, would be unreliable and prohibitively expensive.

### 4.5.3 Fault-Tolerance

The load-balanced architecture is inherently fault-tolerant. First, because it has no centralized scheduler, there is no electrical central point of failure for the router. The only centrally shared devices are the statically configured MEMS switches, which can be protected by extra fibers from each linecard rack, and spare MEMS switches. Second, the failure of one linecard will not make the whole system fail; the MEMS switches are reconfigured to spread data over the correctly functioning linecards. Third, the crossbars in each group can be protected by an additional crossbar slice.

### 4.5.4 Building 160Gb/s Linecards

We assume that the address lookup, header processing and buffering on the linecards are all electronic. At 160Gb/s, a new minimum length 40-byte packet can arrive every 2ns, which can be processed by a pipeline in dedicated hardware in a 90nm ASIC process. 40Gb/s linecards are already commercially available, and reductions in geometries and increases in clock speeds make 160Gb/s possible.

Address lookups are challenging at this speed, but pipelined lookups every 2ns for IPv4 longest-prefix matching are feasible. For example, one could use 24Mbytes of 2ns SRAM (Static RAM) and the brute force lookup algorithm in [39] that completes one lookup per memory reference in a pipelined implementation. Alternatively, one could use four 40Gb/s address lookup devices in parallel.

The biggest challenge is simply writing and reading packets from buffer memory at 160Gb/s. Router linecards contain 250ms or more of buffering so that TCP will behave well when the router is a bottleneck, which requires the use of DRAM (dynamic RAM). Currently, the random access time of DRAMs is 40ns, and historically DRAMs have increased in random access speed by only 10% every 18 months. This problem is solved in [44] by designing a packet buffer using commercial memory devices, but with the speed of SRAM and the density of DRAM. This technique makes it possible to build buffers for 160Gb/s linecards and 128-byte cells.

### 4.5.5 Packaging 16 Linecards in a Rack

Network operators frequently complain about the power consumption of 10Gb/s and 40Gb/s linecards today (200W per linecard is common). If a 160Gb/s linecard consumes more power than a 40Gb/s linecard today, then it will be difficult to package 16 linecards in one rack ($16 \times 200 = 3.2kW$). If improvements in technology do not solve this problem over time, we can put fewer linecards in each rack, so long as $G \times L \geq 640$. For example, we could halve the number of linecards per rack and double the number of groups. This comes at the expense of slightly more MEMS switches ($M \geq L + G - 1$).

## 4.6 Consequences

In this chapter, we proposed an architecture for a high-performance router. With this architecture, a router can scale to a high number of linecards. It can also operate correctly when populated with any number of linecards connected to any ports. Finally, it can send packets without conflicts, and find a new schedule in less than 50ms in the event of a linecard failure.

All the above considerations illustrate what we set out to do: to design a 100Tb/s load-balanced router, with guaranteed 100% throughput under all traffic conditions. The architecture we described, including the switch fabric and the linecards, can be built using technology available today, and fit within the power constraints required by network operators.

# Chapter 5

# Conclusion

This thesis has demonstrated that it is possible today to build a 100Tb/s load-balanced Internet router with an optical switch fabric, a 100% throughput guarantee, and an average packet delay within a constant from the optimum for any traffic arrival pattern. The router does not exhibit packet reordering, and needs no centralized scheduling, no frequent exchange of state information, and no frequent switch fabric reconfigurations.

## 5.1 Towards a Simpler Internet

No high-speed commercial router today can guarantee 100% throughput for all traffic arrival patterns. Therefore, in order to obtain sufficient throughput guarantees, network operators over-provision capacity in central offices by aggregating several routers per central office. In addition, router capacity nearly follows Moore's law and doubles every 18 months [52], while Internet traffic demand is reported to be doubling every year [63, 64]. Therefore, router capacity grows more slowly than Internet traffic demand. Unless something changes, network operators will continue to place more and more routers in each central office. For instance, among the network operators considered in [78, 79], the mean central office size is 7.6, and large central offices have over 40 routers.

However, such ad-hoc aggregations of routers do not guarantee 100% throughput for all traffic arrival patterns. These clusters are also increasingly difficult to manage, and waste significant capacity interconnecting the routers. Operators would therefore like to replace these routers with a single large router per central office.

In this thesis, we have shown that it is possible today to build a router that can handle all traffic in a central office as well as guarantee 100% throughput for all traffic patterns. This router is a load-balanced router.

Therefore, a direct consequence is that network operators can replace router aggregates with a single router in their central offices. This single large router will simplify the management of central offices and guarantee that they are fully utilized.

## 5.2 Future Directions in Load-Balanced Router Architectures

Throughout this thesis, it has been assumed that the load-balanced router has three stages of packet processing and buffering. First, the input stage, with address lookup, packet processing and, at most, a small coordination buffer. Second, the intermediate stage, with most of the packet buffering. And last, the output stage, with at most a small reordering coordination buffer.

All these stages are realized in an electronic linecard. However, this electronic linecard consumes power because of the optical-to-electrical and electrical-to-optical conversions as well as because of the electrical links. In the future, designers might want to use optics in the linecards and implement these three stages with optical components. The use of optics will decrease power consumption and increase bandwidth.

However, two functions in high-speed routers seem to be out of reach of optics for the near future: the address lookup and the main buffering. Therefore, we can expect router designers to group these two functions in the same stage and implement the two other stages using optics. Since the address lookup cannot be located in the output, there are two stages in which it is possible to group the lookup and buffering functions: *the intermediate stage* and *the input stage*. Let us consider these two possibilities.

First, with the basic load-balanced router defined in the Introduction, it is possible to group the lookup and buffering functions in *the intermediate stage*. For instance, assuming arrivals of fixed-size packets, an optical input stage could simply point to different outputs in a round-robin order and spread packets as they arrive. The electronic intermediate stage would implement the address lookup, packet classification and packet buffering. Finally, the optical output stage would receive the (possibly out-of-order) packets and forward them to the output line.

Alternatively, one promising trend in current research is to examine whether to implement the lookup and buffering functions in *the input stage*. In [22], C.-S. Chang *et al.* develop a scheme in which the main buffering is realized at the input stage, while the intermediate stage only contains a small coordination buffer. Under this scheme, the electronic input linecard can realize the address lookup, packet classification and packet buffering functions. Then, the optical intermediate stage and the optical output stage can simply contain a small coordination buffer. As shown in [22], this scheme only requires that arrival traffic be admissible over consecutive frames. Thus, this scheme would apply particularly well to periodic traffic such as SONET. In addition, any match in an input-queued switch is admissible by definition, and can be considered as a frame of size 1. Therefore, this scheme can also be extended to emulate any input-queued switch.

## 5.3 Applying Load-Balanced Routing to Network Design

The main function of a router is to connect $N$ ports, where each port transmits and receives packets at some maximum rate $R$. The main function of a network is to connect $N$ nodes, where each node transmits and receives packets at some maximum rate $R$. On the face of it, the functions are identical. So could we not use the ideas developed in the load-balanced router to implement load-balanced networks?

In fact, we could use the load-balanced router to revolutionize the design of metropolitan area networks [53] as well as the design of the Internet backbone [87].

Each node would spread its arriving traffic across all other nodes, which would forward the traffic to its correct destination. There are many interesting properties in such a network design. Given a uniform mesh, Chapter 2 proves that the load-balanced router would achieve an optimal throughput. In addition, by spreading packets across several paths, it would also achieve a greater fault-tolerance, and a better security with respect to single-path eavesdropping. Finally, multiplexing effects would also make traffic more predictable and less bursty. Of course, challenges remain, such as reducing packet delays. However, by providing guarantees on throughput, fault-tolerance, security and traffic predictability, this novel backbone architecture is promising for the future scalability and reliability of the Internet.

# Appendix A

# Optimality of the Biased Mesh

In this Appendix, we will prove that the biased mesh achieves the maximum possible guaranteed throughput for any possible admissible capacity matrix by establishing Proposition 7.

It helps to study how the load-balancing is done. Let the set of *load-balanced paths*

$$P_{LB}(i,j) = \{p \in P(i,j) : (i \to j) \notin p\}$$

be the set of paths $p$ between nodes $i$ and $j$ such that the link $i \to j$ is not in $p$. We will call paths not in $P_{LB}(i,j)$ *direct paths*. For instance, $1 \to 3 \to 2$ is a load-balanced path between node 1 and node 2, whereas $1 \to 2$ and $1 \to 1 \to 2 \to 2$ are direct paths.

**Proposition 12** *$p \in P(i,j)$ satisfies one of the following two cases:*

*(i) If $p$ is a direct path then $(i \to j) \in p$, or*

*(ii) If $p$ is a load-balanced path then there exist two nodes $k$ and $l$, possibly equal, such that $k \neq i$, $k \neq j$, $l \neq i$ and $l \neq j$, such that $p$ contains $i \to k$ and $l \to j$.*

*Proof:* (i) clearly follows from the definition of $P_{LB}(i,j)$. In (ii), by definition of $P_{LB}(i,j)$, at least one node is different from $i$, and if node $k$ is the first node in path $p$ that is different from $i$, then $k \neq j$ also. Similarly, if node $l$ is the last node in path $p$ that is different from $j$, then $l \neq i$ also. ■

Using this characterization of load-balanced paths, we consider all the rate matrices that are also permutation matrices, such that each node sends all its traffic to some other node. For a permutation $\sigma$, let

$$S_1(\sigma) = \{i : \sigma(i) = i\}$$

denote the set of nodes invariant to $\sigma$, and let

$$S_2(\sigma) = \{i : \sigma(i) \neq i\} = \{1, ..., N\} \setminus S_1$$

denote the remaining nodes. The following lemma proves a general upper bound on the throughput $\theta(C)$ by considering the set of rate matrices that are permutation matrices.

**Lemma 13** *Given a capacity matrix $C$, the throughput $\theta(C)$ has the following upper bound:*

$$\theta(C) \leq \frac{1}{2} + \frac{1}{2N} \times \min_{\sigma \ permutation} \left( \sum_{i \in S_1(\sigma)} C_{ii} + \sum_{i \in S_2(\sigma)} (C_{i\sigma(i)} - C_{ii}) \right) \tag{A.1}$$

*Proof:* By definition of the throughput $\theta(C)$, for any permutation $\sigma$,

$$\theta(C) \leq \theta(C, \sigma).$$

Therefore, we only need to show that for any permutation $\sigma$,

$$\theta(C, \sigma) \leq \frac{1}{2} + \frac{1}{2N} \left( \sum_{i \in S_1(\sigma)} C_{ii} + \sum_{i \in S_2(\sigma)} (C_{i\sigma(i)} - C_{ii}) \right).$$

Consider a given permutation $\sigma$. By definition of $\theta(C, \sigma)$, any node $i$ manages to send traffic at rate $\theta(C, \sigma)$ to node $j = \sigma(i)$. (We know that the optimum can be reached because the throughput $\theta$ is defined using continuous functions on compact sets.)

Consider then a path $p$ between $i$ and $j = \sigma(i)$, and distinguish between the following cases.

