# Scalable Multi-module Packet Switches with Quality of Service

**Santosh Krishnan**

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

**COLUMBIA UNIVERSITY**

2006

ABSTRACT

# Scalable Multi-module Packet Switches with Quality of Service

Santosh Krishnan

The rapid growth in packet-based network traffic has resulted in a growing demand for network switches that can scale in capacity with increasing interface transmission rates and higher port counts. Furthermore, the continuing migration of legacy circuit-switched services to a shared IP/MPLS packet-based network requires such network switches to provide an adequate Quality of Service (QoS) in terms of traffic prioritization, as well as bandwidth and delay guarantees. While technology advances, such as the usage of faster silicon and optical switching components, provide one dimension to address this demand, architectural improvements provide the other. This dissertation addresses the latter topic. Specifically, we address the subject of constructing and analyzing high-capacity QoS-capable packet switches using multiple lower-capacity modules.

Switches with the output-queueing (OQ) discipline, in theory, provide the best performance in terms of throughput as well as QoS, but do not scale in capacity with increasing rates and port counts. Input-queued (IQ) switches, on the other hand, scale better but require complex arbitration procedures, sometimes impractical, to achieve the same level of performance. We leverage the state-of-the-art in OQ and IQ switching systems and establish a new taxonomy for a class of three-stage packet switches, which we call *Buffered Clos Switches*. The taxonomy is created by augmenting existing switching elements with *aggregation, pipelining and parallelization* techniques. This offers a switch designer several alternatives, each driven by specific design and re-use constraints, to build a high-capacity switch composed of lower-capacity basic elements.

We also present a formal framework for optimal packet-switching performance, in order to uniformly characterize the capabilities of the switches in the class. The optimality is based

on establishing *functional equivalence* of a given switch and its associated arbitration algorithms with a well-understood ideal switch. For the items in the above taxonomy, we demonstrate how some existing algorithms perform with respect to the optimality criteria, and then augment the state-of-the-art by presenting algorithms and analytical results for stricter equivalence with an ideal switch.

# Contents

# List of Figures

# List of Tables

# Preface

The design and analysis of scalable packet switches is now a mature field, thanks to several pioneering contributions by the switching research community in the late 1990's. This was fueled partly by a quest for the biggest and most powerful switching system during the telecommunications boom visited upon us in that period. The time is now ripe to further the maturation process by lending the field a semblance of formality and rigor common in classical circuit switching. Motivated by this goal, I present here a compendium of techniques and analyses that may be used by switch designers to methodically construct and study high-capacity switches in a formal and unambiguous framework.

My choice to pursue this subject matter was not entirely coincidental. I was very fortunate to be part of the architecture team of three generations of the best-selling Atlanta$^{\text{TM}}$ switching chip-set at Bell Labs. Most of the original work presented here was indeed inspired by the research conducted and lessons learned in the 1998–2002 time-frame while working on that and related projects. The delayed publication of the material, on my part, may largely be attributed to some entrepreneurial adventures in the interim.

## Acknowledgments

Several people deserve my sincere gratitude for supporting this non-linear effort. First of all, I would like to thank my advisor Prof. Henning Schulzrinne for his invaluable help and encouragement, and my co-advisor, Dr. Fabio Chiussi, who mentored me rather admirably during my years at Bell Labs. Thanks are also due to Prof. Ed Coffman, Prof. Vishal Misra and Prof. Dan Rubenstein for being part of my dissertation committee. This work was partially supported by the Lucent doctoral scholarship, for which I am thankful to the erstwhile Lucent management.

Many of the ideas here were influenced by numerous engaging discussions with my ex-colleagues and fellow researchers at Bell Labs, while working with the Atlanta/$\pi$ and Packetstar teams, in the good old days. Specifically, I would like to thank Alberto Brizio, Abhijit Choudhury, Andrea Francini, Lampros Kalampoukas, Denis Khotimsky, Jonathan Rosenberg, Sheng Shen, Dimitrios Stiliadis and Bernhard Suter for always sharing their expertise, whenever I needed it. I am grateful to the Invento Networks team, who kept life interesting and came along for a ride the last two years. Special thanks are due to Dave Bartolini, Bob MacDonald and Bob Power for spending time with me and helping me learn newer tricks.

Finally, I am thankful to my dear wife Vaishali for her continued support, motivation and, in general, an upbeat attitude towards everything, not to mention her invaluable help in the grammatical editing of this dissertation.

Santosh Krishnan

May 2006

To Vaishali

# Chapter 1

# Introduction

The telecommunications industry has witnessed a dramatic increase in packet-based network deployments over the past decade. Such deployments have spanned fiber-based wide and metropolitan area networks, as well as various flavors of wireline and wireless access networks. The enterprise and data-center industries have seen a similar proliferation in the form of packet-based local area, storage and cluster computing networks. Increasing traffic carried over such networks has propelled the demand for high-capacity packet-switching systems.

The networking protocols supported by these systems vary widely from connection-oriented to connectionless ones depending upon deployment. Legacy Asynchronous Transfer Mode (ATM) and the emerging Multi-Protocol Label Switching (MPLS) [94] typically provide transport services within network domains, the former still prevalent in DSL and cellular access networks and the latter finding favor in service-provider metropolitan and core networks. The ubiquitous Internet Protocol (IP) and Ethernet continue to support end-user connectivity, often overlayed on the above transport networks. Elsewhere, purpose-built technologies such as Infiniband and Myrinet provide interconnectivity for storage networks and cluster computing.

Independent of the specific protocol supported, and the diversity of names given to the respective systems, e.g., multi-service switch, router and packet multiplexor, the core of such systems consists of a *forwarding path* that transfers traffic units from the input to the output interfaces. In the interest of protocol agnosticity, we refer to all such traffic units as *packets* in the generic sense, and the systems as *packet switches*. The capacity of a packet switch depends upon the number of supported interfaces, also known as port count, and the transmission rates of those interfaces. Two commonly used metrics for switching capacity include *forwarding* and *processing* capacity. The former is measured as the amount of cumulative traffic that may be instantaneously dispatched by the switch, e.g., a switch with 16 full-duplex interfaces each operating at 10 Gb/s has a nominal capacity of 160 Gb/s. Processing capacity is measured as the number of packets that may be processed in unit time, e.g., if the above switch can tolerate a minimum packet size of 125 bytes, the processing capacity on each interface would be 10 million packets per second (pps).

The need for higher switching capacity stems from both higher port counts due to dense deployments, and higher transmission rates enabled by improvements in the physical layer. A typical high-end IP router in production today (e.g., Cisco 12000 series, Juniper T-series) supports 10-20 ports, each operating at 10-40 Gb/s, thus requiring a forwarding capacity of several hundred Gb/s. Since IP packet sizes have not changed materially with the increasing rates, assuming a minimum packet size of 64 bytes, the router would require a processing capacity of about 20-80 million pps per interface. These represent more than a tenfold increase in capacities since a decade ago. This trend is expected to continue, albeit at a slower pace compared to the past decade, driven by a greater number of high-speed fiber/DSL ports at the edges of the network, the anticipated adoption of 10-G Ethernet in the metropolitan area, and increasing OC-192 (10 Gb/s) and OC-768 (40 Gb/s) links in the core. A higher forwarding capacity imposes severe constraints on the construction of the switch, e.g., on the design of packet memories, interconnection network and arbitration logic, while a higher processing capacity constrains the complexity of the algorithms such as route determination and scheduling that must operate on a per-packet basis.

As an orthogonal trend, multimedia and other traditionally circuit-switched network services are being migrated to a converged packet-based network. Such services are characterized by traffic *flows* that require Quality of Service (QoS) in the form of bandwidth and delay guarantees. Examples of such flows include Voice-over-IP (VoIP) sessions, Video-on-Demand streams, and MPLS Virtual Private Network (VPN) tunnels providing multi-site office connectivity. While end-to-end QoS in an IP network remains a pipe-dream and is typically addressed by throwing additional capacity into the network, QoS capability is being slowly engineered into selected portions of the network. For example, the TR-58 specification of the DSL forum and the PacketCable specification of CableLabs allow for QoS provisioning of end-user flows in the access network. Elsewhere, QoS for aggregates of end-user flows are becoming common in MPLS-based transport networks. Consequently, the forwarding path of a modern packet switch needs to provide preferential treatment to specified flows so that their negotiated QoS requirements may be met.

While the increase in packet-based network traffic has been facilitated by enhancements in physical layer technologies and QoS capability has been addressed by advances in link scheduling, the forwarding path itself remains a bottleneck primarily because memory bandwidths have not experienced comparable improvements. Consequently, there is a continuing interest in building packet switches using multiple modules of lower capacities. This dissertation addresses the design and analysis of such multi-module switches. We present architectures that may be used to construct the forwarding path of a large-scale packet switch, and the associated algorithms to optimize throughput and provide QoS to the served traffic flows. Towards this goal, we propose a new taxonomy for a class of scalable switches, together with an analytical framework to characterize optimal performance for the algorithms employed within such switches.

## 1.1 Motivation

The design and analysis of classical multi-module circuit switches (see [45] for an overview) provide a historical precedence and inspiration for this work. Scalable switches, using an interconnection network of smaller components, were proposed to decrease the number of expensive electronic crosspoints within a large switch. Notable examples of such architectures include the Banyan, Benes, Clos and Cantor networks. This work benefited immensely from a well-established performance framework, namely, the property of *blocking*, which characterized the performance of a given architecture in isolation from the properties of specific circuit arrival processes. Levels of blocking behavior, namely, strict-sense, wide-sense and rearrangeable, allowed to accurately deduce the functional equivalence of a given design with an ideal single-stage switch, in terms of circuit admission capability. Such rigor and precision are lacking in prior packet-switching work.

Admittedly, significant progress has already been made towards the design of multi-module packet switches in the form of combined input-output queueing (CIOQ) switches (e.g., see [24, 70]), memory-space-memory switches [19, 22], and more recently, parallel switches [50, 62] and load balanced Birkhoff-Von Neumann (BVN) [9] architectures. Such works have yielded several arbitration and load-balancing algorithms for the respective designs, yet, in the absence of a uniform performance framework, several diverse, often inadequate, measures and/or simulation results have been used to claim optimality, making it difficult to make fair comparisons between switches. This has likely restricted the practical implementations of sophisticated high-performance designs in the industry. For example, a recent survey [102] as well as a look at systems based on the standard Advanced Telecom Computing Architecture (ATCA) platform confirm that a majority of packet switches continue to adopt simple variations of the shared-memory output-queued (OQ) or a crossbar-based input-queued (IQ) design, often with unknown performance characterizations.

Among patently imperfect measures is a practice still prevalent in the industry to rig together an interconnection between the input and output interfaces and claim the nominal capacity, i.e., the sum of the interface rates, as the *throughput* of the system. For example, a number of so-called terabit routers are but multiple switching blades interconnected by a separate cluster controller switch. The mere fact that a path exists between any two interfaces of a system does not directly imply that the nominal rates can be sustained on each interface. Consequently, the advertised capacity of the system is rendered meaningless in engineering the network. Another approach, common in earlier ATM literature, is to quantify switching performance by deriving packet loss ratios in the finite buffers within the switch for specific arrival patterns. It can be argued that such ratios are primarily indicative of the arrival processes themselves, and not entirely meaningful in choosing a switch design. Fortunately, these approaches are falling out of favor in the modern literature, replaced by concepts such as non-blocking and 100% throughput. While the latter constitute a step in the right direction, they are not entirely satisfactory.

The concept of non-blocking, which is in reality a circuit-switching property, has been variously and often ambiguously applied to characterize a packet switch. One of its more reasonable interpretations is a switch that is non-blocking in structure, i.e., a design in which a path can always be found between an idle input-output interface pair. While this might be a necessary

| **Measure** | **Limitation** |
|---|---|
| Full interconnection | Interface throughput not necessarily sustainable |
| Loss ratios for finite switch buffers | Property of arrivals, not of the switch |
| Non-blocking structure | Does not address throughput |
| 100% throughput | Does not address arbitrary, e.g., inadmissible, arrivals |

Table 1.1: Common measures for packet switching performance

condition to optimize throughput, it is not sufficient as evidenced by a simple crossbar switch, which is trivially non-blocking yet not necessarily optimal in throughput. A better measure is that of 100% throughput (example applications can be seen in [29, 70]), which refers to the ability of a switch to ensure stable queues for admissible traffic in the long term. Well-behaved traffic is taken for granted either via auxiliary admission control means or as a result of end-to-end flow control. While this approach comes closest in the literature to providing a meaningful performance framework, it does not address equivalence with an ideal switch for arbitrary, including inadmissible, traffic, leading sometimes to necessitate additional considerations such as starvation prevention. Table 1.1 summarizes these common measures and their limitations.

The above problems are compounded by a lack of clear demarcation between the logical architectures of packet switches, and their chosen implementations. For example, a classical $N \times N$ OQ switch is logically similar in operation to several space-division multiplexed designs [110] such as the bus-matrix switch and the full-mesh switch each of which employs $N^2$ disjoint paths, with respective queues, between the input-output pairs. A similar case can be made for a buffered crossbar switch, in which there has been some recent interest, which places those $N^2$ queues at the crosspoints of a $N \times N$ crossbar, with the main implementation concern being minimizing the on-chip queue sizes via backpressure to larger off-chip queues at the inputs. The logical similarity is due to the fact that in each of these designs, arriving packets can immediately be presented to the destination queues that are organized on a per-output basis, and each of those queues are random-accessible in the departure decisions. Our contention is that any analysis of such designs constitutes an analysis of the implementation and its associated trade-offs and not that of the logical switch architecture.

These considerations lead us to two important questions: what qualifies as the logical architecture of a multi-module packet switch, and what models can be used to characterize optimal performance of a packet switch design? Once these questions are answered, the remaining task would be to identify the algorithms that may be implemented to achieve such optimal performance. Prior taxonomical work [88] have covered these questions for ATM switch topologies, though most of the focus was on the structural blocking property of the interconnection network. Further studies (e.g., [12]) have dealt with the design and analysis of a specific class of Banyan-based topologies. Though such works, and others previously mentioned in this section, have enabled to implement practical multi-module switches, providing functional equivalence with an ideal switch remains an open topic of research for many of the simple non-blocking networks augmented with buffers at arbitrary points within them. This dissertation takes a small step in addressing that topic.

## 1.2 Contributions

The primary contribution of this work is a constructive approach to building a high-capacity packet switch, using stages of lower capacity memory and space elements. We restrict ourselves to designs that resemble the three-stage Clos network, augmented with buffers. For switches with this structure, which we call *Buffered Clos switches*, we establish a taxonomy of packet switch designs. Each item in the taxonomy is accompanied by the design and re-use constraints that it resolves, so that switch implementors may identify the most appropriate item to use. To accurately characterize the capabilities of the switches in the taxonomy, we introduce a new performance framework for packet switches based on the ability of their resident algorithms to achieve *functional equivalence* with an ideal switch. While some items in the classification are already addressed to a large extent in the literature, the taxonomy itself and the performance framework are novel. In addition to providing a road-map to construct scalable switches, this work allows to ascertain the properties of existing vendor equipment by inspecting the structure and the associated algorithms for functional equivalence.

To place this work in the context of existing results in the literature, we present it in a new perspective, namely, as an exercise in how formal methods from graph theory, combinatorics and stochastics may be beneficially used to design and analyze switching algorithms. This exposition is concentrated on IQ and CIOQ switches, which form the starting point of our taxonomy and the inspiration for our performance framework. For the heavily researched CIOQ switch, we augment the state of the art by presenting algorithms that may be used to achieve stricter equivalence with an ideal switch. We propose a new intuitive proof that shows how a CIOQ switch may be considered as an analog of the circuit Clos switch. This result is used to exhibit how simple matching algorithms may be used to achieve bandwidth and delay guarantees. We then present three new fundamental results on the throughput capability of such switches. Specifically, we prove that a class of critical matching algorithms ensures equivalence with an ideal switch without requiring internal speedup, and that well-chosen maximal matchings are sufficient for asymptotic emulation of an ideal switch even under inadmissible traffic.

Proceeding further in the above taxonomy, we present the design and analysis of CIOQ switches with *aggregation* and *pipelining*, two crucial ingredients to scaling. These fall into the category of what we call single-path buffered Clos switches. We show the application of these two transformations to convert a CIOQ switch, without losing performance, into general memory-space-memory switches. We present methods that shadow a CIOQ switch, combined with matrix decomposition to account for aggregation, and with sequential, balanced or concurrent dispatch methods to account for pipelining, in order to inherit the performance of the reference CIOQ switch. Some existing switches appear similar to these, nevertheless, the presented algorithms and analytical results are novel. Lastly, we propose and analyze a new *parallel* packet switch architecture, which belongs to what we call multi-path buffered Clos switches. We cover the topics of stable load balancing and sequence control for such switches. It should be noted that while all the contributions mentioned in this section are original, some of the accompanying results have already been published elsewhere by the author.

### 1.2.1 Applicability and Scope

This work is restricted in several aspects to keep it tractable. We deal mainly with unicast traffic. While the ability to handle multicast traffic is certainly relevant to packet switching, it is not addressed with the same level of detail except, in some cases, to point out easily adaptable extensions. We also do not include specialized switching algorithms that may be used to optimize the performance for adaptive traffic such as TCP. Our assumption is that the presented performance framework by itself will be beneficial to adaptive traffic as well, and other mechanisms, such as active queue management, to enforce fairness among adaptive flows, can be added on to this work without much difficulty. In addition, we do not address the end-to-end network behavior and network engineering. Instead, we concentrate on individual node mechanisms whose suitability for end-to-end optimal behavior may be studied independently. This work is also agnostic of the protocol processing specifics and control-plane handling, e.g., signaling the flows for QoS, and route control.

This dissertation covers only the switching node architectures and algorithms. There are several simplifying assumptions that will restrict the direct application of this work to practical implementations. For example, the effect of finite-sized buffers is not explored in detail. For the most part, the buffers in the logical switching elements are considered to be infinite, with the assumption that the tails of the queue-length distributions may be used to yield packet loss ratios in actual designs. Similarly, the full effect of variable-sized packets, and the overhead of the associated segmentation and reassembly component, is not adequately addressed. We also admit that several of the presented algorithms are centralized in nature. The complexity of the logic as well as a requirement to build multi-board systems might necessitate non-trivial extensions to achieve a comparable distributed implementation. Consequently, this work should be treated as a theoretical advancement, which may be used as a guide to devise suitable heuristics to enable practical designs.

The author himself was responsible for several key components of the architecture of the commercial Lucent/Agere Atlanta(tm) chipset [19] and the protocol-independent ($\pi$) chipsets [21, 22], all of which were based on memory-space-memory and buffered crossbar designs before those terms became fashionable in the literature. The author was also responsible for a prototype architecture of a parallel packet switch named $\pi$-group. While this dissertation does not cover any of those implementations in detail, those products provided proofs of concept for the ideas contained here.

## 1.3 Organization

The remainder of this dissertation is organized as follows. Chapter 2 contains our view of the packet switching model. We introduce the basic logical building blocks that are meaningful for packet switching, namely, memory and space elements, and make a case for meaningful notions of optimal performance. The latter includes three basic propositions that encompass the QoS and throughput properties of a switch. These are presented before the chapter on literature in order to place the latter in proper perspective.

Chapter 3 covers the related work in switch design, including a short exposition of circuit-based Clos networks, and a main focus on the results in IQ and CIOQ packet switching. These are presented in a new context, namely, as a usage of formal methods from diverse mathematical areas to arrive at high-performance switching algorithms. We cover the question of how matching algorithms are devised for input-queued switches, and how those matchings have been applied to QoS and throughput performance in the prior art.

Chapter 4 presents our taxonomy of buffered Clos switches, and the new performance framework of functional equivalence. This is accompanied by a presentation of how existing results fit into the new framework. The taxonomy allows switch designers to assemble basic building blocks into sophisticated multi-module switch designs, while the framework of equivalence allows to accurately characterize and compare the performance of the algorithms implemented within such switches.

Chapter 5 augments the state-of-the-art of CIOQ switches, the first item in our taxonomy. We provide a proof of how a circuit Clos switch is analogous to a packet CIOQ switch, using which we exhibit how bandwidth and delay guarantees may be provided. On the throughput of such switches, we show that a class of critical matching algorithms ensure equivalence with an ideal switch for admissible traffic. Furthermore, we show that an existing class of simple algorithms, namely maximal matching, may be used for stricter equivalence with an ideal switch under partially admissible traffic, and present a specific maximal matching that ensures asymptotic emulation of the ideal for abritrary traffic, which solidifies this equivalence. Finally, the throughput and QoS results are combined to yield a unified matching framework.

Chapter 6 covers single-path buffered Clos switches, namely, switches with aggregation and pipelining. We present and analyze matching algorithms for such switches, based on shadowing a high-performance CIOQ switch. The transitivity property of functional equivalence is applied to translate the optimal throughput and QoS results of CIOQ switches to such single-path designs. For switches with aggregation, we propose a shadow-and-decompose method, as well as a lower-complexity matching method, using a more efficient queueing strategy, in order to obtain performance comparable to CIOQ. Towards the same goal, for switches with pipelining, we propose and analyze sequential, concurrent and equal dispatch methods. To conclude, the possibility of recursively constructing switches with smaller components, using a larger number of stages, is briefly presented.

Chapter 7 contains the architecture and analysis of parallel packet switches, which are multi-path manifestations of buffered Clos switches. We propose load balancing algorithms for such switches and analyze their switching performance. We also cover issues unique to multi-path designs, primarily that of sequence control. For completeness, we include a short exposition on the other common multi-path design in the literature, namely, the BVN switch.

Chapter 8 presents the conclusions of this work, including problems that remained unaddressed and avenues for future research.

# Chapter 2

# Switching Model

We first overview the switching framework relevant to this work, including definitions of basic concepts such as contention, blocking and throughput, followed by a case for some meaningful notions of optimal performance. We then present models for the forwarding path of a packet switch, and their atomic logical building blocks. We briefly introduce the output-queued (OQ) and input-queued (IQ) switching models, the former representing the ideal reference switch for our performance framework.

## 2.1 Switching Basics: Overview

### 2.1.1 Circuits: Blocking

Circuit-based networks divide the physical communication media into units called *channels*. Depending upon the multiplexing technique, channels may correspond to timeslots, frequency bands, codes or wavelengths. A *circuit* is an end-to-end traffic pipe established by pre-determining the path to be traversed through the network, and assigning channels, if they are available, on the links that comprise the path. Traffic is segmented into *frames*, and each circuit switching node in the path transfers the frames that arrive on a given channel of an input link to the respective pre-established channel of an output link. Examples of such nodes include SONET switches based on electronic crossbars and WDM waveguide routers based on optical components such as tunable lasers and MEMS mirrors.