1. If $p \notin P_{LB}(i, j)$, then from Proposition 12, $p$ contributes at least $T_{ij}^p$ to $C_{ij}$.

2. If $p \in P_{LB}(i, j)$, then by Proposition 12 there exists two nodes $k$ and $l$ such that $k \neq i$, $k \neq j$, $l \neq i$ and $l \neq j$, and such that $p$ contains $i \to k$ and $l \to j$. Hence, $p$ will use a rate of at least $T_{ij}^p$ out of the capacity $C_{ik}$ in order to carry the link $i \to k$, and will also use a rate of at least $T_{ij}^p$ out of the capacity $C_{lj}$ in order to carry the link $l \to j$.

   Therefore, $p$ requires a total rate of at least $2 \cdot T_{ij}^p$ from the non-diagonal elements of the capacity matrix $C$.

These two cases show that the link between $i$ and $j = \sigma(i)$ can use non-diagonal capacity both with direct and load-balanced paths.

In particular, the first case studies the *direct* paths. It shows that the link between $i$ and $j = \sigma(i)$ uses a rate of at least $\sum_{p \notin P_{LB}(i,\sigma(i))} T_{i\sigma(i)}^p$ out of $C_{i\sigma(i)}$ for the direct paths. This is a contribution to the non-diagonal capacity if and only if $\sigma(i) \neq i$, i.e., $i \in S_2(\sigma)$. Also, since the capacity for the direct link should be greater than its rate in order to be feasible, we get

$$C_{i\sigma(i)} \geq \sum_{p \notin P_{LB}(i,\sigma(i))} T_{i\sigma(i)}^p. \tag{A.2}$$

The second case studies the *load-balanced* paths. It shows that the link between $i$ and $j = \sigma(i)$ uses a rate of at least $\sum_{p \in P_{LB}(i,\sigma(i))} 2 \cdot T_{i\sigma(i)}^p$ out of the non-diagonal elements of $C$ for the load-balanced paths.

As a feasibility condition, the sum of the capacities of all the non-diagonal links should be more than the sum of all the rates required from these non-diagonal links. Therefore, using the two cases studied above, we get

$$\text{non-diagonal capacity} \geq \text{non-diagonal required rate},$$

i.e.,

$$\sum_{i,j\neq i} C_{ij} \quad \geq \quad \sum_{i\in S_2(\sigma)} \sum_{p\notin P_{LB}(i,\sigma(i))} T^p_{i\sigma(i)} + \sum_{i=1}^{N} \sum_{p\in P_{LB}(i,\sigma(i))} 2T^p_{i\sigma(i)}.$$

We now study the two sides of this equation. On the left hand side, since $C$ is admissible, we have

$$N - \sum_i C_{ii} \geq \sum_{i,j\neq i} C_{ij}.$$

On the right hand side, the sum of all the rates required from these non-diagonal links can be rewritten as

$$\begin{aligned}
&\sum_{i\in S_2(\sigma)} \sum_{p\notin P_{LB}(i,\sigma(i))} T^p_{i\sigma(i)} \\
&+\left[\sum_{i=1}^{N}\left(\sum_{p\notin P_{LB}(i,\sigma(i))} 2T^p_{i\sigma(i)} + \sum_{p\in P_{LB}(i,\sigma(i))} 2T^p_{i\sigma(i)}\right) - \sum_{i=1}^{N} \sum_{p\notin P_{LB}(i,\sigma(i))} 2T^p_{i\sigma(i)}\right] \\
&= 2N\theta(C,\sigma) - 2\sum_{i\in S_1(\sigma)} \sum_{p\notin P_{LB}(i,\sigma(i))} T^p_{i\sigma(i)} - \sum_{i\in S_2(\sigma)} \sum_{p\notin P_{LB}(i,\sigma(i))} T^p_{i\sigma(i)},
\end{aligned}$$

using $T = \theta(C,\sigma)\cdot\sigma$ and $S_1(\sigma)\cup S_2(\sigma) = \{1,...,N\}$ in the last equality. Using Equation (A.2), the sum of the non-diagonal rates can therefore be lower bounded by

$$2N\theta(C,\sigma) - 2\sum_{i\in S_1(\sigma)} C_{i\sigma(i)} - \sum_{i\in S_2(\sigma)} C_{i\sigma(i)}.$$

Finally, we combine the equations and use the definition of $S_1(\sigma)$: $i\in S_1(\sigma)$ iff $\sigma(i) = i$. We get

$$N - \sum_i C_{ii} \geq 2N\theta(C,\sigma) - 2\sum_{i\in S_1(\sigma)} C_{ii} - \sum_{i\in S_2(\sigma)} C_{i\sigma(i)}.$$

Therefore,

$$\theta(C,\sigma) \leq \frac{1}{2} + \frac{1}{2N}\left(\sum_{i\in S_1(\sigma)} C_{ii} + \sum_{i\in S_2(\sigma)} (C_{i\sigma(i)} - C_{ii})\right).$$

■

Now we consider specific permutations. For $0 \leq k \leq N-1$, define the permutation $\sigma_k$ as the $k^{th}$ sub-diagonal, i.e., assume that node $i$ destines all its traffic to $\sigma_k(i) = i + k \bmod N$. We can then apply Lemma 13 to find the upper bound corresponding to each permutation, as expressed in the following lemma.

**Lemma 14** *Given a capacity matrix $C$, the throughput $\theta(C)$ has the following upper bounds:*

$$\theta(C) \leq \frac{1}{2} + \frac{\sum_{i=1}^{N} C_{ii}}{2N}, \tag{A.3}$$

*and*

$$\theta(C) \leq \frac{1}{2} + \frac{\min_{1\leq k\leq N}(\sum_{i=1}^{N}(C_{i(i+k \bmod N)} - C_{ii}))}{2N}. \tag{A.4}$$

*Proof:* For $k = 0$, $S_1(\sigma_k) = \{1, ..., N\}$, hence the upper-bound from Lemma 13 is $\frac{1}{2} + \frac{\sum_{i=1}^{N} C_{ii}}{2N}$. Similarly, for $1 \leq k \leq N-1$, $S_2(\sigma_k) = \{1, ..., N\}$, hence this upper-bound is $\frac{1}{2} + \frac{\sum_{i=1}^{N}(C_{i\sigma_k(i)} - C_{ii})}{2N}$. ■

**Proposition 15** *If the capacity matrix $C$ is admissible, i.e., $C$ is a doubly sub-stochastic matrix, then the throughput $\theta(C) \leq \frac{N}{2N-1}$.*

*Proof:* We will prove this by contradiction. Suppose that $\theta(C) > \frac{N}{2N-1}$. For $0 \leq k \leq N-1$, let

$$x_k = \sum_{i=1}^{N} C_{i(i+k \bmod N)}.$$

It follows from (A.3) and (A.4) that

$$x_0 > \frac{N}{2N-1}, \tag{A.5}$$

and for $k = 1, 2, \ldots, N-1$,

$$x_k - x_0 > \frac{N}{2N-1}. \tag{A.6}$$

Therefore, we have $x_k > \frac{2N}{2N-1}$ for $k = 1, 2, \ldots, N-1$. Summing up for all $k$ yields

$$N < \sum_{k=0}^{N-1} x_k = \sum_{i=1}^{N} \sum_{j=1}^{N} C_{i,j}.$$

This contradicts the assumption that the capacity matrix $C$ is a doubly sub-stochastic matrix. ■

As the biased mesh with capacity matrix $\hat{C}$ achieves the throughput $N/(2N-1)$, it then follows from Proposition 15 that the biased mesh is optimal among all the admissible capacity matrices.

# Appendix B

# Uniqueness of the Optimal Capacity Matrix

In this Appendix, we will prove that the biased mesh is the only capacity matrix that achieve the optimal throughput $N/(2N-1)$, and therefore we will be able to establish Proposition 8.

**Lemma 16** *If an admissible capacity matrix $C$ achieves the optimal throughput $N/(2N-1)$, then the capacity matrix $C$ satisfies*

$$\sum_{i=1}^{N} C_{ii} = \frac{N}{2N-1}, \tag{B.1}$$

*and for $k = 1, 2, \ldots, N-1$,*

$$\sum_{i=1}^{N} C_{i(i+k \bmod N)} = \frac{2N}{2N-1}. \tag{B.2}$$

*Proof:* As in the proof of Proposition 7, let

$$x_k = \sum_{i=1}^{N} C_{i(i+k \bmod N)}.$$

If an admissible capacity matrix $C$ achieves the optimal throughput $N/(2N-1)$, then we have from (A.3) and (A.4) that

$$x_0 \geq \frac{N}{2N-1}, \tag{B.3}$$

and for $k = 1, 2, \ldots, N-1$,

$$x_k \geq \frac{2N}{2N-1}. \tag{B.4}$$

If one of the inequalities in (B.3) and (B.4) is strict, then $\sum_{k=0}^{N-1} x_k$ will be strictly larger than $N$ and this will contradict to the assumption that $C$ is admissible. Therefore, we conclude that all the inequalities in (B.3) and (B.4) are in fact equalities. ■

**Lemma 17** *If an admissible capacity matrix $C$ achieves the optimal throughput $N/(2N-1)$, then for any permutation $\sigma$,*

$$\sum_{i \in S_2(\sigma)} (C_{i\sigma(i)} - 2C_{ii}) = 0.$$

*Proof:* Equation (B.1) in Lemma 16 provides $\sum_{i \in S_1(\sigma)} C_{ii} + \sum_{i \in S_2(\sigma)} C_{ii} = N/(2N-1)$ for any permutation $\sigma$. Hence, using Lemma 13, we get

$$\frac{N}{2N-1} \leq \frac{1}{2} + \frac{1}{2N} \min_{\sigma} \left( \frac{N}{2N-1} + \sum_{i \in S_2(\sigma)} (C_{i\sigma(i)} - 2C_{ii}) \right),$$

where the minimum is taken over the set of permutation matrices. Therefore, $0 \leq \min_\sigma \left( \sum_{i \in S_2(\sigma)} (C_{i\sigma(i)} - 2C_{ii}) \right)$, i.e., for any permutation $\sigma$,

$$0 \leq \sum_{i \in S_2(\sigma)} (C_{i\sigma(i)} - 2C_{ii}).$$

We now use the fact that there are exactly $(N-1)!$ permutations $\sigma$ such that $\sigma(i) = j$ for any nodes $i$ and $j$. As a consequence, given a node $i$, there are exactly

$(N-1)!$ permutations $\sigma$ such that $i \notin S_2(\sigma)$, i.e., such that $\sigma(i) = i$. Therefore, there are exactly $N! - (N-1)!$ permutations $\sigma$ such that $i \in S_2(\sigma)$. We can deduce that

$$\begin{aligned}\sum_\sigma \left( \sum_{i \in S_2(\sigma)} (C_{i\sigma(i)} - 2C_{ii}) \right) &= (N-1)! \sum_{i,j \neq i} C_{ij} - 2(N! - (N-1)!) \sum_i C_{ii} \\ &= (N-1)! \sum_{i,j} C_{ij} - (2N! - (N-1)!) \sum_i C_{ii} \\ &= N! - (N-1)! \cdot (2N-1) \cdot \frac{N}{2N-1} = 0,\end{aligned}$$

where we use (B.1) in the last equality. Therefore, given that the sum of all these numbers is 0, and that they were all shown to be nonnegative, this means that they are all null. ■

The next lemma enables us to determine the exact value of the diagonal elements of $C$.

**Lemma 18** *If an admissible capacity matrix $C$ achieves the optimal throughput $N/(2N-1)$, then for all $i$,*

$$C_{ii} = \frac{1}{2N-1}.$$

*Proof:* Pick arbitrarily any node — for instance, node 1 without loss of generality. For any node $j \neq 1$, consider the permutation $\sigma$ such that $\sigma(1) = j$, $\sigma(j) = 1$, and the restriction of $\sigma$ to the other elements is the identity. By Lemma 17, $C_{1j} + C_{j1} = 2(C_{11} + C_{jj})$. Summing over all such $j$'s yields $\sum_{j=2}^N (C_{1j} + C_{j1}) = \sum_{j=2}^N 2(C_{11} + C_{jj})$. Adding $2C_{11}$ on each side of the equation and using (B.1) and (B.2) yields

$$1 + 1 = 2(N-1)C_{11} + \frac{2N}{2N-1}.$$

Hence $C_{11} = \frac{1}{2N-1}$. Since we picked the first node arbitrarily, this is similarly true for any node. ■

**Proposition 19** *The only matrix $C$ that can achieve the optimal throughput $N/(2N-1)$ is the capacity matrix $\hat{C}$ from the biased mesh.*

*Proof:* Combining Lemmas 17 and 18, for any permutation $\sigma$, $\sum_{i \in S_2(\sigma)} C_{i\sigma(i)} = (2 \cdot |S_2(\sigma)|)/(2N-1)$, where $|S_2(\sigma)|$ denotes the number of elements in $S_2(\sigma)$.

Define matrix $D$ such that $D_{ij} = C_{ij}$ for $i \neq j$, and $D_{ii} = 2/(2N-1) = 2C_{ii}$. Then all row and column sums of $D$ are equal to $1 + 1/(2N-1)$ (because $C$ is doubly stochastic). In addition, for any permutation $\sigma$,

$$\begin{aligned}\sum_i D_{i\sigma(i)} &= \sum_{i \in S_1(\sigma)} D_{i\sigma(i)} + \sum_{i \in S_2(\sigma)} D_{i\sigma(i)} = \sum_{i \in S_1(\sigma)} D_{ii} + \sum_{i \in S_2(\sigma)} D_{i\sigma(i)} \\ &= (2 \cdot |S_1(\sigma)|)/(2N-1) + (2 \cdot |S_2(\sigma)|)/(2N-1) = \frac{2N}{2N-1}.\end{aligned}$$

Hence, any permutation on $D$ has the same sum! For any two nodes $i, j$, construct two permutations equal everywhere except on $\{D_{11}, D_{i1}, D_{1j}, D_{ij}\}$. Then $D_{11} + D_{ij} = D_{i1} + D_{1j}$. Therefore, all elements of $D$ can be written as $D_{ij} = D_{i1} + (D_{1j} - D_{11}) = u_i + v_j$, where $u$ and $v$ are two sequences defined on $\{1, ..., N\}$. Since all row and column sums of $D$ are the same, all elements of $D$ are equal; therefore, all non-diagonal elements of $C$ are equal, and finally $C = \hat{C}$. ■

Therefore, we have finally established Proposition 8.

# Appendix C

# Proof that UFS Has 100% Throughput

The following useful lemma characterizes the behavior of a work-conserving server:

**Lemma 20** *([18] p. 7) Consider a work-conserving server and let $A(t)$ and $B(t)$ respectively denote its cumulative number of arrivals and services until time $t$. Assume that its service capacity is one packet per time-slot. Then for all $t \geq 0$,*

$$B(t) = \min_{0 \leq s \leq t} [A(s) + t - s].$$

We are ready to prove that UFS guarantees 100% throughput. Consider a given output $k$. Let $A^k(t)$, $B^k(t)$ and $C^k(t)$ respectively denote the cumulative number of arrivals at the inputs, at the intermediate inputs, and at output $k$ of the load-balanced router, for all packets destined to output $k$.

**Theorem 21** *UFS has the same throughput as an ideal output-queued router, irrespective of the (infinite) arrival process.*

*Proof:* First, note that each input can contain at most $N(N-1)$ packets without having a full frame. Also, the presence of a full frame makes the input work-conserving (within a delay bounded by $N-1$). Finally, there is at most one arrival per time-slot at each input. Therefore, when the input becomes work-conserving, it contains at most $N(N-1)+N-1 = N^2-1$ packets (after services), and its number of packets cannot increase further. When the input is not work-conserving anymore, it is because there was no full frame left, and therefore the number of packets in the input was bounded again by $N(N-1)$ packets. Therefore, there can be at most $N^2-1$ packets in any input at any time (not counting the packet being serviced). Hence, the $N$ inputs contain at most $N^3$ packets, and in particular at most $N^3$ packets destined to output $k$. As a consequence, for any $t \geq 0$,

$$B^k(t) \geq A^k(t) - N^3.$$

(This assumes that packet transmissions are instantaneous. If packet transmission delay is $\tau > 0$, this result becomes $B^k(t) \geq A^k(t-\tau) - N^3$ and the following equations can be generalized accordingly.)

Second, let $B^k_{FF}(t)$ be the number of packets in the full frames completely arrived at the intermediate inputs. For any flow, there are at most $N-1$ packets in the intermediate queues such that their full frame has not yet completely arrived. (For the mesh model allowing simultaneous transmissions, this number is always zero.) Since there are at most $N$ flows going to output $k$,

$$B^k_{FF}(t) \geq B^k(t) - N(N-1) \geq A^k(t) - N^3 - N(N-1).$$

The switching stage is work-conserving whenever there is at least one full frame completely arrived at the intermediate inputs. Therefore, the switching stage will service at least as many packets as an hypothetical work-conserving server of unit capacity whose arrivals would follow $B^k_{FF}(t)$. Hence, using Lemma 20:

$$\begin{aligned} C^k(t) &\geq \min_{0\leq s\leq t}[B^k_{FF}(s) + t - s] \\ &\geq \min_{0\leq s\leq t}[A^k(s) + t - s - N^3 - N(N-1)]. \qquad \text{(C.1)} \end{aligned}$$

We now compare this router with an output-queued router that would have the same arrivals. The behavior of the packets destined to output $k$ in this output-queued router can be modeled by a work-conserving server with arrivals $A^k(t)$ and deterministic constant inter-service time of 1. We will call $C^k_{OQ}(t)$ the cumulative number of services of this server. Then, using Lemma 20, we get

$$C^k_{OQ}(t) = \min_{0 \leq s \leq t} [A^k(s) + t - s]. \tag{C.2}$$

Finally, we can compare Equation C.1 and Equation C.2:

$$C^k(t) \geq \min_{0 \leq s \leq t} [A^k(s) + t - s] - N^3 - N(N-1) = C^k_{OQ}(t) - N^3 - N(N-1).$$

Hence, the number of packets serviced by UFS is within a constant (with respect to $t$) from the number of packets serviced by an output-queued router. This implies that UFS *always* has the same throughput as an output-queued router, irrespective of the infinite arrival process. ■

# Appendix D

# Proof that FOFF Sends Packets in Order

## D.1 Intuition on the FOFF scheme

FOFF has three distinctive features that will be used in the proof. First, for any given flow, packets leave the input stage in order. Second, this scheme is work-conserving for frames, in the sense that every $N$ time-slots, if there is at least one full frame then $N$ packets will be served in the next frame. Finally, the third feature is that if there is no full frame left, then flows are served in round-robin, which avoids starvation when throughput is strictly less than 1.

## D.2 Assumptions

We now assume that we take a snapshot of the router every $N$ time-slots. We will denote this period of $N$ time-slots as a *frame slot*. During every frame slot, each input can send at most one packet to each intermediate input. Similarly, each intermediate input can send at most one packet to each output.

We will assume that at each frame slot, the first input sends first all its packets to the intermediate inputs, then the second input does so, and so on. Of course, the router does not need to operate this way, but this is easier to understand conceptually. There are two possible ways of implementing such an order. A first approach is to define the origin of time differently for each linecard. It is then possible to evaluate the queue occupancy in each linecard in staggered time slots. A second approach is to use shift registers, with no memory speed-up. At time slots $i$, $N+i$, $2N+i$, ..., input linecard $i$ selects an output destination. For instance, assume that this output is $k$. Over the next $N$ time slots, input linecard $i$ sends packets destined to output $k$ in a round-robin order to the $N$ intermediate input linecards. Specifically, in time slot $i+j-1$, input linecard $i$ sends to intermediate input linecard $j$ a packet destined to output $k$. Then, the incoming packets to intermediate input linecard $j$ are delayed by $N-j$ time slots. Similarly, in time slot $j+k-1$, intermediate input linecard $j$ sends a packet destined to output linecard $k$. Once again, the incoming packets to output linecard $k$ are delayed by $N-k$ time slots. This scheme guarantees the *frame slot* snapshot model defined above with a fixed delay of $N$ time slots for a packet to be sent from one linecard stage to the next.

Finally, by convention, we will assume that for every frame slot, the following order occurs: arrivals at the inputs, departures from the inputs, arrivals at the intermediate inputs, and so on. Queue sizes are measured at the end of the frame slot.

## D.3 Notations and Lemmas

A feature of the FOFF algorithm is that each of the three linecard stages can be implemented with only $N$ queues. This is clear for the first two stages. For the third stage, Theorem 25 will show that the reordering buffer can be implemented with at most $N^2$ packets arranged into $N$ queues, one for each intermediate input. In order to prove it, we first introduce notations and consecutively prove several lemmas. In the lemmas, we will show that inputs send approximately the same number of packets to each intermediate input. This results in an occupancy that is approximately the same in all intermediate inputs. Therefore, the reordering, which results from a difference of queue occupancy between intermediate inputs, can be bounded. And consequently, the reordering buffer size at the output can also be bounded.