A *contention* is a condition in which more than one frame compete for a given link at the same instant. If this link belongs to the set of outputs of a switch, we refer to the condition as *external* contention. If it belongs inside the interconnection network of the switch, we refer to the condition as *internal* contention. Since channels are pre-established on every network link traversed by a circuit, there is no external contention and no inherent necessity to provide buffers for pending frames, other than for synchronization purposes. Consequently, the performance measures of interest in circuit switching include only the admissibility of circuits, or call-level behavior, and the capability to realize internal paths within the switch to accommodate admitted circuits. The former is studied using *loss models* [95], which characterize circuit acceptance probabilities based on the statistics of call arrivals and holding times, a celebrated example of which is the Erlang loss system used to engineer telephony networks. The latter is determined by the architecture of the switching node itself.

A circuit switch achieves maximum throughput as long as an internal path can be established for every circuit that is admitted solely on the basis of the available resources of the external links. Such a property, effectively, allows us to remove the specifics of the internal switch architecture from network engineering considerations, which then depend only on the available link resources in the network. A switch that satisfies this property is referred to as *non-blocking*. It is said to be non-blocking in the strict sense if an internal path can be established without disturbing any existing ones. If a re-arrangement of the existing paths is necessary and sufficient to support a newly admitted circuit, the switch is called re-arrangeably non-blocking. All non-blocking switches may be considered *functionally equivalent* in terms of the circuits they can support.

A trivial example of a strictly non-blocking $N \times N$ circuit switch, with each external link supporting a single circuit, is a crossbar that employs $N^2$ electronic crosspoints to fully interconnect the $N$ inputs to the $N$ outputs. Much of the work in circuit switching [45, 88] addresses the construction of a multi-module bufferless fabric using an interconnection network of smaller components. This was motivated primarily by the fact that electronic crosspoints were expensive and their number in a crossbar increases quadratically with the size of the switch. Such networks may have internal contention due to commonality in the internal paths between input-output pairs, and hence can potentially be blocking. An internal *speedup*, defined as the ratio of the total link capacity between two internal stages to the total external capacity, is usually used to counter blocking. Examples of notable interconnection networks are the Banyan network and the Batcher sorting network, which require $O(N \log N)$ and $O(N \log^2 N)$ crosspoints, respectively. Both these networks are self-routing with a unique path between every input-output pair, however, the resulting structure is blocking. A popular example of a non-blocking network, which we shall revisit on several occasions, is the three-stage Clos network, which uses $O(N^{1.5})$ crosspoints. A more complex example is the Cantor network, which uses $O(N \log^2 N)$ crosspoints and remains non-blocking by employing several planes of a blocking sorting network. Indeed, the literature in the design of bufferless interconnection networks is fairly rich, yet rigorous due to the well-defined cost parameter and performance framework, namely, the number of crosspoints and blocking behavior, respectively.

### 2.1.2 Packets: QoS and Throughput

Packet-based networks do not rely on holding dedicated link resources and on switching between pre-established physical layer channels. Instead, the physical and link layers have the flexibility to employ either synchronous means such as SONET, or asynchronous ones such as Ethernet, on a link-by-link basis. Forwarding decisions at each switching node are made separately for each individual packet, based on the packet header information and the state maintained within the switch.

A traffic *flow*, in the most generic sense, refers to a stream of related packets. A flow may be fine-grained, corresponding to an end-to-end application session, or coarse-grained such as a permanent MPLS path providing connectivity between two IP subnets. In fact, each switch may view the aggregate traffic between each of its input-output pairs as a flow of the coarsest granularity. Flows are identified by a switch either using a label in the header, as in ATM and MPLS, or through a header-filter based on ranges of source and destination addresses, protocol types and/or transport-layer port numbers. Resources may be negotiated and configured for a subset of flows traversing a switch through a signaling phase prior to packet transmission or via long-term service level agreements (SLA). The traffic presented to a typical packet switch consists of a combination of such *guaranteed QoS* flows, for which the traffic profiles (e.g., a token bucket specification [28]) and service requirements, including the desired average rate and tolerable maximum delay, are known in advance, and *best-effort* flows without any pre-specified profiles or requirements. Even the former are allowed to violate their negotiated profiles, in which case the excess component of their offered traffic is treated on a best-effort basis.

The foremost distinguishing feature of a packet switch is the inherent presence of external contention, i.e., several packets destined to the same output may compete for that link simultaneously. This phenomenon can be sustained over an arbitrary period of time, resulting in a backlog of unserved packets, which need to be buffered. An arriving packet may be dropped in response to congestion in the finite amount of available buffers, resulting in a packet loss ratio for the corresponding flow. The admitted backlog is then *scheduled* in a chosen fashion, which determines the packet delays, the observed flow service rates and the total throughput of the switch. Consequently, the performance measures of interest in packet switching include not only the admissibility of flows but also the abilities to meet the negotiated QoS requirements and to maximize throughput, both in the presence of external contention. Multi-rate loss models [95], similar to the ones in circuit switching, may be used to address the former for guaranteed QoS flows, while empirical ones (e.g., [5, 89]) are used to characterize the expected traffic for best-effort flows. These models, based only on the external link resources, may be meaningfully used to engineer the network, provided a packet switch possesses the above abilities, thereby removing specifics of the internal switch architecture from such engineering considerations.

In summary, the primary goals in designing a packet switching node include provisions to ensure that the observed packet throughput on the external links approaches the advertised link capacities, and provisions to provide preferential treatment, in the form of scheduling and buffer management, to specified flows so that their negotiated QoS requirements are met. When design constraints necessitate building a multi-module switch, the challenge is to meet these goals in the presence of both external and internal contention.

## 2.2 Notions of Optimal Performance

We now crystallize some of the concepts seen in the overview, and arrive at a few propositions that serve as guiding principles for switching analysis. A circuit request between an input-output link pair of a circuit switch is considered *admissible* if there are available channels on the specified external links. Let $V$ denote a set of simultaneously admissible circuits. A non-blocking switch ensures that a path within the switch is realizable for every $v \in V$. Since this holds for any given $V$, including *any* given definition of an optimal set, and since circuit switches drop inadmissible requests, we assert the following:

**Proposition 1.** *The non-blocking property ensures optimal throughput of a circuit switch.*

Packet switching calls for similar properties that ensure optimal QoS and throughput, limited solely by external link resources, for the most general models of offered traffic. Let the traffic arriving at input $i$ and destined to output $j$ of a packet switch be referred to as the input-output flow $(i, j)$. Let the amount of arriving traffic, e.g., in bits, for each flow be maintained by the arrival matrix $A$, where $A_{i,j}[t_1, t_2)$ refers to the arrivals in the interval $[t_1, t_2)$ for flow $(i, j)$. The short-form $A(n)$ may be used for $A[0, n)$. In general, this is governed by a stochastic arrival process. The long-term average rates of arrival yield the *offered rate* matrix $\lambda$, i.e.,

$$\lambda_{i,j} = \lim_{n\to\infty} \frac{A_{i,j}(n)}{n}. \tag{2.1}$$

The arrivals are considered admissible if the following conditions are satisfied:

$$\forall i \sum_j \lambda_{i,j} \le C \quad \text{and} \quad \forall j \sum_i \lambda_{i,j} \le C, \tag{2.2}$$

where $C$ refers to the link capacity of the inputs and outputs. The above is typically used to restrict the modeled arrivals due to the following reasons. The first inequality is a physical constraint imposed by the input link. A violation of the second would imply that at least some of the output links cannot sustain the offered rates. As a special case, the arrivals are considered *bounded admissible* with timescale $T$, if the average rates $\lambda$ hold for every interval of length $T$, and $\lambda$ is admissible, i.e.,

$$\forall i, j, t \;\; \frac{A_{i,j}[t, t+T)}{T} \le \lambda_{i,j} \tag{2.3}$$

and (2.2) is satisfied. In other words, (2.3) imposes a timescale on the averaging of arrival rates. The arrivals are considered *partially admissible* if the second inequality in (2.2) holds for a proper subset of the outputs. Unlike circuit switching, neither admissibility nor a knowledge of $\lambda$ may be presumed due to the presence of best-effort traffic.

The *required rate* matrix $R$ contains the effective-rate requirements of the guaranteed QoS component. In general, this can be independent of $\lambda$. If rates are specified for finer-grain flows, these may be aggregated per input-output pair to yield $R$. As opposed to $\lambda$, the required rate matrix may be presumed to be known and admissible, i.e.,

$$\forall i \sum_j R_{i,j} \leq C \quad \text{and} \quad \forall j \sum_i R_{i,j} \leq C. \tag{2.4}$$

The above is ensured by the admission control procedure that negotiates QoS on the basis of the external link resources. Consequently, a switch that is able to allocate any given combination of admissible rates throughout the internal paths between respective input-output pairs may be considered optimal in QoS capability. We refer to the act of allocating rates as *flow fitting* or providing virtual bandwidth trunks through the switch. Notice that these trunks may be viewed as multi-rate circuits, e.g., a request for 5 Mb/s on a link of 20 Mb/s capacity may be viewed as a simultaneous request for 5 circuits on a link with a capacity to admit 20 circuits. Therefore, it is straight-forward to see that a multi-stage packet switch that has the same logical structure, i.e., an identical interconnection network with identical capacities on each internal link but augmented with contention buffers in certain stages, as a non-blocking circuit switch is guaranteed to find the requisite internal paths to fit any set of admissible flows.

**Proposition 2.** *A packet switch with a non-blocking structure is optimal in QoS capability.*

Due to the variability in the packet arrivals, mere path realization is not sufficient. Additional mechanisms, specifically, scheduling policies in each stage of the switch, are required to ensure QoS satisfaction. Nevertheless, in terms of switch architecture, a non-blocking design is a good first step. Since much of the early packet switching work was based on ATM switches, wherein the entire traffic undergoes admission control, the ability to fit flows was considered the primary objective, also leading to several claims of optimal throughput based solely on the non-blocking property.

Notice that if all $\lambda_{i,j}$ are known beforehand, and $\lambda$ is admissible, we may use it instead of $R$ in order to create virtual bandwidth trunks within the switch. In this special case, the ability to provide optimal QoS is sufficient to realize maximum long-term throughput as well. However, the knowledge of offered rates is an invalid assumption in packet switching, necessitating additional measures for optimality in throughput. Since the amount of traffic that may be successfully dispatched on an output link is limited by the capacity of that link, maximum throughput is ensured as long as an output link of the switch never idles when a packet destined to it is backlogged anywhere within the switch. Switches with this property are referred to as system-wide *work conserving*. Therefore, a packet switch may be considered *ideal* if it is work conserving under any offered traffic, and is able to allocate any given combination of required rates at the same time.

If $\lambda$ is admissible, every queue in an ideal switch remains stable. If $\lambda$ is partially admissible, only those queues that contain traffic destined to oversubscribed outputs, i.e., outputs $j$ for which $\sum_i \lambda_{i,j} \geq C$, become unstable. Within an oversubscribed output $j$, an ideal switch has the ability to serve each individual flow $(i, j)$ with a specified proportion of the output capacity. This results in a subset of the queues that contain traffic to output $j$ to remain stable, specifically, the queues corresponding to $(i, j)$ for which $\lambda_{i,j}$ is less than the respective proportion. Notice that any switch that can maintain the stability of the same set of queues as an ideal switch ends up providing the same asymptotic throughput to each corresponding flow, and hence the same throughput for the switch as a whole. We refer to such a switch as *functionally equivalent* to an ideal switch. In other words, we may use the stability of a given set of queues for equivalence purposes in packet switching, much like admissibility of a given set of circuits in circuit switching. If it is difficult or impossible to prove that a switch design under consideration is ideal, one may then claim optimal throughput by establishing functional equivalence with an ideal switch, for the most general model of offered traffic.

**Proposition 3.** *A packet switch that is functionally equivalent, in terms of queue stability, with a well-known ideal switch is optimal in throughput.*

We can now relate the techniques used in the existing literature to the above notion of optimal throughput. One approach is to rely on directly establishing a switch to be work conserving (e.g., [65]), while another is to ensure that the packets depart from the outputs of a given switch in an *exact emulation* of the departure from a well-known ideal switch, for identical arrival patterns (e.g., [24]). One widely used approach that relies on establishing queue stability, and hence implicitly on functional equivalence with an ideal switch, is the notion of 100% throughput

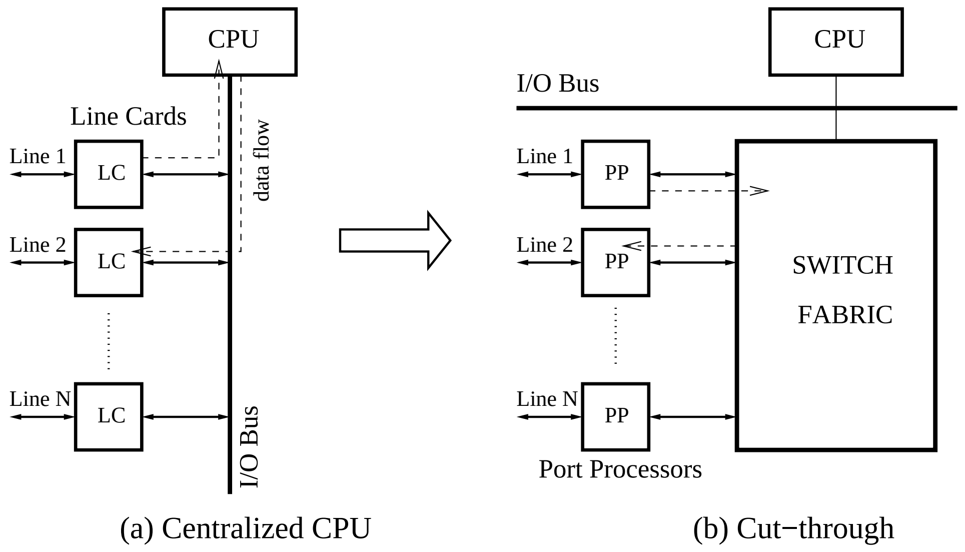


Figure 2.1: Forwarding models: (a) Centralized CPU, (b) Cut-through

(e.g., [29, 70]). The technique is to prove the stability of all the queues for admissible offered rates. This is a reasonable approach except, maybe, for the presumption of admissibility.

To summarize, we may use the following steps to design a multi-module packet switch that remains optimal in QoS and throughput performance. We start with a non-blocking structure, possibly borrowed from circuit switching, augment it with internal buffers wherever necessary to account for contention, and provide for scheduling policies that can guarantee virtual bandwidth trunks to satisfy any admissible required-rate matrix. We then ensure that the devised scheduling policies, which govern the dispatch decisions from module to module and finally to the external links, are able to maintain stability in the same set of queues as an ideal switch for the same offered traffic, not necessarily admissible. The performance capability becomes weaker whenever restrictions are placed on the offered traffic.

## 2.3 Forwarding Models

With a framework in place for optimal packet switching performance, independent of the internal switch architecture, as desired, we are now ready to address the latter, namely, the topic of switch design itself. We identify two basic forwarding models for a packet switch, the centralized CPU-based model and the cut-through fabric model.

The first generation of routers, and many of the current lower capacity ones, perform software-based forwarding using a centralized processor. A general purpose CPU is connected to multiple line cards through an I/O bus, as shown in Fig. 2.1(a). The lines shown are full-duplex, i.e., they contain both input and output interface components. In practice, the line cards may be located off of the same board as the CPU, with an on-board bus such as PCI providing connectivity to the CPU, or on separate boards on a chassis, with a star or bus connectivity between them. The cards perform the physical and link layer protocol processing and forward the incoming packets to the CPU. Access to the I/O bus may be controlled by the CPU, e.g., by polling the line cards, or arbitrated via a separate medium-access technique. The headers of the incoming packets are analyzed by the CPU to determine the outgoing interface and any special handling, if applicable, following which they are dispatched to the respect output in a chosen order.

In general, a packet may experience contention at several points within the system, namely, at the I/O bus waiting to gain access and get transferred to the CPU, at the CPU waiting to be processed, and finally, as is the case with any packet switch, at the output interface. Accordingly, queueing buffers are provided in the line cards next to the input and the output interfaces, and in the CPU memory. An evident disadvantage of this model is that the limited communication bandwidth of the I/O bus and the processing power of the CPU present significant bottlenecks to packet throughput, and thereby limit the port counts and the interface rates that can be supported. Providing QoS is practically ruled out if packets need to be queued *before* they can even be identified by the CPU.

Consider an example of a $16 \times 16$ switch with 1 Gb/s ports. In order to make such a switch *wire-speed*, i.e., to ensure that any queueing happens only after a packet has been identified, and to remove the I/O bus and the CPU as internal contention points, the components need to be dimensioned as follows. Assuming a minimum packet size of 64 bytes, the CPU needs a processing capacity of 32 million pps (16 Gb/s of total offered traffic divided by 512 bits), which translates to an upper bound of about 32 cycles on a 1 GHz CPU to account for memory access and

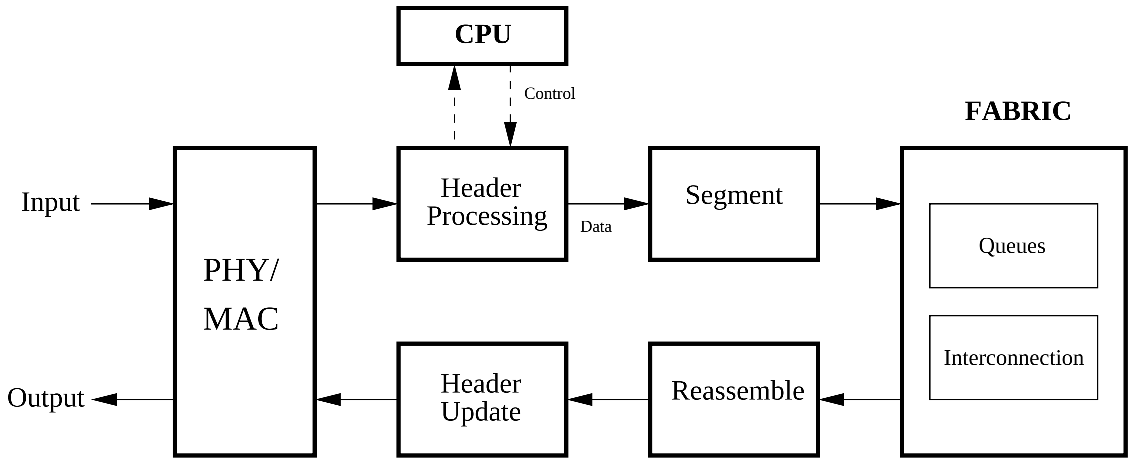


Figure 2.2: Pipeline of a cut-through forwarding path

instructions. To sustain the full-duplex traffic, the I/O bandwidth required is 32 Gb/s, compared, e.g., to the 8 Gb/s available using a 64-bit 133 MHz PCI bus. The bandwidth required of the CPU memory is also 32 Gb/s, allowing merely 16 ns for a memory transfer assuming a wide 64-byte memory bus. Clearly, these are tall orders even for the modestly sized switch in the example. Consequently, the path traversed by the packet through a central CPU is often referred to as the *slow path*.

Modern designs use the *cut-through* model illustrated in Fig. 2.1(b). Special-purpose port processors, attached to each full-duplex line, operate on packets at the ingress and egress of the switch. In addition to physical and link-layer processing, packet header processing is also performed within these units using local state information. Such processing includes network address lookup for output determination, flow identification via filters, flow policing if applicable, statistics collection, and other protocol-specific tasks. The results of these operations that are relevant to packet dispatch are encoded and inserted into a special local header. The packets are then routed directly to the respective output interfaces through a dedicated switch fabric, also known as the *fast path*. If the fabric expects fixed-size data units, the port processor is also responsible for the associated segmentation and reassembly (SAR) of variable-sized packets. A central CPU is connected to the port processors either via a special port of the switch fabric, as shown in the figure, or through a dedicated I/O bus. In this model, only the network and system control traffic is routed to the CPU, which implements the relevant routing and management protocols, maintains the overall state of the system and programs the local states into the port processors and switch fabric. In other words, the slow path continues to handle the control plane, while the dedicated fast path handles the data plane. Fig. 2.2 shows an example pipeline of a cut-through forwarding path.

In practice, there are several variations of this logical model. Each port processor unit may be implemented on a separate board, attached to a chassis, with a star topology on the backplane interconnecting them to a switch fabric board. Alternatively, the interconnection may be on the backplane itself as a full-mesh between port processor cards, which may also house the queues of the switch. In any case, all internal contention in this model is concentrated within the fabric, provided the port processors operate at wire speed. For the same $16 \times 16$ switch example as above, this requires a processing capability of 2 million pps in each unit. Communication IC vendors already offer so-called network processors that implement the necessary components in silicon at interface rates of 10 Gb/s and processing capacities of 20-30 million pps. More importantly, as opposed to the previous model, these capacities do not need to increase linearly with the port count. The switch fabric needs to sustain a forwarding capacity of 16 Gb/s, with a memory bandwidth that depends on the architecture, though it clearly does not exceed 32 Gb/s. Vendors today use several different types of designs to offer fabrics that operate at a nominal capacity of several hundred Gb/s.

While there are quite a few interesting problems in scaling the port processors, such as the best sorting structures to use for address lookup, for example, we concentrate on the design of the fabric itself.

## 2.4 Building Blocks

We introduce two types of basic forwarding elements to construct a fabric, namely *memory elements* and *space elements*. These are logical entities that address the two primary functionalities of a packet switch, namely, queueing to absorb contention, and providing interconnection. From the perspective of multi-module switch design, we consider these entities to be the atomic build-

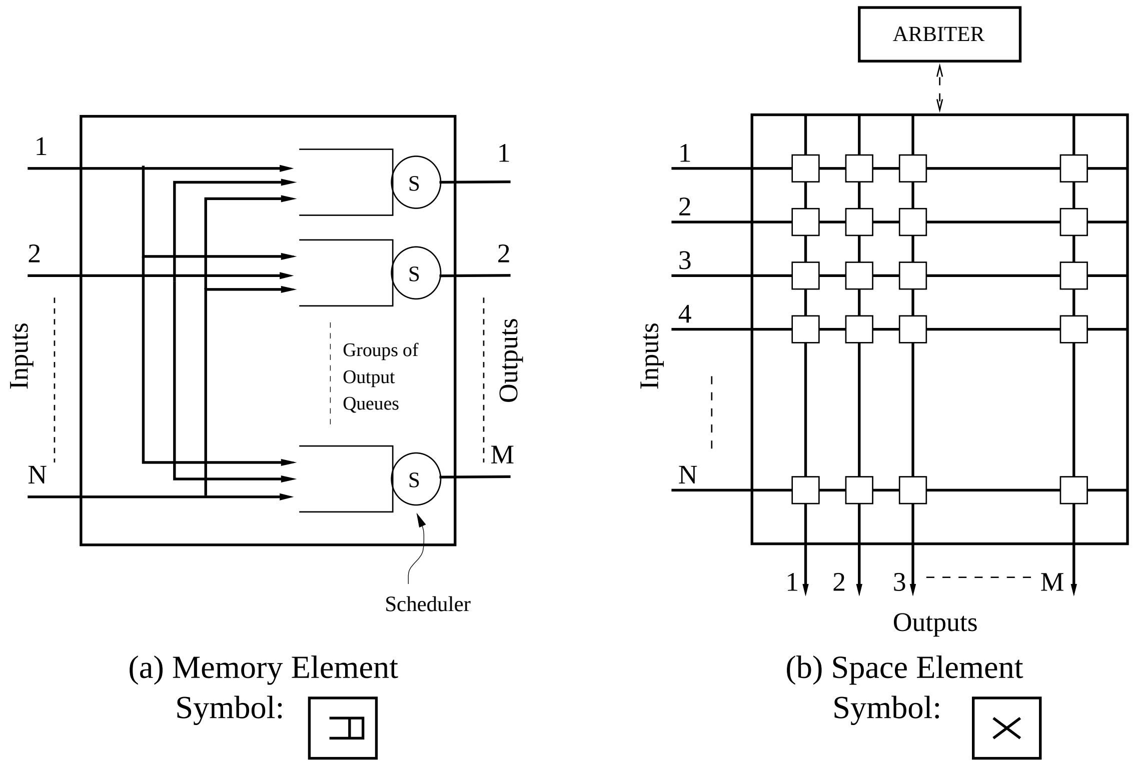


Figure 2.3: Forwarding elements: (a) memory element, (b) space element

ing blocks.