We will assume in the remainder of the proof that the propagation delay $d$ between the inputs and the intermediate inputs is equal to zero. Similarly, we will also assume that the propagation delay between the intermediate inputs and the outputs is equal

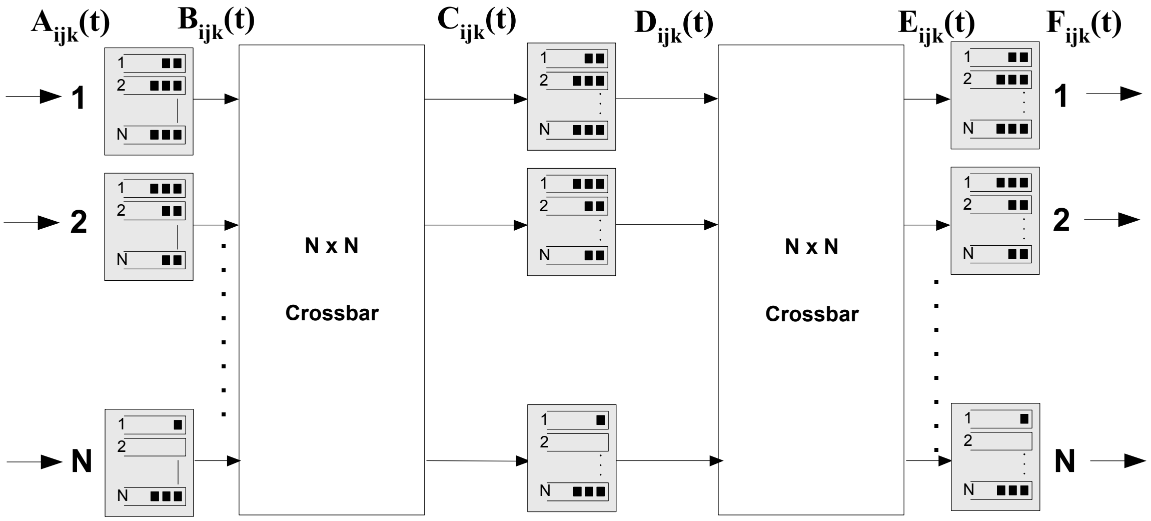


Figure D.1: Proof notations.

to zero as well. If this delay was a positive constant, it would change the total delay by a constant, but would not affect any of the other results.

For $1 \leq i, j, k \leq N$, let $S(i,k)$ denote the set of all packets belonging to the flow from input $i$ to output $k$, and $S(i,j,k)$ be the set of all packets in $S(i,k)$ that are designated to go (or already went) through intermediate input $j$. Clearly $S(i,k) = \bigcup_j S(i,j,k)$. As shown in Figure D.1, let the couples $(A_{ijk}(t),\ B_{ijk}(t))$, $(C_{ijk}(t),\ D_{ijk}(t))$ and $(E_{ijk}(t),\ F_{ijk}(t)))$ denote the cumulative number of (arrivals, departures) of packets in $S(i,j,k)$ at the end of frame slot $t$ at input $i$, intermediate input $j$ and output $k$. For instance, $B_{ijk}(t) - A_{ijk}(t)$ represents the number of packets from $S(i,j,k)$ stored in input linecard $i$. Given these assumptions and notations, we have the following equations.

First, by causality, $A_{ijk}(t) \geq B_{ijk}(t) \geq C_{ijk}(t) \geq D_{ijk}(t) \geq E_{ijk}(t) \geq F_{ijk}(t)$.

Second, the instantaneous transmission implies that $C_{ijk}(t) = B_{ijk}(t)$ and $E_{ijk}(t) = D_{ijk}(t)$.

Third, in input $i$, arriving packets that belong to $S(i,k)$ are sent in round-robin order among all intermediate linecards, starting with intermediate linecard 1. Formally, if $A_{S(i,k)}(t)$ is the cumulative number of arriving packets that belong to $S(i,k)$,

[21] proves that:

$$A_{ijk}(t) = \left\lceil \frac{A_{S(i,k)}(t) + 1 - j}{N} \right\rceil .$$

Therefore, given any tuple $\{i, j_1, j_2, k\}$, the difference between $A_{ij_1k}(t)$ and $A_{ij_2k}(t)$ is bounded by 1, and $A_{ij_1k}(t)$ can be greater only if $j_1 < j_2$. This is a general property of round-robin scheduling. Similarly, since packets from a given flow leave inputs in order, they are also serviced in round-robin, and therefore the same is true for $B_{ij_1k}(t)$ and $B_{ij_2k}(t)$. Given the equality $C(t) = B(t)$, it applies also to $C_{ij_1k}(t)$ and $C_{ij_2k}(t)$. This proves the following lemma:

**Lemma 22** *For each flow, the difference in the cumulative number of arrivals at two different intermediate inputs is bounded by* 1*. In addition, if an intermediate input has more arrivals than another one, then its index must be lower.*

Let $Q_{jk}(t)$ be the occupancy of the VOQ at intermediate input $j$ for output $k$ at the end of frame slot $t$. Given Lemma 22, the next lemma will bound the difference in occupancy for a given output between different intermediate input linecards.

**Lemma 23** *For all intermediate inputs $j_1, j_2$, output $k$ and frame slot $t$:*

$$|Q_{j_1k}(t) - Q_{j_2k}(t)| \leq N.$$

*Proof:* Assume without loss of generality that $j_1 < j_2$. Since the link between intermediate input $j_1$ and output $k$ is work-conserving at each frame slot, we have [18]:

$$Q_{j_1k}(t) = \max_{0 \leq s \leq t} \left( \sum_i [C_{ij_1k}(t) - C_{ij_1k}(s)] - (t - s) \right).$$

For instance, this maximum could be reached for some $s'$:

$$Q_{j_1k}(t) = \sum_i [C_{ij_1k}(t) - C_{ij_1k}(s')] - (t - s').$$

Similarly,

$$\begin{aligned} Q_{j_2k}(t) &= \max_{0\le s\le t}(\sum_i [C_{ij_2k}(t) - C_{ij_2k}(s)] - (t-s)) \\ &= \sum_i [C_{ij_2k}(t) - C_{ij_2k}(s'')] - (t-s''). \end{aligned}$$

The first consequence is:

$$\begin{aligned} Q_{j_2k}(t) &\ge \sum_i [C_{ij_2k}(t) - C_{ij_2k}(s')] - (t-s') \\ &= \sum_i [C_{ij_2k}(t) - C_{ij_2k}(s')] + \{Q_{j_1k}(t) - \sum_i [C_{ij_1k}(t) - C_{ij_1k}(s')]\} \\ &= Q_{j_1k}(t) + \sum_i \{[C_{ij_2k}(t) - C_{ij_1k}(t)] + [C_{ij_1k}(s') - C_{ij_2k}(s')]\} \\ &\ge Q_{j_1k}(t) + \sum_i \{-1+0\} \\ &= Q_{j_1k}(t) - N. \end{aligned}$$

Similarly, the second consequence is:

$$\begin{aligned} Q_{j_1k}(t) &\ge Q_{j_2k}(t) + \sum_i \{[C_{ij_1k}(t) - C_{ij_2k}(t)] + [C_{ij_2k}(s'') - C_{ij_1k}(s'')]\} \\ &\ge Q_{j_2k}(t) + \sum_i \{0-1\} \\ &= Q_{j_2k}(t) - N. \end{aligned}$$

Hence $|Q_{j_1k}(t) - Q_{j_2k}(t)| \le N$. ■

Since packets from a given flow can traverse different intermediate input linecards, it is possible that packets will arrive at the output linecards out-of-order. The next lemma bounds the amount of reordering for a given flow.

**Lemma 24** *If two packets $p_1$ and $p_2$ belong to $S(i,k)$, and $p_1$ arrives at input $i$ before $p_2$, then $p_1$ will arrive at output $k$ at most $N$ frame slots after $p_2$.*

*Proof:* If a packet $p_1$ from $S(i,k)$ arrives at intermediate input $j_1$ at $t_1$, it will leave the intermediate input at $t_1 + Q_{j_1k}(t_1)$ because of the work-conserving property mentioned above. Since packets of the same flow leave the input stage in order, a packet $p_2$ from $S(i,k)$ arriving later at input $i$ will thus arrive at intermediate input $j_2$ at $t_2 \geq t_1$. As a result, $p_2$ will leave $j_2$ at

$$\begin{aligned} t_2 + Q_{j_2k}(t_2) &= t_1 + Q_{j_2k}(t_1) + \\ &\quad [t_2 - t_1 + Q_{j_2k}(t_2) - Q_{j_2k}(t_1)] \\ &\geq t_1 + Q_{j_2k}(t_1) \end{aligned}$$

(by the work-conserving property). Hence $p_2$ will not leave the intermediate input before $t_1 + Q_{j_1k}(t_1) + (Q_{j_2k}(t_1) - Q_{j_1k}(t_1)) \geq t_1 + Q_{j_1k}(t_1) - N$. Therefore $p_1$ will leave the intermediate stage at most $N$ frame slots after $p_2$ leaves. ■

## D.4 Theorem

The following theorem shows how the reordering buffer can be implemented with only $N$ queues and $N^2$ packets.

**Theorem 25** *If there are $N^2 + 1$ packets in the queue, the head-of-line packet of at least one of the $N$ queues can be sent while keeping packets in order. Therefore, the reordering buffer size needs to be at most $N^2$ ($N^2 + 1$ if including the packet currently being serviced) in order for packets to leave the router in order.*

*Proof:* As assumed above, at output $k$, during a given frame slot, we first have up to $N$ arrivals from the $N$ intermediate inputs, and then we have up to $N$ departures. We will of course assume that packets can only depart in order.

We will show that whenever the number of packets queued in a given output $k$ is strictly bigger than $N^2$, the output is work-conserving, i.e., the head-of-line packet of at least one of the $N$ queues can depart while keeping packets ordered. Since the number of arrivals cannot exceed the number of departures when the output is work-conserving, this would clearly show that the queue occupancy of output $k$ is bounded by $N^2$. As a consequence, we would only need a reordering buffer size of at most $N^2$ in order for packets to leave the router in order.

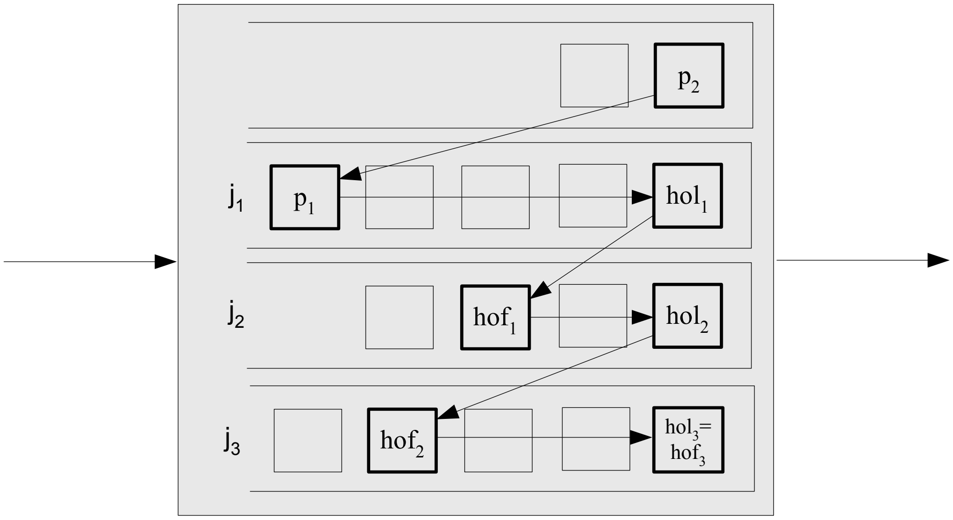


Figure D.2: View of the reordering buffer of output $k$. In this example, the head of line of queue $j_3$ is also head of flow, and therefore can leave the output without any packet reordering

Assume that there are more than $N^2$ packets queued in a given output $k$ at frame slot $t$. Since at most $N$ packets could arrive at output $k$ in one frame slot, by the pigeonhole principle at least one of the packets queued did not arrive in the last $N$ frame slots. Call this packet $p_2$, and let $i$ be such that $p_2 \in S(i,k)$. By Lemma 24, the *head of flow* $p_1$ of $p_2$ is also necessarily queued in output $k$, in some queue $j_1$. Therefore, there is at least one *head of flow* queued in output $k$ if there are more than $N^2$ packets in output $k$.