An $N \times M$ memory element, shown in Fig. 2.3(a), accepts packets from the $N$ input interfaces and enqueues them into queues that are arranged into groups, one group for each of the $M$ outputs. A multicast packet is enqueued into several queues simultaneously. Note that the output groups need not be in physically separate memories, and may indeed share resources from a common pool. At the other extreme, each group may be implemented using a bank of several physical memories. The structure of an output group may vary depending on the kind of preferential treatment to be given to the queued packets. For example, in the simplest case, we may have one queue per output for a total of $M$ queues in the element. We may have one queue per input in each group, allowing to keep traffic belonging to each input-output pair separate, for a total of $NM$ queues in the system. Alternatively, we may have multiple queues organized on the basis of a combination of criteria such as flow identity, eventual destination, and/or packet priority. A scheduler logic at each output dictates the outgoing packet rate and the relative order of the packets. The policy employed may vary from simple first-in first-out (FIFO) acting on a single output queue to a sophisticated weighted fair queueing (WFQ) [30] mechanism acting on per-flow queues. In practice, a buffer management logic is responsible for selectively admitting packets into the finite memory resource. Since our primary focus is on contention and queue stability, unless otherwise mentioned, we assume the memory size to be infinity.

The two main characteristics of an atomic memory element are that incoming packets are immediately enqueued into their respective output queues, making them available for immediate dispatch, and that the output scheduler has the flexibility to organize the queues in any fashion to facilitate arbitrary scheduling policies. The only points of contention in the element are at the output links. Given an interface rate of $C$ and a minimum packet size of $L$, the element needs a memory bandwidth of $(N+1)C$ at each of the outputs in order to accommodate $N$ simultaneous enqueue operations (from the $N$ inputs) and a single dequeue operation. At the same time, the scheduler logic needs to dequeue at a frequency of $C/L$ to sustain the output interface rate. Therefore, the following are the main constraints that limit the capacity of a single element. The available memory bandwidth limits the supported port counts and interface rates. The required frequency of scheduling, determined by the smallest packet size, limits the complexity of the scheduler. Finally, the implementation is also constrained by the amount of circuitry required to realize the full mesh of paths, which grow quadratically with the switch size.

An $N \times M$ space element, shown in Fig. 2.3(b), is composed of a memoryless cross-connect and an arbiter[1] (possibly distributed) that is responsible for configuring the element to interconnect specific inputs to outputs. Due to the lack of queues, an interconnection needs to be free of conflicts, i.e., at any instant $t$, an input may be connected to no more than one output, and vice-versa. Such a configuration, usually represented by an $N \times M$ matrix of binary entries $\pi(t)$, is known as a *matching*. The rows of the matrix correspond to the input links and the columns to output links. The conflict-free property may be represented as:

$$\forall t \ \ \{\forall i \sum_j \pi_{i,j}(t) \leq 1 \ \ \text{and} \ \ \forall j \sum_i \pi_{i,j}(t) \leq 1\} \tag{2.5}$$

[1]Some parts of the literature refer to an arbiter as a *switch scheduler*. In order to avoid confusion with the link scheduler, found in the memory element, we will continue to call it an arbiter.

| **Element** | **Properties** | **Constraints** |
|---|---|---|
| Memory | Immediate output enqueue<br>Flexible queue organization | Memory bandwidth: $O(N)$ per output<br>Scheduler frequency: $O(1/L)$<br>Full-mesh connectivity: $O(N^2)$ |
| Space | No queues<br>Conflict-free interconnection<br>Non-blocking | Arbiter frequency: $O(1/L)$<br>Cost vs. pin-count |

Table 2.1: Forwarding elements: Properties and constraints

Packets at the inputs are presented as equal-sized units called *cells*[2]. The time taken to transfer a single cell from an input to its respective output is referred to as a *timeslot*. The entire element is synchronized on timeslots, and the arbiter computes a sequence of matchings $\pi(n)$, so that if $\pi_{i,j}(n) = 1$ in timeslot $n$, a cell is allowed to be transfered from input $i$ to output $j$ in that timeslot. The properties of the sequence determine the throughput of the element.

In addition to being memoryless and limited to conflict-free configurations, an atomic space element is assumed to be non-blocking, i.e., the internal implementation is ensured to realize a path for any conflict-free configuration. Note that the points of internal contention in the element are shifted to the input links due to the property (2.5) of matchings. Since a matching is re-computed in every timeslot, the arbitration logic needs to operate at a frequency of $C/L$, which will limit the complexity of the policy. As the latter often depends on the size of the element, this ends up restricting the port counts as well. For example, a centralized arbiter that implements a policy based on the full knowledge of cells available for dispatch, for all the input-output pairs, will have a complexity of $\Omega(NM)$. In practice, the pin count and the internal crosspoints that can be cost-effectively realized also limit the size and interface rates supported by the element. For example, if an electronic crossbar is used to implement a space element, its cost increases at least quadratically with the port count. Table 2.1 summarizes the properties and design constraints of each element.

Since we treat these elements as logical entities, their implementations subsume different variations of logic that may be used to exhibit the above properties. For example, a single log-

[2]Note that this term merely refers to the internal unit of transfer and is not to be confused with ATM cells. Some systems also refer to such units as *pages*.

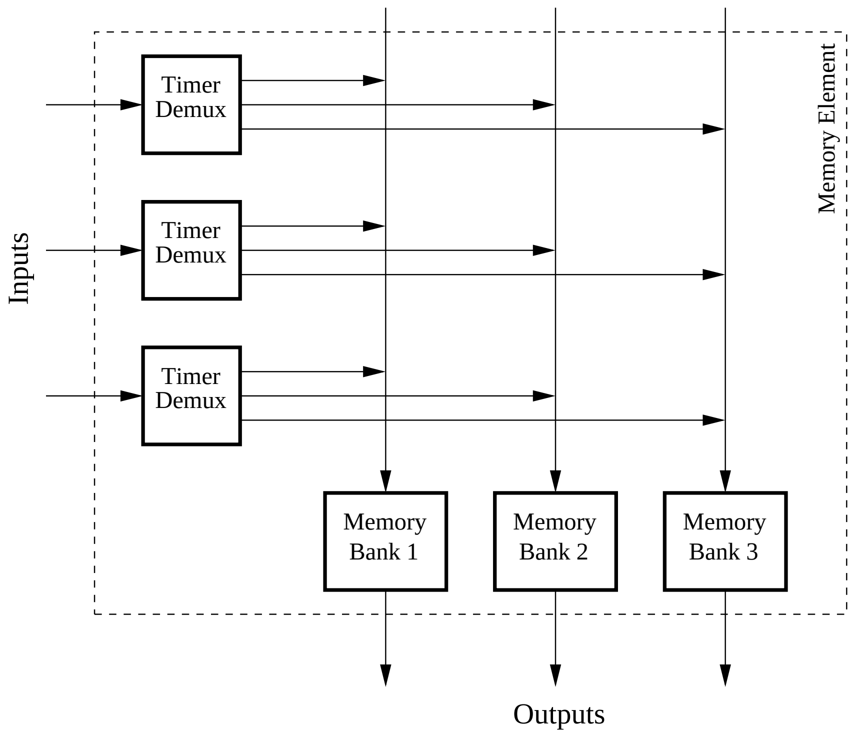


Figure 2.4: Memory element implementation example: Knockout configuration

ical memory element may be implemented in a distributed fashion, with an on-board full-mesh circuitry interconnecting the inputs to several memory banks, one for each output. This circuitry may indeed use $NM$ distinct traces as shown in Fig. 2.3(a), a shared timeslotted bus running at a capacity of $NMC$ as in the Prelude switch [31], or $M$ parallel buses running at $NC$ as shown in Fig. 2.4. An example of the latter is the Knockout switch [116] in which each bus may run at a lesser speed than the required maximum.

To alleviate the memory bandwidth required in each bank, we may use $N$ physical memories, each operating at a capacity of $2C$ (for a simultaneous enqueue and dequeue), one corresponding to each input. A single output scheduler controls the entire bank for each output. A switch that uses such a logic, i.e., $NM$ disjoint paths to $NM$ distinct memories is referred to as a bus-matrix switch [110]. A buffered crossbar is a variation that uses $(N + M)$ traces and $NM$ distinct memories by placing the latter at crosspoints of an electronic crossbar. However, since these are on-chip memories, their realizable sizes tend to be quite small, opening up a different set of problems. Irrespective of the implementation and the bandwidth of each physical memory, the requirement for each bank continues to grow linearly with the switch size.

A space element may be readily implemented by a single electronic crossbar. Such components today are capable of interface rates in the order of 1-10 Gb/s, with port counts in the order of 20-32. These can be re-configured in less than 100 ns thereby providing a very small timeslot. Optical components such as waveguide routers (e.g., [115]) can support a significantly higher interface rate, however the port counts supported by currently available ones are lower, and more importantly, the re-configuration times may be as high as a few microseconds, thus providing a very coarse timeslot and limiting the smallest tolerable cell sizes to be very high. Again, since we view these elements as logical entities, all the techniques from the circuit switching literature may be used to construct a large non-blocking interconnection network that emulates a single space element. For example, a single logical element may be implemented by a multi-module bufferless Clos network.

## 2.5 Common Existing Switches

A packet switch that employs the cut-through forwarding model, with a single memory element for the switch fabric is referred to as an *output queued* (OQ) switch. An OQ switch, in terms of logical elements, is shown in Fig. 2.5(a). The only contention points in an OQ switch are at the output links. Since packets are immediately presented to the output link schedulers, an OQ switch is capable of being work conserving. Furthermore, since the departure time for each packet is controlled by a single scheduler, the switch is able to allocate any combination of required rates to individual flows, of any granularity. Consequently, given the required interface counts and rates, a hypothetical OQ switch of those dimensions may be considered as an ideal switch for performance comparisons. In practice, while earlier generation switches [31, 110, 116] commonly used this design, pure OQ switches are rarely used in current systems, beyond a capacity of a few Gb/s, primarily due to the required memory bandwidth. It is due to this that there is even an interest in other, more scalable, designs.

An *input queued* (IQ) switch, shown in terms of logical elements in Fig. 2.5(b), concen-

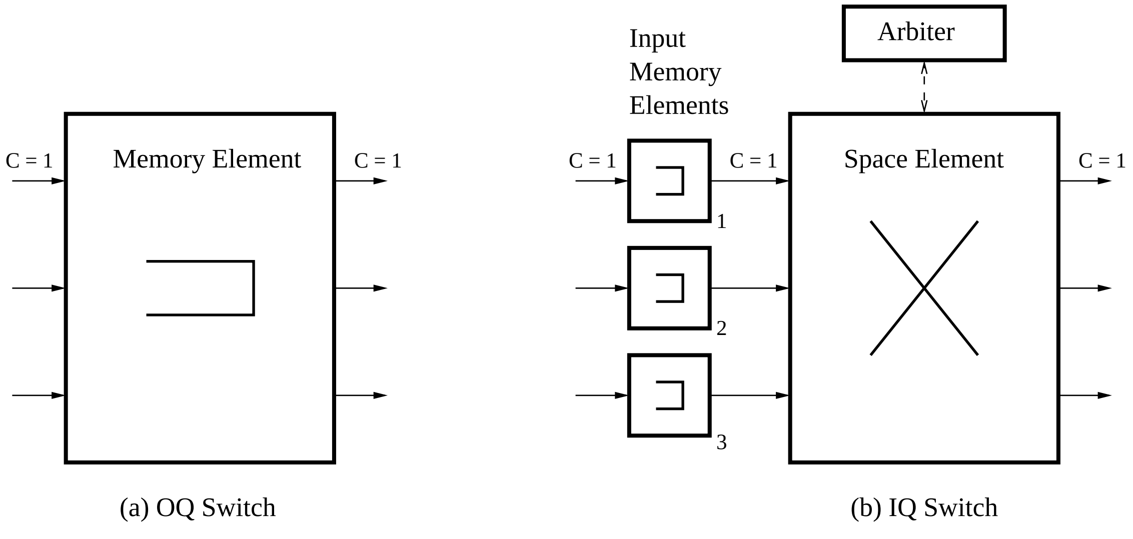


Figure 2.5: Common existing switching models: (a) Output-queued, (b) Input-queued

trates all the memory at the input interfaces of the system. An $N \times N$ fabric consists of a single space element, e.g., a crossbar, with $N$ instances of $1 \times 1$ memory elements connecting each of the system inputs to the input ports of the space element. The $N$ elements are necessary to absorb the input contention that is inherent to the space element. We assume that packets are segmented into cells either in the memory elements or in the port processors prior to entering the switch fabric. Notice that the memory elements are not allowed to function in a work conserving fashion, due to the contention, and it is the sequence of matchings in the space element that drives the former, and thereby determines the throughput and QoS capability of the system. In other words, when $\pi_{i,j} = 1$ in the space element, memory element $i$ is instructed to schedule and dequeue a packet destined to output $j$. Indeed, most of the literature in IQ switching covers matching policies aimed to achieve high performance in the presence of internal contention. Since the maximum memory bandwidth required in the system is $2C$ (in each memory element), and more importantly, since it is independent of the size of the switch, this design is extremely scalable and quite popular, limited in dimensions only by the realizable size of the space element and the complexity of its resident arbitration policies.

Early IQ switches used a single FIFO queue in each of the input memories. The adverse

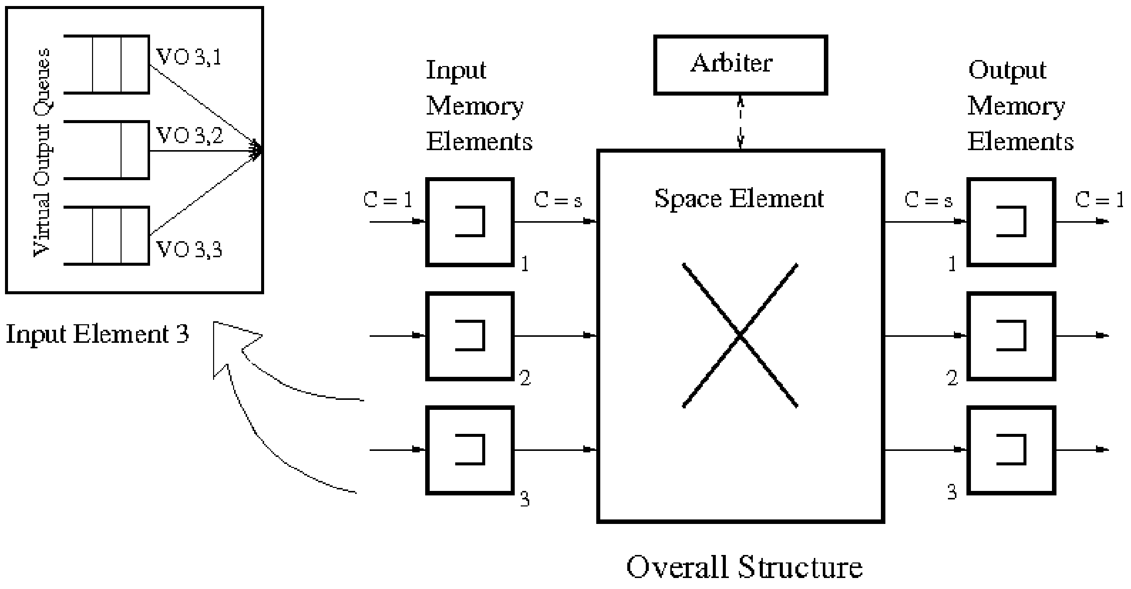


Figure 2.6: A combined input-output queueing switch (CIOQ)

effect of such an arrangement is that the only cell that is eligible for transfer to its destination is the one at the head of the queue. Consequently, a cell destined to an idle output cannot be dispatched if it is enqueued behind a cell destined to a currently busy output, even if there is no contention for the input link to the space element. This phenomenon is called *head-of-line* (HOL) blocking. Karol et al. [55] showed that even under relatively benign arrival traffic assumptions, specifically, independent Bernoulli sources on each input link uniformly distributed to all the outputs, the throughput of this system is limited to $2 - \sqrt{2}$ (59%) times the available bandwidth. This situation is easily remedied by queueing cells in an input memory into $N$ separate queues, one for each output, referred to as *virtual output queues* (VOQ) [109]. This allows the cells belonging to each input-output pair to be considered independently by the arbitration policy. Essentially, when $\pi_{i,j} = 1$, the output scheduler in element $i$ chooses a cell from the virtual output queue corresponding to input-output flow $(i, j)$. Henceforth, we will assume such virtual output queueing in all designs based on space elements. Note that such an organization merely implies a different data structure for ordering cells, and does not necessitate separate physical memories. The required memory bandwidth in each memory element continues to be $2C$ in the presence of virtual output queueing.

To alleviate input contention and facilitate less complex arbitration policies, a *speedup* may be employed between the input memories and the space element. The speedup $s > 1$ is the ratio of the bandwidth of the internal links to that of the external interfaces. Essentially, this enables to transfer cells at a faster rate through the space element. In this case, queues are also required at the outputs of the system to account for the rate mismatch between the internal and external links. Such a switch, shown in Fig. 2.6 is referred to as a combined input-output queueing (CIOQ) switch. When $s = 1$, this reduces to a simple IQ switch, and when $s = N$, it starts behaving like an OQ switch. This design experiences both input and output contention, though the former is less than in the case of a pure IQ switch. The maximum memory bandwidth continues to be independent of the switch size, and is equal to $(1 + s)C$ in each memory element. A cell timeslot measured on the system inputs and outputs is referred to as the external timeslot, while the timeslot of the space element is referred to as the internal timeslot. Clearly, the latter is shorter in order to account for the faster cell transfer, and is equal to $(1/s)$ times the former. This implies a higher frequency of re-configuration.

Due to their relatively low memory bandwidth requirements, IQ and CIOQ switches present a good starting point to explore scalable multi-module designs. On the other hand, due to its optimal performance, OQ switches provide us with an ideal reference for comparison. Most of the modern literature in packet switching concentrates on arbitration policies for the former so that they may approach the performance of OQ switches.

# Chapter 3

# Formal Methods in Switching

In theory, there is no difference between theory and practice. In practice, there is.

– *Yogi Berra*, circa 1980

We now present the related work in switch design as it pertains to the forwarding models and performance measures introduced in the previous chapter. We include a short overview of the circuit-switched Clos network, and its properties, which will be applied later to our own packet-switched designs. Next, we focus on the throughput and QoS properties of the popular IQ and CIOQ switches in the literature, which will provide a launching pad for the remainder of this work. Specifically, we focus on how results from graph theory, combinatorics and stochastic stability have been applied to devise and analyze matching algorithms for those switches. Some of the related work in more sophisticated designs, such as the parallel packet switch, have been deferred to the discussion sections of later chapters, after such designs have been introduced and our own contributions exposed.

## 3.1 Clos Network

The Clos network is a three-stage interconnection composed of space elements, typically implemented by crossbars, in each stage. While it was first introduced and analyzed in the 1950s, it remains relevant to this date, and we will borrow this overview from Pattavina's more recent synopsis [88]. As shown in Fig. 3.1, an $N \times N$ switch is composed of three fully connected stages,

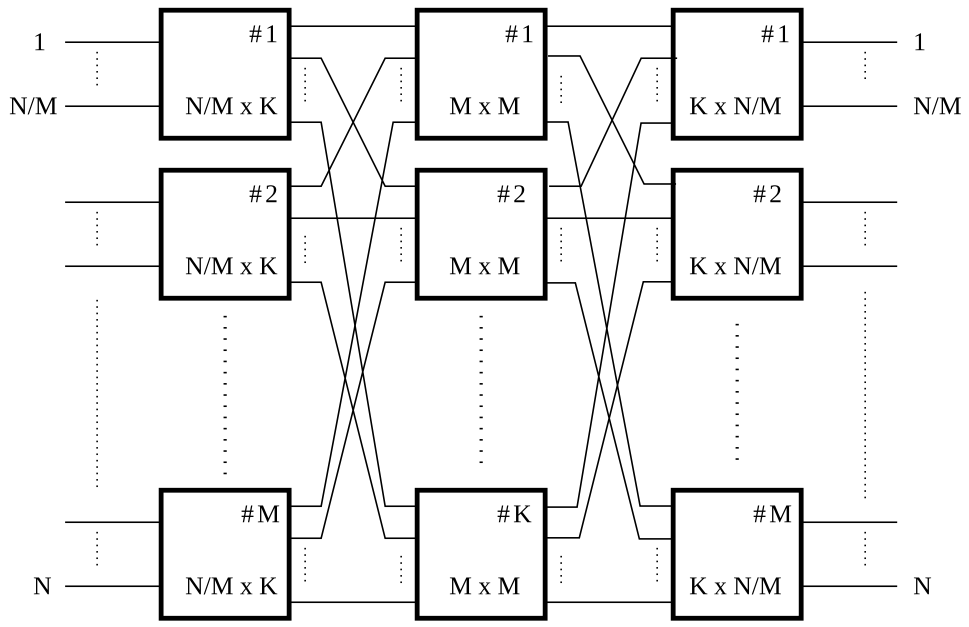


Figure 3.1: An $N \times N$ Clos network

where each internal and external link has a capacity of one circuit. The first stage is composed of $M$ instances of $N/M \times K$ non-blocking space elements. These are interconnected to the third stage composed of $M$ instances of $K \times N/M$ elements, using $K$ elements of size $M \times M$ in the central stage. The offered traffic load consists of the arrival processes of circuit requests. Recall that a circuit request between input-output pair $(i, j)$ is admissible if both external links, input $i$ and output $j$, are idle. The Clos theorem [25], which is paraphrased below, provides the sufficiency condition for the network to be strict-sense non-blocking, i.e., for the network to be able to assign an internal path to an admissible circuit without disturbing the existing configurations.