We now show that there is at least one head of flow in the router that is also head of line of one of the queues in output $k$. This will prove that the head-of-line packet of at least one of the $N$ queues can depart while keeping packets in order.

Given the above assumption, if packet $p_1$ is head-of-line for its intermediate input $j_1$, then $p_1$ can depart. Otherwise, consider the head-of-line for $j_1$, $hol_1$, which has already arrived at output $k$ (illustrated in Figure D.2). If $hol_1$ is a head of flow, it can depart. Otherwise, let $hof_1$ be the head of the flow of $hol_1$, which went through some intermediate input $j_2$. We now show that $hof_1$ is queued in output $k$. We know that $hof_1$ arrived at its intermediate input no later than $hol_1$, because they are from the same flow. In addition, $hol_1$ arrived before $p_1$, because they went through the same intermediate input. Finally, $p_1$ arrived no later than $p_2$ (same flow). Therefore, $hof_1$ reached the intermediate inputs before $p_2$. By Lemma 23, since $p_2$ arrived at output $k$ by frame slot $t-N$, $hof_1$ must have arrived at output $k$ by frame slot $t$. Therefore, $hof_1$ must be queued in output $k$.

Again, if $hof_1$ is the head-of-line for its intermediate input $j_2$, $hof_1$ can depart. Otherwise, consider $hol_2$, the head of line for $hof_1$. If $hol_2$ is head of flow, it can depart. Otherwise, consider the head of flow for $hol_2$, $hof_2$, which went through some intermediate input $j_3$. By repeating the same method, we can continue considering the head of flow of the head of line for successive intermediate inputs $j_1, j_2, j_3, ...$. Since each new head of flow arrived strictly before the one previously considered, the algorithm cannot visit the same intermediate input twice. Given that there are only $N$ intermediate inputs, the algorithm must converge towards a packet that is both head of flow and head of line. This packet can then depart while keeping packets in order. ■

# Appendix E

# Proof of Average Packet Delay with FOFF

Our objective is to prove that the average packet delay through a load-balanced router using FOFF is within a constant of the average packet delay through a (work-conserving) output-queued router, assuming that both are defined.

In the proof, we will consider some arrival sequence $A(t)$. Given this arrival sequence, we will assume that the average delay through a work-conserving router is defined and finite, and will attempt to prove that, if defined, the average delay through the load-balanced router is within a constant of the average delay for a work-conserving router. The proof will rely on several lemmas. These lemmas will consider the effect of using delay lines on the departures from a work-conserving router, and try to compare the load-balanced router with work-conserving routers using delay lines.

Throughout the lemmas, let $B(t)_{\{A(t)\}}$ denote the cumulative number of departures in a router with cumulative arrival traffic $A(t)$. For instance, by Lemma 20, in a work-conserving router (noted $WC$), the cumulative number of departures $B(t)^{WC}_{\{A(t)\}}$ satisfies:

$$B(t)^{WC}_{\{A(t)\}} = \min_{0 \leq s \leq t} [A(s) + t - s]. \tag{E.1}$$

Two routers $S^1$ and $S^2$ are defined as *equal* if they have the same number of packet departures for the same arbitrary packet arrivals, irrespective of the packet order. In other words, the two routers are equal if their respective cumulative numbers of departures satisfy for all sequences $\{A(t)\}$:

$$B(t)^1_{\{A(t)\}} = B(t)^2_{\{A(t)\}}.$$

Similarly, a router $S^1$ is said to be *better than* another router $S^2$ if for any arrival traffic $A(t)$, and for any time-slot, $S^1$ has at least as many departures as $S^2$. This will be noted as

$$B(t)^1_{\{A(t)\}} \geq B(t)^2_{\{A(t)\}}.$$

For instance, a work-conserving router is known to be better than any router of output capacity 1.

The following lemma will prove that it is possible to permute work-conserving routers and delay lines, and still get equal systems.

**Lemma 26** *Consider a first system $S^1$ comprising a work-conserving router followed by delay lines of delay $d$, noted as $S^1 = (WC, DL(d))$ (where $DL(d)$ represents a Delay Line of duration $d \geq 0$). Similarly, consider a second system $S^2$ comprising delay lines of delay $d$ followed by a work-conserving router, noted as $S^2 = (DL(d), WC)$. Then $S^1$ and $S^2$ are equal.*

*Proof:* Let $A$ denote the cumulative number of arrivals at each router, with $A(0) = 0$. Let $B_1$ (resp. $B_2$) denote the cumulative number of departures from each system. For $t \geq d$,

$$B_1(t) = B(t-d)^{WC}_{\{A(t)\}} = \min_{0 \leq s \leq t-d} [A(s) + (t-d) - s],$$

and

$$B_2(t) = B(t)^{WC}_{\{A(t-d)\}} = \min_{0 \leq s \leq t} [A(s-d) + t - s].$$

But then, using $u = s - d$,

$$B_2(t) = \min_{-d \leq u \leq t-d} [A(u) + t - (u+d)] = \min_{-d \leq u \leq t-d} [A(u) + (t-d) - u].$$

Comparing this with the expression of $B_1(t)$, we thus just need to show that for $u < 0$,

$A(u) + (t - d) - u \geq A(0) + (t - d) - 0$, in order to prove that $B_2(t) = B_1(t)$. Since for $u < 0$, $A(u) = A(0) = 0$, the result follows. ■

The following lemma will help us compare tandems of routers with tandems of work-conserving routers.

**Lemma 27** *Consider a system composed of two routers $S^1$ and $S^2$. Assume that for any arrival traffic $A(t)$, $S^1$ is better than $WC$, i.e.,*

$$B(t)^1_{\{A(t)\}} \geq B(t)^{WC}_{\{A(t)\}}. \tag{E.2}$$

*Also assume that $S^2$ behaves better than $WC$ when their arrivals are the departures from $S^1$. In other words, if the departures from $S^1$ are written as*

$$D(t) = B(t)^1_{\{A(t)\}}, \tag{E.3}$$

*then*

$$B(t)^2_{\{D(t)\}} \geq B(t)^{WC}_{\{D(t)\}}. \tag{E.4}$$

*Then the tandem $(S^1, S^2)$ behaves better than a tandem of two work-conserving routers $(WC, WC)$.*

*Proof:* We assumed that $B(t)^2_{\{D(t)\}} \geq B(t)^{WC}_{\{D(t)\}}$. The first term of Inequality (E.4) denotes the departures from the tandem $(S^1, S^2)$, while the second term denotes the departures from the tandem $(S^1, WC)$. Thus, since we know that $(S^1, S^2)$ is better than $(S^1, WC)$, we only need to show that $(S^1, WC)$ is better than $(WC, WC)$ in order to prove the lemma. However, we also know that $S^1$ has more departures than $WC$ given the same arrival pattern. These departures are then used as the arrivals at the second $WC$ in both tandems. By Equation (E.1), a $WC$ having more arrivals will have at least as many departures, therefore $(S^1, WC)$ will have at least as many departures than $(WC, WC)$. This proves the lemma. ■

The following lemma considers a system that is only work-conserving when its queue is above a given capacity. This lemma is useful, for instance, for the analysis of a system that only services packets by bursts, as in [44].

**Lemma 28** *Consider a system that satisfies the following two properties. First, there exists a critical queue size $Q_c$ such that the router is always work-conserving when at least $Q_c$ packets are queued. Second, whenever $Q$ packets are queued with $Q < Q_c$, the router takes at most $T$ time-slots to send $Q$ packets. Then the system is better than a tandem $(WC, DL(T))$ of a work-conserving router and a delay line of delay $T$.*

*Proof:* Let $Q(t)_{\{A(t)\}}$ (resp. $Q(t)^{WC}_{\{A(t)\}}$) represent the queue size of the system (resp. of a work-conserving router) at time-slot $t$ under arrivals $A(t)$.

We now prove by contradiction that $B(t)^{WC}_{\{A(t)\}} - B(t)_{\{A(t)\}} \leq Q_c - 1$ for all $t$. In other words, we assume that $t_0$ is the first time-slot when this inequality is not satisfied, and we prove that $t_0$ cannot exist. If $t_0$ exists:

$$B(t_0)^{WC}_{\{A(t)\}} - B(t_0)_{\{A(t)\}} \geq Q_c$$

and

$$B(t_0 - 1)^{WC}_{\{A(t)\}} - B(t_0 - 1)_{\{A(t)\}} \leq Q_c - 1.$$

In addition, given the work-conserving property (Equation (E.1)), we get

$$B(t_0)^{WC}_{\{A(t)\}} \leq B(t_0 - 1)^{WC}_{\{A(t)\}} + 1,$$

with equality only if there is at least one packet queued in the work-conserving system to service at time $t_0$. Putting these three inequalities together,

$$B(t_0 - 1)_{\{A(t)\}} \geq B(t_0)_{\{A(t)\}}.$$

But since the cumulative number of departures cannot decrease, these two quantities have to be equal, and therefore there had to be at least one packet queued in the work-conserving system just before the departures at time $t_0$. As a consequence, the queue size in the system just before the departures at $t_0$ is:

$$\begin{aligned}
& A(t_0) - B(t_0-1)_{\{A(t)\}} \\
= \;& [A(t_0) - B(t_0-1)^{WC}_{\{A(t)\}}] + [B(t_0-1)^{WC}_{\{A(t)\}} - B(t_0-1)_{\{A(t)\}}] \\
\geq \;& 1 + [Q_c - 1] = Q_c. \qquad \text{(E.5)}
\end{aligned}$$

But then the system should be work-conserving by assumption, and therefore $B(t_0 - 1)_{\{A(t)\}} \neq B(t_0)_{\{A(t)\}}$, which contradicts the assumption.

By the proof above, $B(t)^{WC}_{\{A(t)\}} - B(t)_{\{A(t)\}} \leq Q_c - 1$ for all $t$. In addition, the difference between these queue sizes can be serviced in at most $T$ time-slots. (Note that the packets in the difference will always be available to be serviced because $Q(t)_{\{A(t)\}} = A(t) - B(t)_{\{A(t)\}} \geq B(t)^{WC}_{\{A(t)\}} - B(t)_{\{A(t)\}}$.) As a consequence, $B(t)^{WC}_{\{A(t)\}} \leq B(t+T)_{\{A(t)\}}$ for all $t$, hence $B(t-T)^{WC}_{\{A(t)\}} \leq B(t)_{\{A(t)\}}$ for all $t$. ■

The following lemma reminds us of the fact that having more packets leave earlier implies having a lower average delay.