**Theorem** (Clos)**.** *A request for an admissible circuit is guaranteed to be satisfied, without reconfiguring the paths of existing circuits, as long as* $K \geq (2N/M - 1)$.

The above theorem is easily proven by contradiction. The minimum number of cross-points in such a network is $O(N^{1.5})$, which results when $M = \sqrt{2N}$. The required internal speedup of the network is immediately calculated as $(2 - M/N)$, or close to 2 for large $N$. Given a request between $(i, j)$, a greedy algorithm which visits the $K$ central elements, in any arbitrary order, will find an element $k$ whose input $\lceil\frac{i}{N/M}\rceil$ and output $\lceil\frac{j}{N/M}\rceil$ are both unconnected, essentially providing a path for the request through $k$. Note that $\lceil\frac{i}{N/M}\rceil$ is the first-stage element to which the input $i$ belongs, and $\lceil\frac{j}{N/M}\rceil$ is the third-stage element to which the output $j$ belongs. The central-stage element $k$, in effect, provides a free path between those two elements. We refer to such a greedy algorithm as *Clos fitting*. For each circuit, it has a run-time complexity of $O(K)$, as in the worst case, it may have to visit every space element.

We observe here that the network continues to be non-blocking if any aggregate of admissible circuit requests are simultaneously presented to an idle switch. The Clos fitting algorithm simply partitions the aggregate into $K$ configurations, one for each space element. This variation, shown below, can be applied to three-stage *packet* switches and is a subject of the analysis in Chapter 5.

**Corollary** (Clos). *An admissible aggregate request is guaranteed to be fully partitioned into $K$ conflict-free configurations by any Clos fitting algorithm as long as $K \geq (2N/M - 1)$.*

Another combinatorial aspect of the above properties is that similar results can be obtained when the quantities being fitted on the links are any additive scalars. Let the capacity of each link in Fig. 3.1 be normalized to 1 unit, and let the quantities to be fitted on an input-output pair vary within $[b_{\min}, b_{\max}]$. A quantity $b$ is admissible for the pair $(i, j)$ if both input $i$ and output $j$ have existing allocations that do not exceed $(1-b)$. A path can be realized for an admissible quantity as long as the following condition holds [81]:

$$K \geq 2\left\lfloor \frac{(N/M) - b_{\max}}{1 - b_{\max} + b_{\min}} \right\rfloor + 1 \tag{3.1}$$

This reduces to the Clos theorem when all quantities are unity, the extra terms in the above expression accounting for the fragmentation of link resources due to the variability in the quantities. Note that the scalars may represent the peak or the effective bandwidths of packet flows. Hence, we may construct a three-stage network that is structurally equivalent to the Clos network, and enable fitting of admissible flows, irrespective of whether the individual components are memory or space elements, as long as each element can internally (through arbitration or scheduling) realize the bandwidths of the individual flows. This outcome may be viewed as a manifestation of Proposition 2 for Clos topologies.

The network in Fig. 3.1 can realize an internal path for an admissible circuit without speedup if existing configurations are allowed to be re-arranged. The Slepian-Duguid theorem [45], paraphrased below to suit our terminology, provides the sufficiency condition for the network to be re-arrangeably non-blocking.

**Theorem** (Slepian-Duguid)**.** *A request for an admissible circuit is guaranteed to be satisfied as long as $K \geq N/M$ and re-configuration of existing circuits is allowed.*

The procedure for satisfying an admissible circuit, which we refer to as *Slepian-Duguid fitting*, works as follows. Let the circuit originate from input element $i$ and terminate at output element $j$. Due to the admissibility of the circuit, either there exists a central-stage element $k$ with both input $i$ and output $j$ unconnected, or there exist two elements $k_1$ and $k_2$, the former with unconnected input $i$ and the latter with unconnected output $j$. The procedure terminates immediately if the earlier condition is met, in which case the path is realized though $k$. Otherwise, the circuits in $k_1$ and $k_2$ are re-arranged to accommodate the new circuit. In the worst case, $M$ re-arrangements are necessary, with each step needing to inspect $O(M)$ entries. Combined with the step required to find $k_1$ and $k_2$, the algorithm has a run-time complexity of $O(M^2 + K)$.

We illustrate the intuition behind the re-arrangements through an example, shown in Fig. 3.2. Consider a $12 \times 12$ switch, with 3 $(4 \times 4)$ elements in the first and third stages. In accordance with the above theorem, there are 4 $(3 \times 3)$ central elements. Let the current configurations of the central elements be as shown in the top row. Note that if entry $(i, j)$ is set to 1 in element $k$, it implies that first-stage element $i$ is connected to third-stage element $j$, by central element $k$. Let a request arrive for a new circuit between an input of first-stage element 2 and an output of third-stage element 2. This circuit is admissible because the respective elements currently support only 3 circuits each. We first observe that row 2 of element 1, and column 2 of element 3 (both dotted), are empty and hence unconnected. Therefore, we choose 1 and 3 as $k_1$ and $k_2$ and re-arrange their circuits in the following manner. Arbitrarily, we start with element 1 and set $(2, 2)$ to 1 in order to satisfy the new circuit. This causes a conflict with the entry in $(1, 2)$, which is moved to element 3. A chain of alternate exchanges, shown by the numbered block arrows, are performed till we obtain conflict-free configurations in both elements. This procedure never visits the same row (column) twice and is guaranteed [45] to converge.

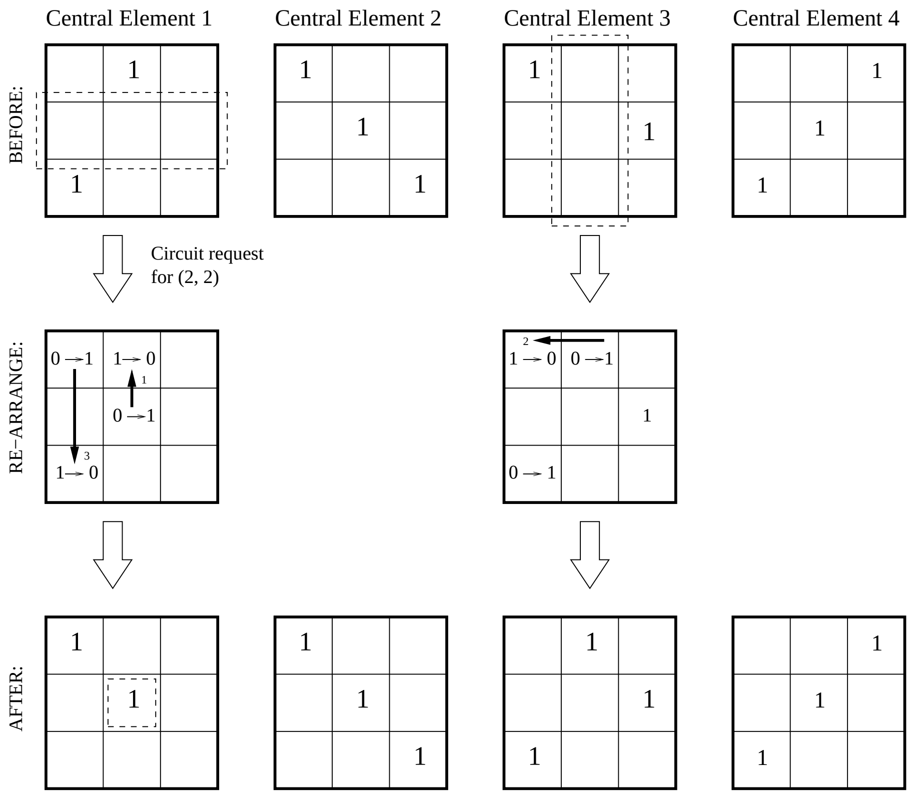


Figure 3.2: Example of Slepian-Duguid-based circuit re-arrangement

Note that, much like Clos fitting, this algorithm can also be used to partition an aggregate request for simultaneously admissible circuits into $K$ configurations. However, since the requirement on $K$ is lower, the number of partitions, in the worst case, is about half as that with Clos fitting. Note also that the ability to fit aggregate requests that are admissible, i.e., there are no more than $N/M$ circuits originating (terminating) from (to) each first (third) stage element, as long as $K \geq N/M$ is a direct consequence of the combinatorial result [43] that an integer matrix with maximum line sum of $n$ can be expressed as a sum of $n$ permutations.

## 3.2 Matching Algorithms for Input-Queued Switches

Since IQ and CIOQ packet switches require a memory bandwidth that is independent of the switch size, a significant amount of research has been conducted to devise and analyze matching algorithms for them. Recall that the properties of the matching, used to configure the space element of such a switch, dominate the overall performance in terms of both switch throughput and its ability to provide bandwidth and delay (QoS) guarantees. Accordingly, the primary goals in the literature have been to devise matchings that can ensure asymptotic 100% throughput [29, 70, 79, 80] for all admissible arrivals (2.2), exact 100% throughput [72, 114] for all bounded admissible arrivals (2.3), virtual bandwidth trunks [7, 46, 58, 100] for admissible requested rates (2.4) independent of arrivals, or an exact emulation [24, 65, 107] of a high-performance switch. Further restrictions are sometimes placed on arrivals, e.g., stationarity of the processes, at the expense of weakening the final results. Note that asymptotic 100% throughput above refers to the ability to maintain queue stability, and exact 100% throughput to the ability to bound the queue lengths and cell delay.

We cover two questions in this section: how are matchings computed, and how does a sequence of matchings translate into throughput and QoS properties of the switch. The answers to the first are rooted in classical graph theory, while the latter is aided by more recent advances in combinatorics and stochastic stability. We shed light not only on the final results but also on the methods used to arrive at them, as similar methods will be used in our own analyses in later sections.

### 3.2.1 Switch Control Loop

For the sake of brevity, we collectively refer to IQ and CIOQ switches as input-queued switches, and consider the former as a special case with speedup $s = 1$. Let the model shown in Fig. 2.6 be the common reference for such switches, with all the rates normalized so that $C = 1$. As a starting point, we assume virtual output queueing (VOQ) in all input-queued designs. Since the only internal contention lie at the inputs of the space element, the analysis typically focuses on the VOQ system, and not on the output memory elements. For example, for admissible arrivals, it is clear that the queues, if any, at a work-conserving output will remain stable irrespective of the matching algorithm.