**Lemma 29** *Consider an infinite traffic arrival sequence $A(t)$. Assume that the average delays for a work-conserving router and some router $S$ are both defined.*[1] *Assume also that $S$ satisfies $B(t)_{\{A(t)\}} \geq B(t)^{WC}_{\{A(t)\}}$. Then the average delay for $S$ is at most the average delay for the work-conserving router.*

*Proof:* When defined, the average packet delay does not depend on the packet order. This is because the total packet delay until some time $t$ is just the integral of the difference between $A(t)$ and $B(t)_{\{A(t)\}}$ (resp. $B(t)^{WC}_{\{A(t)\}}$) between 0 and $t$. Therefore, we can assume that the packet order is the same in both routers without loss of generality. Packets always leave the router earlier than (or at the same time as) the work-conserving router. Therefore, the result follows. ■

**Theorem 30** *Consider an infinite traffic arrival sequence $A(t)$. Assume that the average delays for a work-conserving router and a load-balanced router using FOFF are both defined. Then the average packet delay through a load-balanced router using*

[1] If the average delay for a router does not exist, at least a limsup exists and satisfies the same properties.

*FOFF is within a constant of the average packet delay through a (work-conserving) output-queued router.*

*Proof:* First, each input of the load-balanced router services one packet per time-slot whenever this input has at least one full frame of $N$ packets. By the pigeonhole principle, this necessarily happens whenever the input has at least $N(N-1)+1$ packets. In addition, whenever the input has at most $Q = N(N-1)$ packets (before arrivals), it takes at most $N^2$ time-slots to service at least $Q$ packets. Therefore, by Lemma 28, any input linecard is better than a tandem of a work-conserving router and a delay line of delay $N^2$. But since any input can have at most one arrival and one departure per time-slot, the work-conserving router is just a forwarding router. Thus any input linecard is better than a delay line of delay $N^2$.

Similarly, consider the system formed by all packets destined to a given output $k$ in the intermediate linecards. Applying the pigeonhole principle as above, if this set has $N(N-1)+1$ packets, all intermediate inputs will be non-empty (because of Lemma 23) and the system will be work-conserving for output $k$. And, as above, whenever this system has at most $Q = N(N-1)$ packets (before arrivals), it takes at most $N^2$ time-slots to service at least $Q$ packets.

Finally, Theorem 25 proves that an output linecard is work-conserving whenever at least $Q_c = N^2 + 1$ packets are queued. Assume that an output contains $Q < Q_c$ packets. For each packet $p$ in the output linecard that arrived at the output linecard at time $t - d$, where $d \geq 0$, any packet ahead of $p$ in the flow order of $p$ will arrive no later than $t - d + N^2$ at the output (Lemma 24). Then, by time $t - d + N^2$, either $p$ departs, or it is waiting to depart but another packet is serviced at each time-slot until it departs, i.e., the output is work-conserving until $p$ departs. Therefore, the output services at least as many packets as a work-conserving router that would accept any such packet $p$ at time $t - d + N^2$. In other words, whenever the output linecard has at most $Q = N^2$ packets, it takes at most $N^2$ time-slots to service at least $Q$ packets (not necessarily the same). As with the inputs, any output is therefore better than a delay line of delay $N^2$.

Therefore, we can use the above results to analyze the load-balanced router. Consider the system formed by all packets destined to a given output $k$ in the intermediate linecards. We found above that this system behaves better in the load-balanced router than a work-conserving router followed by a delay line of $N^2$. In addition, the inputs and the output $k$ add two delay lines of delay $N^2$ using their respective arrivals. We can then use Lemma 27 and the fact that $B(t)^{WC}_{\{A(t)\}}$ is independent of the input at which packets arrive in a work-conserving router in order to prove that the load-balanced router works better than a tandem $(DL(N^2), WC, DL(N^2), DL(N^2))$. By Lemma 26, it works better than $(WC, DL(3N^2))$. Finally, by Lemma 29, if the average delay for a work-conserving router is defined, then the average delay for the load-balanced router is within $3N^2$. ■

# Appendix F

# Proofs for the Linecard Schedule

In this appendix, we will prove that given any kind of valid schedule among L-L, L-G or G-G, we can construct the other two. In particular, we will prove the validity of the algorithm that builds a G-G schedule out of an L-G schedule.

## F.1 Proof for Theorem 11

First, as shown in 4.4.4, it is easy to successively build a valid L-G schedule from a valid L-L schedule and a valid G-G schedule from a valid L-G schedule.

On the other hand, 4.4.6 shows the algorithm that successively constructs a valid L-G schedule from a valid G-G schedule and a valid L-L schedule from a valid L-G schedule. This is possible as long as we can decompose the matrix into a sum of permutations (for example, using a Birkhoff-von Neumann decomposition), which we can always do when the sums on each row and each column are equal [19, 12].

## F.2 Proofs for the Construction of the Valid G-G Schedule

*Proof:* We will now prove the validity of the algorithm that builds a G-G schedule out of an L-G schedule. For this purpose, we will operate by reverse induction on $t = N, N-1, ..., 1$.

Assume that for a given $t$,

$$\begin{cases} \sum_{j'=1}^{G} M_{ij'}^t = L_i t & \text{for all } i \\ \sum_{i'=1}^{G} M_{i'j}^t = L_j t & \text{for all } j \\ 0 \leq M_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t & \text{for all } i,j \end{cases}$$

This is obviously true for $t = N$ by definition of $M^t$. We will show at the end of the proof that if we assume it for $t$, it's also true for $t-1$.

First, since $Q^t = M^t - P^t$, using the definition of $P^t$ we get:

$$\begin{cases} (\sum_{j'=1}^{G} Q_{ij'}^t) \bmod t = 0 & \text{for all } i \\ (\sum_{i'=1}^{G} Q_{i'j}^t) \bmod t = 0 & \text{for all } j \\ 0 \leq Q_{ij}^t \leq t-1 & \text{for all } i,j \end{cases}$$

Therefore, $a^t$ and $b^t$ are integer vectors.

Second, from the definition of $a^t$ and $b^t$, they both have the same sum — we will call it $\sigma$.

Third, we now prove that there exists a matrix $R^t$ satisfying the conditions above. In order to prove this, we can use the result from exercise 3.13 in [73], which is based on König's theorem [55]. [73] states that $R^t$ exists iff for each subset $s_1$ of the rows and for each subset $s_2$ of the columns, $\sigma + |E(s_1,s_2)| \geq \sum_{i \in s_1} a_i + \sum_{j \in s_2} b_j$, where $|E(s_1,s_2)|$ denotes the number of non-zero elements $Q_{ij}^t$ with $i \in s_1, j \in s_2$. But we know that

$$\sum_{i \in s_1, j \in s_2} Q_{ij}^t = \sum_{i \in s_1} Q_{ij}^t - \sum_{i \in s_1, j \notin s_2} Q_{ij}^t$$

$$\sum_{i \in s_1, j \in s_2} Q_{ij}^t \geq t \sum_{i \in s_1} a_i - \sum_{j \notin s_2} Q_{ij}^t$$

$$\sum_{i \in s_1, j \in s_2} Q_{ij}^t \geq t \sum_{i \in s_1} a_i - t \sum_{j \notin s_2} b_j$$

In addition, since $t-1 \geq Q_{ij}^t$,

$$\sum_{i \in s_1, j \in s_2} t\delta_{Q_{ij}^t \geq 0} \geq \sum_{i \in s_1, j \in s_2} Q_{ij}^t,$$

therefore

$$\sum_{i\in s_1, j\in s_2} t\delta_{Q^t_{ij}\geq 0} \geq t\sum_{i\in s_1} a_i - t\sum_{j\notin s_2} b_j.$$

Since $|E(s_1, s_2)| = \sum_{i\in s_1, j\in s_2} \delta_{Q^t_{ij}\geq 0}$,

$$|E(s_1, s_2)| \geq \sum_{i\in s_1} a_i - \sum_{j\notin s_2} b_j = \sum_{i\in s_1} a_i + \sum_{j\in s_2} b_j - \sigma.$$

Thus $R^t$ exists. Note that it is possible to construct it in polynomial time using Ford-Fulkerson (Appendix F.3).

Fourth, we now show that the resulting schedule $S^t$ will satisfy the MEMS constraint, i.e., for all $i, j$,

$$S^t_{ij} \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil .$$

We know that

$$S^t_{ij} = P^t_{ij} + R^t_{ij} = \left\lfloor \frac{1}{t} M^t_{ij} \right\rfloor + R^t_{ij},$$

$$R^t_{ij} \leq Q^t_{ij} = M^t_{ij} - t\left\lfloor \frac{1}{t} M^t_{ij} \right\rfloor ,$$

$$R^t_{ij} \leq 1,$$

and

$$M^t_{ij} \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t.$$

Distinguish two cases regarding this last inequality. If this inequality is an equality,

$$M^t_{ij} = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t,$$

so $R^t_{ij} \leq Q^t_{ij} = 0$ and $S^t_{ij} = M^t_{ij}/t = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil$. Otherwise, this inequality is strict,

$$M^t_{ij} < \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t,$$

so there exists $\epsilon > 0$ such that

$$M_{ij}^t/t = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - \epsilon = (\left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - 1) + (1 - \epsilon).$$

Thus,

$$\left\lfloor \frac{1}{t} M_{ij}^t \right\rfloor = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - 1 + \lfloor 1 - \epsilon \rfloor \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - 1$$

and

$$S_{ij}^t \leq \left\lfloor \frac{1}{t} M_{ij}^t \right\rfloor + R_{ij}^t \leq \left\lfloor \frac{1}{t} M_{ij}^t \right\rfloor + 1 \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil.$$

Note that in both cases

$$S_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil.$$

Fifth, we complete the recursion hypothesis and show that:

$$\begin{cases} (i)\ \sum_{j'=1}^{G} M_{ij'}^{t-1} = L_i(t-1) & \text{for all } i \\ (ii)\ \sum_{i'=1}^{G} M_{i'j}^{t-1} = L_j(t-1) & \text{for all } j \\ (iii)\ 0 \leq M_{ij}^{t-1} \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil (t-1) & \text{for all } i, j \end{cases}$$

We will use the assumptions on $M$ stated at the start of the proof, and the definition $M^{t-1} = M^t - S^t$.

Let us first prove $(i)$ by showing that

$$\sum_{j'=1}^{G} S_{ij'}^t = L_i$$

(the proof for $(ii)$ is similar). By definition,

$$\sum_{j'=1}^{G} S_{ij'}^t = \sum_{j'=1}^{G} (P_{ij'}^t + R_{ij'}^t),$$

and

$$\sum_{j'=1}^{G} R_{ij'}^t = a_i = (\sum_{j'=1}^{G} Q_{ij'}^t)/t,$$

thus

$$\sum_{j'=1}^{G} S_{ij'}^{t} = \sum_{j'=1}^{G} (tP_{ij'}^{t} + Q_{ij'}^{t})/t = \sum_{j'=1}^{G} M_{ij'}^{t}/t = L_i.$$

Let us now prove $(iii)$. We know that $M_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t$, therefore $P_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil$. Also we can decompose $M_{ij}^t$ in base $t$ as $M_{ij}^t = P_{ij}^t t + Q_{ij}^t$, and

$$\begin{aligned} M_{ij}^{t-1} &= M_{ij}^t - S_{ij}^t \\ &= (P_{ij}^t t + Q_{ij}^t) - (P_{ij}^t + R_{ij}^t) \\ &= P_{ij}^t (t-1) + (Q_{ij}^t - R_{ij}^t). \end{aligned}$$

Distinguish two cases. In the case where

$$P_{ij}^t = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil,$$

then

$$M_{ij}^t = P_{ij}^t t, \;\; Q_{ij}^t = 0, \;\; R_{ij}^t = 0,$$

thus

$$M_{ij}^{t-1} = P_{ij}^t (t-1) = M_{ij}^t \frac{t-1}{t} \leq \left\lceil \frac{L_i L_j}{N} \right\rceil (t-1).$$

Otherwise, in the case where

$$P_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - 1,$$

we have

$$\begin{aligned} M_{ij}^{t-1} &= P_{ij}^t (t-1) + (Q_{ij}^t - R_{ij}^t) \\ &\leq (\left\lceil \tfrac{L_i \cdot L_j}{N} \right\rceil - 1)(t-1) + Q_{ij}^t \\ &\leq \left\lceil \tfrac{L_i \cdot L_j}{N} \right\rceil (t-1) - (t-1) + (t-1) \\ &= \left\lceil \tfrac{L_i L_j}{N} \right\rceil (t-1), \end{aligned}$$

since $Q_{ij}^t \leq t-1$ as shown before. Hence $(iii)$ is correct in both cases and the three properties are proven by induction.

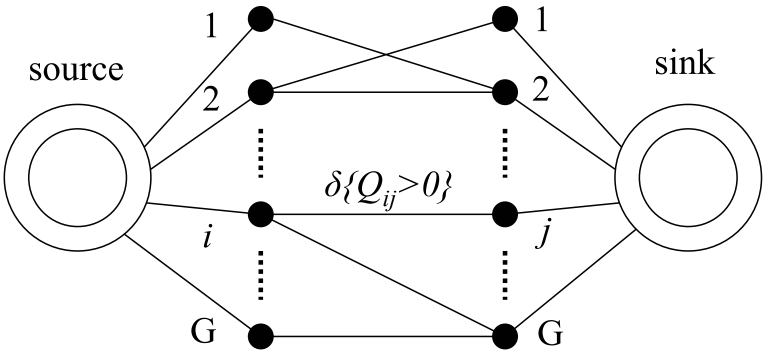


Figure F.1: Illustration of the Ford-Fulkerson construction.

Finally, note that all parts of the algorithm run in polynomial time, and thus the algorithm also operates in polynomial time. ■

## F.3 Ford-Fulkerson Algorithm

As explained above, it is possible to use Ford-Fulkerson's max-flow algorithm [36] in order to construct $R^t$, and therefore the schedules $S^t$. More specifically, as illustrated in Figure F.1, construct the network as follows. There is one source, $G$ inputs, $G$ outputs, and one sink. The source is connected to each input $i$ with capacity $a_i$. Each input $i$ is connected to each output $j$ with capacity $\delta_{Q_{ij} \geq 1}$, i.e., capacity 1 if $Q_{ij} \geq 1$ and 0 otherwise. Finally, each output $j$ is connected to the sink with capacity $b_j$. Since capacities are integer, the resulting flows will also be integers, and will thus yield a correct matrix $R^t$.

# Bibliography

[1] N. Alon, "A simple algorithm for edge-coloring bipartite multigraphs," *Information Processing Letters*, Vol. 85, No. 6, pp. 301–302, March 2003. (*p. 65*)

[2] M. Ajmone Marsan, A. Bianco, P. Giaccone, E. Leonardi and F. Neri, "Packet scheduling in input-queued cell-based switches," *Proc. of IEEE Infocom '01*, Vol. 2, pp. 1095–1103, Anchorage, Alaska, April 2001. (*p. 9*)

[3] T.E. Anderson, S.S. Owicki, J.B. Saxe and C.P. Thacker, "High speed switch scheduling for local area networks," *ACM Trans. on Computer Systems*, Vol. 11, No. 4, pp. 319–352, November 1993. (*p. 9*)

[4] ANSI (American National Standards Institute), T1.TR.68-2001, "Enhanced network survivability performance," February 2001. (*p. 67*)

[5] S. Arekapudi, "Feasibility study of configuration algorithms for the switch fabric of a load-balanced switch with an arbitrary number of linecards," *Stanford University HPNG Technical Report TR04-HPNG-032305*, Stanford, CA, March 2004. (*p. 68*)

[6] S. Arekapudi, S.-T. Chuang, I. Keslassy and N. McKeown, "Configuring a load-balanced switch in hardware," *under review.* (*p. 68*)

[7] F. Baccelli and P. Bremaud, *Elements of Queueing Theory*, Springer-Verlag, New York, 1994. (*pp. 7 and 8*)

[8] F. Baker, "Requirements for IP Version 4 Routers", RFC 1812, June 1995, available at http://www.faqs.org/rfcs/rfc1812.html. (*pp. 10, 32 and 46*)

[9] S. Beckett, *En attendant Godot*, 1952. (English translation: *Waiting for Godot*, 1954). (*p. 38*)

[10] J.C.R. Bennett, C. Partridge and N. Shectman, "Packet reordering is not pathological network behavior," *IEEE/ACM Transactions on Networking*, Vol. 7, No. 6, pp. 789–798, December 1999. (*pp. 10 and 32*)

[11] P. Bernasconi, C. Doerr, C. Dragone, M. Capuzzo, E. Laskowski and A. Paunescu, "Large N x N waveguide grating routers", *Journal of Lightwave Technology*, Vol. 18, No. 7, pp. 985–991, July 2000. (*p. 47*)

[12] G. D. Birkhoff, "Tres observaciones sobre el algebra lineal," *Universidad Nacional de Tucuman Revista*, Serie A, Vol. 5, pp. 147–151, 1946. (*pp. 26, 65 and 106*)

[13] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," *ACM Computer Communication Review*, Vol. 32, No. 1, pp. 20–30, January 2002. (*pp. 11 and 32*)

[14] A. Broder and M. Mitzenmacher, "Using multiple hash functions to improve IP lookups," *Proc. of IEEE Infocom '01*, pp. 1454–1463, 2001. (*p. 34*)

[15] Z. Cao, Z. Wang and E. Zegura, "Performance of hashing-based schemes for internet load balancing," *Proc. of IEEE Infocom '00*, pp.332–341, Tel Aviv, Israel, March 2000. (*p. 34*)

[16] G. Chandranmenon and G. Varghese, "Trading packet headers for packet processing," *IEEE/ACM Transactions on Networking*, Vol. 4, No. 2, pp. 141–152, April 1996. (*p. 34*)

[17] T. Chaney, J. A. Fingerhut, M. Flucke and J. S. Turner, "Design of a Gigabit ATM Switch," *Proc. of IEEE Infocom '97*, pp. 1801–1809, April 1997. (*p. 2*)

[18] C.-S. Chang, *Performance Guarantees in Communication Networks*, Springer-Verlag, New York, 2000. (*pp. 7, 8, 88 and 94*)

[19] C.-S. Chang, J.-W. Chen and H.-Y. Huang, "On service guarantees for input-buffered crossbar switches: a capacity decomposition approach by Birkhoff and Von Neumann," *IEEE IWQoS, London*, 1999. (*pp. 26, 65 and 106*)

[20] C.-S. Chang, D.-S. Lee and Y.-S. Jou, "Load balanced Birkhoff-von Neumann switches, Part I: one-stage buffering," *Computer Communications*, Vol. 25, No. 6, pp. 611–622, 2002. (*pp. 2, 4, 7 and 9*)

[21] C.-S. Chang, D.-S. Lee and C.-M. Lien, "Load balanced Birkhoff-von Neumann switches, Part II: multi-stage buffering," *Computer Communications*, Vol. 25, No. 6, pp. 623–634, 2002. (*pp. 2, 33 and 94*)

[22] C.-S. Chang, D.-S. Lee and C.-Y. Yue, "Providing guaranteed rate services in the load balanced Birkhoff-von Neumann switches," *IEEE Infocom '03*, San Francisco, CA, 2003. (*pp. 33 and 76*)

[23] H.J. Chao, S.Y. Liew and Z. Jing, "A dual-level matching algorithm for 3-stage Clos-network packet switches," *Hot Interconnects XI*, Stanford, CA, August 2003. (*p. 2*)

[24] F. Chiussi and A. Francini, "A distributed scheduling architecture for scalable packet switches," *IEEE Journal on Selected Areas in Communications*, Vol. 18, No. 12, pp. 2665–2683, December 2000. (*p. 2*)

[25] F. Chiussi, J. Kneuer, and V. Kumar, "Low-cost scalable switching solutions for broadbandnetworking: the ATLANTA architecture and chipset," *IEEE Communications Magazine*, Vol. 35, No. 12, pp. 44–53, December 1997. (*p. 2*)

[26] R. Cole, K. Ost and S. Schirra, "Edge-coloring bipartite multigraphs in O(E log D) time," *Combinatorica*, Vol. 21, pp. 5–12, 2001. (*p. 65*)

[27] R. Cole, K. Ost and S. Schirra, "Edge-coloring bipartite multigraphs in O(E log D) time," *New York University Technical Report NYU-TR1999-792*, New York, September 1999. (*p. 65*)

[28] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," *Proc. of IEEE Infocom '00*, Vol. 2, pp. 556–564, Tel Aviv, Israel, March 2000. (*p. 9*)

[29] W. J. Dally, Velio Communications, "A single chip terabit switch," *Hot Chips XIII*, Stanford, August 2001. (*p. 70*)

[30] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE/ACM Transactions on Computers*, Vol. 39, No. 6, pp. 775–785, June 1990. (*p. 2*)

[31] W. J. Dally, P. Carvey and L. Dennison, "Architecture of the Avici terabit switch/router," *Proc. of Hot Interconnects VI*, pp. 41-50, August 1998. (*p. 2*)

[32] G. Dittmann and A. Herkersdorf, "Network processor load balancing for high-speed links," *Proc. of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2002)*, pp. 727–735, San Diego, California, July 2002. (*p. 34*)

[33] J. T. Dixon and K. L. Calvert, "Increasing demultiplexing efficiency in TCP/IP network servers," *International Conference on Computer Communications and Networks*, October 1996. (*p. 34*)

[34] A. M. Duguid, "Structural properties of switching networks," *Brown University Progress Report BTL-7*, 1959. (*p. 65*)

[35] M. Fomenkov, K. Keys, D. Moore and K. Claffy, "A longitudinal study of internet traffic from 1998-2001: a view from 20 high performance sites," *Proc. of WISICT '04*, Cancun, Mexico, January 2004. (*p. 32*)

[36] L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, 1962. (*p. 111*)

[37] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely and C. Diot, "Packet-level traffic measurements from the Sprint IP backbone," *IEEE Network*, 2003. *(to appear)* (*p. 32*)

[38] M. D. Grammatikakis, D. F. Hsu, M. Kraetzl and J. Sibeyn, "Packet routing in fixed-connection networks: a survey," *Journal of Parallel and Distributed Processing*, Vol. 54, no. 2, pp. 77–132, 1998. (*p. 2*)

[39] P. Gupta, S. Lin and N. McKeown, "Routing lookups in hardware at memory access speeds," *Proc. of IEEE INFOCOM '98*, pp. 1240–1247, San Francisco, CA, April 1998. (*p. 72*)

[40] P. R. Halmos, *Lectures on ergodic theory*, Chelsea Publishing Company, New York, 1956. (*p. 5*)

[41] J. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*, Kluwer Academic Publishers, Boston, 1990. (*p. 65*)

[42] ITU-T (International Telecommunication Union Standardization Sector), Recommendation G.841, "Types and characteristics of SDH network protection architectures," July 1995. (*p. 67*)

[43] S. Iyer, A. Awadallah and N. McKeown, "Analysis of a packet switch with memories running slower than the line-rate," *Proc. of IEEE Infocom '00*, pp. 529–537, Tel Aviv, Israel, March 2000. (*p. 1*)

[44] S. Iyer, R. R. Kompella and N. McKeown, "Designing buffers for router line cards," *Stanford University HPNG Technical Report - TR02-HPNG-031001*, Stanford, CA, March 2002. (*pp. 73 and 102*)

[45] S. Iyer and N. McKeown, "Analysis of the parallel packet switch architecture," *IEEE/ACM Transactions on Networking*, Vol. 11, No. 2, pp. 314–324, April 2003. (*pp. 1 and 33*)

[46] S. Iyer and N. McKeown, "Making parallel packet switches practical," *Proc. of IEEE Infocom '01*, Vol. 3, pp. 1680–1687, Anchorage, Alaska, USA, March 2001. (*p. 1*)

[47] S. Iyer, R. Zhang and N. McKeown, "Routers with a single stage of buffering," *ACM SIGCOMM '02*, Pittsburgh, USA, Aug. 2002. Also in Computer Communication Review, Vol. 32, No. 4, pp. 251–264, October 2002. (*p. 1*)

[48] R. Jain, "A comparison of hashing schemes for address lookup in computer networks," *IEEE Transactions on Communications*, Vol. 40, No. 3, pp. 1570–1573, October 1992. (*p. 34*)

[49] S. Jaiswal, G. Iannaccone, C. Dior, J. Kurose and D. Towsley, "Measurement and classification of out-of-sequence packets in a tier-1 IP backbone," *IEEE Infocom '03*, San Francisco, CA, 2003. (*pp. 10 and 32*)

[50] K. Kar, T. V. Lakshman, D. Stiliadis and L. Tassiulas, "Reduced complexity input buffered switches," *Hot Interconnects VIII*, Palo Alto, August 2000. (*p. 39*)

[51] L. Kencl and J. Y. Le Boudec, "Adaptive load sharing for network processors," *Proc. of IEEE Infocom 2002*, pp. 545–554, New York, NY, June 2002. (*p. 34*)

[52] I. Keslassy, S.-T. Chuang, K. Yu, D. Miller, M. Horowitz, O. Solgaard and N. McKeown , "Scaling Internet routers using optics," *ACM SIGCOMM '03*, Karlsruhe, Germany, August 2003. Also in *Computer Communication Review*, Vol. 33, No. 4, pp. 189–200, October 2003. (*p. 74*)

[53] I. Keslassy, M. Kodialam, T. V. Lakshman and D. Stiliadis, "New scheduling schemes for delayed graphs with applications to WDM rings," *in preparation.* (*p. 76*)

[54] I. Keslassy and N. McKeown, "Maintaining packet order in two-stage switches," *IEEE Infocom '02*, New York, NY, June 2002. (*p. 33*)

[55] D. König, "Graphok és alkalmazásuk a determinánsok és a halmazok elméletére [Hungarian]," *Mathematikai és Természettudományi Értesitö*, Vol. 34, pp. 104–119, 1916. (*pp. 65 and 107*)

[56] A. Krishnamoorthy, "Optoelectronics flip-chip-bonded to CMOS VLSI circuits," *IMAPS Advanced Technology Workshop on Next Generation IC and Package Design*, Solvang, CA, July 1999. (*p. 71*)

[57] A. L. Lentine *et al.*, "Arrays of optoelectronic switching nodes comprised of flip-chip-bonded MQW modulators and detectors on silicon CMOS circuitry," *IEEE Photonics Technology Letters*, Vol. 8, pp. 221–223, February 1996. (*p. 71*)

[58] E. Leonardi, M. Mellia, F. Neri and M. A. Marsan, "On the stability of input-queued switches with speed-up," *IEEE/ACM Transactions on Networking,* Vol. 9, No. 1, pp. 104–118, February 2001. (*p. 9*)

[59] R. M. Loynes, "The stability of queues with non-independent inter-arrival and service times," *Proc. of the Cambridge Philosophical Society*, Vol. 58, pp. 497–520, 1962. (*pp. 7 and 8*)

[60] N. McKeown, "iSLIP: a scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, Vol 7, No.2, pp. 188–201, April 1999. (*p. 9*)

[61] N. McKeown, A. Mekkittikul, V. Anantharam and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, Vol. 47, No. 8, pp. 1260–1272, August 1999. (*p. 9*)

[62] I. Norros, "A storage model with self-similar input," *Queueing Systems*, Vol. 16, pp. 387–396, 1994. (*p. 6*)

[63] A. M. Odlyzko, "Comments on the Larry Roberts and Caspian Networks study of Internet traffic growth," *The Cook Report on the Internet*, pp. 12–15, Dec. 2001. (*p. 74*)

[64] A. M. Odlyzko, "Internet traffic growth: sources and implications," *Proc. of SPIE Optical Transmission Systems and Equipment for WDM Networking II*, Vol. 5247, pp. 1–15, Orlando, FL, September 2003. (*p. 74*)

[65] E. Oki, J. Zhigang, R. Rojas-Cessa and H. J. Chao, "Concurrent round-robin-based dispatching schemes for Clos-network switches," *IEEE/ACM Transactions on Networking*, Vol. 10, No. 6, pp. 830–844, December 2002. (*p. 2*)

[66] G.I. Papadimitriou, C. Papazoglou and A.S. Pomportsis, "Optical switching: switch fabrics, techniques, and architectures," *Journal of Lightwave Technology*, Vol. 21, No. 2, pp. 384–405, Febrary 2003. (*pp. 9, 13, 47 and 52*)

[67] B. Pesach, G. Bartal, E. Refaeli, A. J. Agranat, J. Krupnik and D. Sadot, "Free-space optical cross-connect switch by use of electroholography," *Applied Optics*, Vol. 39, No. 5, pp. 746–758, February 2000. (*p. 47*)

[68] K. Petersen, *Ergodic Theory*, Cambridge University Press, Cambridge, 1983. (*p. 8*)

[69] G. Pfister, "An introduction to the InfiniBand architecture," *High Performance Mass Storage and Parallel I/O,* IEEE Press, 2001. (*p. 2*)

[70] K. W. Ross, "Hash routing for collections of shared web caches," *IEEE Network*, Vol. 11, No. 6, pp. 37-44, November-December 1997. (*p. 34*)

[71] R. Russo, B. Metzler, P. Droz and L. Kencl, "Scalable and adaptive load balancing on IBM PowerNP," *Technical report No. RZ-3431*, IBM Research Report, July 2002. (*p. 34*)

[72] R. Ryf, J. Kim, J. Hickey, A. Gnauck, D. Carr, F. Pardo, C. Bolle, R. Frahm, N. Basavanhally, C. Yoh, D. Ramsey, R. Boie, R. George, J. Kraus, C. Lichtenwalner, R. Papazian, J. Gates, H. Shea, A. Gasparyan, V. Muratov and J. Griffith, "1296-port MEMS transparent optical crossconnect with 2.07 petabit/s switch capacity," *Optical Fiber Comm. Conf. and Exhibit (OFC) '01*, Vol. 4, Postdeadline paper PD28, 2001. (*pp. 9, 13, 47 and 52*)

[73] A. Schrijver, "A course in combinatorial optimization," February 2003, available at http://www.cwi.nl/~lex/files/dict.ps. (*pp. 65 and 107*)

[74] A. Schrijver, "Bipartite edge-coloring in O(Δm) time," *SIAM J. Comput.*, Vol. 28, pp. 841–846, 1999. (*p. 65*)

[75] S. Scott and G. Thorson, "The Cray T3E network: adaptive routing in a high performance 3D torus," *Proc. of Hot Interconnects IV*, August 1996. (*p. 2*)

[76] A. Singh, W. J. Dally, A. K. Gupta and B. Towles, "GOAL: a load-balanced adaptive routing algorithm for torus networks," *International Symposium on Computer Architecture (ISCA)*, San Diego, CA, USA, June 2003. (*p. 2*)

[77] D. Slepian, "Two theorems on a particular crossbar switching network," *unpublished memorandum*, 1952. (*p. 65*)

[78] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," *ACM SIGCOMM '02*, Pittsburgh, USA, August 2002. (*p. 74*)

[79] N. Spring, R. Mahajan, D. Wetherall and T. Anderson, "Measuring ISP topologies with Rocketfuel," *IEEE/ACM Transactions on Networking*, Vol. 12, No. 1, February 2004. (*p. 74*)

[80] Y. Tamir and H. C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 1, pp. 13–27, 1993. (*p. 9*)

[81] Telcordia, GR-499 CORE, "Transport systems generic requirements (TSGR): common requirements criteria," Issue 2, December 1998. (*p. 67*)

[82] Telcordia, GR-253 CORE, "Synchronous optical network (SONET) transport systems: common generic criteria," Issue 3, September 2000. (*p. 67*)

[83] L. G. Valiant, "A scheme for fast parallel communication," *SIAM Journal on Computing*, Vol. 11, No. 2, pp. 350–361, 1982. (*p. 2*)

[84] L. Valiant and G. Brebner, "Universal schemes for parallel communication," *Proc. of the 13th Annual Symposium on Theory of Computing*, pp. 263–277, May 1981. (*p. 2*)

[85] J. von Neumann, “A certain zero-sum two-person game equivalent to the optimal assignment problem,” *Contributions to the Theory of Games*, Vol. 2, pp. 5–12, Princeton University Press, Princeton, NJ, 1953. (*p. 26*)

[86] W. Willinger, M. S. Taqqu and A. Erramilli, “A bibliographical guide to self-similar traffic and performance modeling for high speed networks,” *Stochastic Networks, Theory and Applications*, Oxford University Press, pp. 339–366, 1996. (*p. 6*)

[87] R. Zhang-Shen, I. Keslassy and N. McKeown, “Designing a predictable Internet backbone,” *in preparation.* (*p. 76*)