Let $A(n)$ denote a discrete-time arrival matrix, where $n$ refers to the external timeslot of the switch. Let $\lambda$ denote the offered rate matrix and $R$ the required rates, the latter being

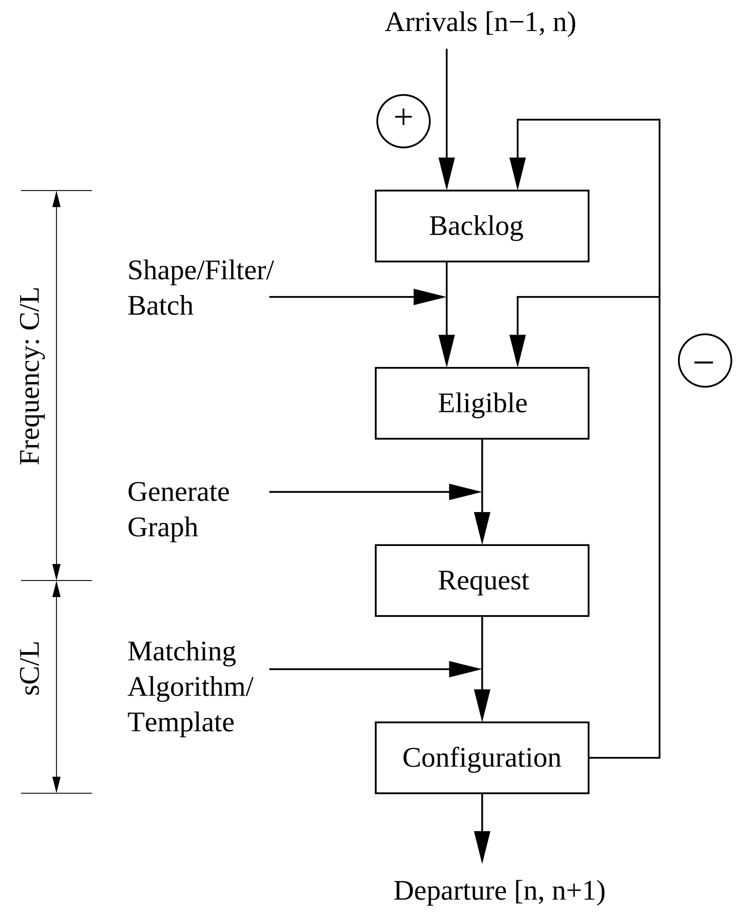


Figure 3.3: Control loop for an input-queued switch

necessarily admissible. The cumulative amount of traffic departing from the VOQs in the input elements, for input-output flows $(i, j)$, are maintained by matrix $D(n)$. Assuming an initially empty switch, the backlog in the VOQ system may be calculated as

$$Q(n) = A(n) - D(n). \tag{3.2}$$

The analytical task then is to characterize $D(n)$ in terms of the chosen matching algorithm, and given properties of $A(n)$, establish properties of $Q(n)$. The departure matrix is governed by the switch control loop illustrated in Fig. 3.3. The evolution of the backlog matrix is controlled by (3.2). The matrix $Q(n)$ may be filtered (shaped or batched) into an eligible matrix

$\Pi(n)$, where

$$\forall n \;\; \Pi(n) \leq Q(n)$$

refers to the subset of cells in the backlog that are to be considered by the arbitration policy. The filtering stage may be skipped in which case the above becomes an equality. The eligible matrix may be viewed as an aggregate request matrix, i.e., if $\Pi_{i,j}(n) = m$, it denotes a simultaneous request for $m$ connections between $(i, j)$.

In each internal timeslot, matchings are generated using templates computed off-line (e.g., based on $R$), using the state of aggregate requests, or a combination thereof. To compute the matching, $\Pi(n)$ may be converted into a bipartite graph $G : (P, Q, E)$, where $P$ contains nodes corresponding to the input links, $Q$ the output links, and $E$ is the edge-set represented by the following adjacency matrix:

$$G_{i,j}(n) = \begin{cases} 1 & \text{if } \Pi_{i,j}(n) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Note that this is a truncated adjacency matrix in which the rows correspond to $P$ and columns to $Q$. This suffices as there are no edges between the vertices in $P$, and between those in $Q$. Some matchings use a weighted graph, where the weights may correspond to $R$ or $\Pi(n)$. The matching $\pi(n)$ is a sub-graph of $G(n)$, such that no more than one edge is incident on each vertex. The cardinality of a matching is given by $|\pi| = \sum_{i,j} \pi_{i,j}$. Specific algorithms yield specific properties for $\pi(n)$, which configures the space element and determines $D(n)$, and thereby, the long-term state of the system. Since the lower part of the control loop operates at a speedup of $s$, the departures may be computed as

$$D(n) = \sum_{k=1}^{ns} \pi(k). \tag{3.3}$$

### 3.2.2 General Matching Techniques

#### Maximal Matching

A matching $\pi(n)$ is referred to as *maximal*, or non-idling, if no edge from $G(n)$ can be added to it. In other words, if $G_{i,j} = 1$, then $\pi_{i,j} = 1$, $\pi_{i,k} = 1$ for some $k$, or $\pi_{k,j} = 1$ for some $k$. This is one of the most popular policies in switching due to its simplicity. In fact, a greedy algorithm that visits the entries in $G$ one by one, in any order, is sufficient to realize such a matching. The entire row $i$ and column $j$ are marked as ineligible for further consideration when $\pi_{i,j}$ is set to 1. As it visits each entry, skipping ineligible rows and columns, the algorithm simply sets $\pi_{i,j}$ to 1 whenever $G_{i,j} = 1$. Clearly, this sequential greedy algorithm has a run-time complexity of $O(N^2)$, where $N$ is the size of the switch. If the adjacencies are maintained by an edge list, this easily reduces to $O(N + |E|)$.

A simple parallel algorithm, based on the exclusive-read exclusive-write (EREW) PRAM[1] model, may be used to compute the matching in $O(N)$ time, using $N$ processors. Each row $i$ of $G$ is inspected in parallel until a non-zero value is found in an eligible column. The eligibility state is shared by all processors, and to ensure EREW operation on the column state, the parallel inspections are conducted in a staggered manner, specifically, the processor at row $i$ starts at entry $G_{i,i}$ and each processor works in lock step. Manifestations of such a technique may be found in distributed implementations such as iSLIP [78]. More complex schemes have been devised to obtain a maximal matching in sub-linear time, e.g., the deterministic Israeli-Shiloach algorithm [48], based on Euler tour computations, has a complexity of $O(\log^3 N)$ using $(N + |E|)$ processors.

## Maximum-Size Matching

A maximum-size matching (MSM), as the name suggests, has the maximum cardinality. A space element configured by an MSM has the maximum *instantaneous* throughput. An MSM may be obtained by incrementally finding an augmenting path to the current matching [86, Chap. 10], essentially by searching for paths that terminate in unconnected vertices and consist of edges that are alternately free and matched. Since $\max(|\pi|) = N$ and each augmentation may be completed in $O(N + |E|)$ time, for dense bipartite graphs, this algorithm has a run-time complexity of $O(N^3)$.

Alternative approaches to MSM rely on computing the maximum flow in an equivalent network (see, again, [86] for details), an example of which is illustrated in 3.4. The classic

[1]The EREW PRAM is a model for a parallel computer in which multiple processors cooperate in the same task, and share a common state in random-accessible memory. Access to the memory is restricted so that multiple processors may not read or write to the same location at the same instant.

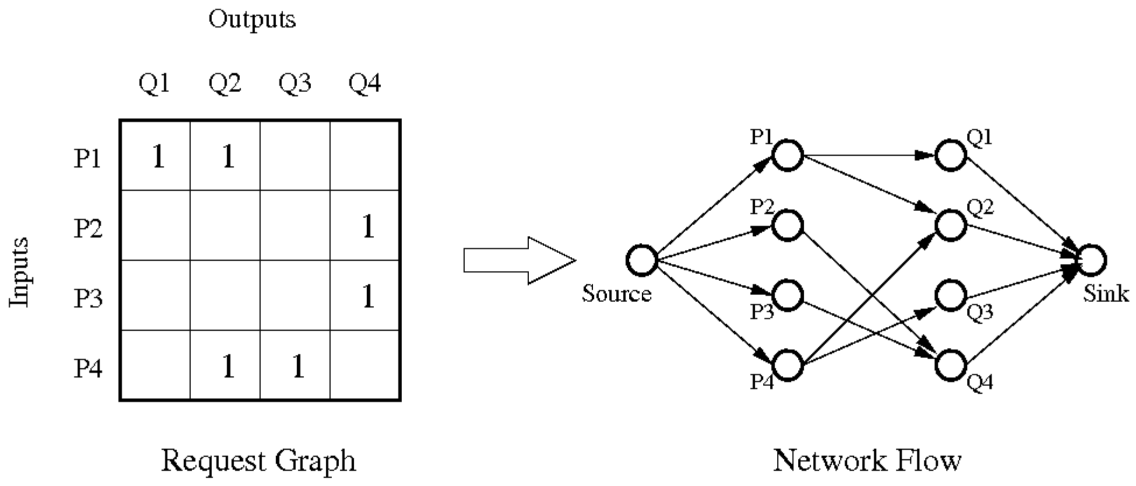


Figure 3.4: Maximum Size Matching using an equivalent flow network

Ford-Fulkerson method [27, Chap. 27] is based on $O(|\pi|)$ steps of finding augmenting paths in the residual flow network, and runs in $O(|\pi||E|)$ time, which in our case is again $O(N^3)$. The Edmonds-Karp implementation [27] uses a breadth-first search to obtain those paths. An improvement, due to Karzanov [86, Chap. 9], uses the concept of pushing so-called preflows through vertices, in order to find several augmenting paths in the auxiliary residual network, in the same step. The Hopcroft-Karp [44] algorithm is a variation of this method, applied to bipartite graphs with unit capacities, and runs in $O(N^{2.5})$ time. Notice that both the above approaches, namely, finding an alternating path and finding a maximum flow through augmentations, are quite similar in philosophy to the Slepian-Duguid re-arrangement method.

There are several randomized parallel algorithms (e.g., [84]) for the EREW PRAM model that compute a maximum matching in poly-logarithmic time using a polynomial number of processors, however, no similar deterministic solution is yet known for general bipartite graphs.

**Critical and Perfect Matchings**

While an MSM provides maximum instantaneous throughput to the space element, it has been known for some time, and formalized recently in [60], that a maximum matching may be insufficient to sustain maximum throughput. The intuition behind this observation is that if there are

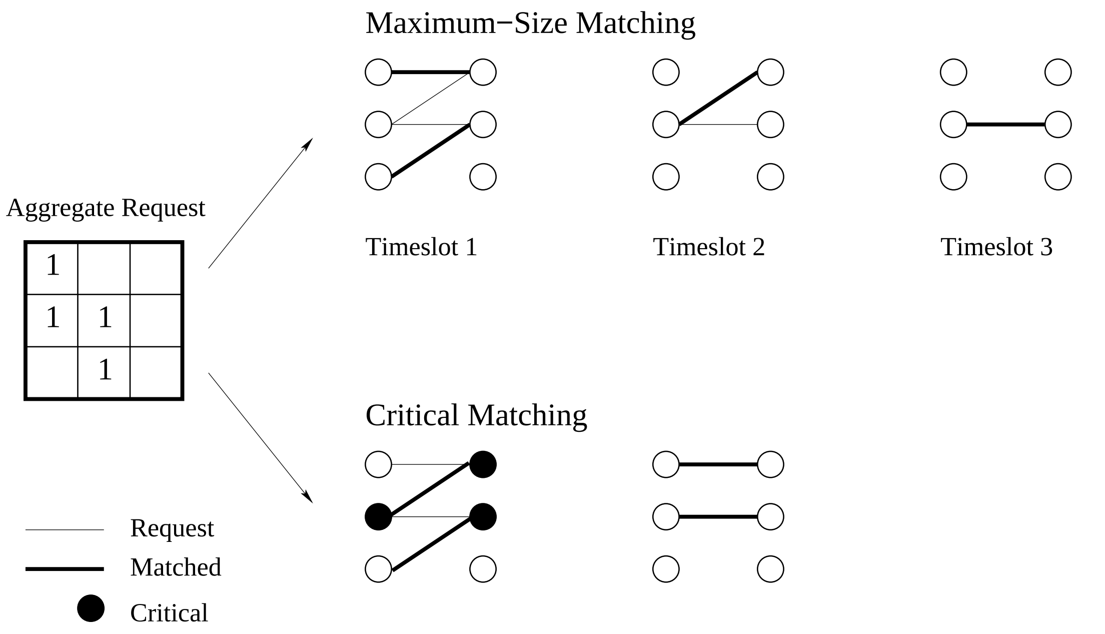


Figure 3.5: Maximum Size Matching versus Critical Matching

several maximum matchings to choose from, an injudicious choice hampers future matchings. This is illustrated in a rather simple example, in Fig. 3.5. The top row is a maximum matching in each step, but an incorrect choice in the first step leads to a requirement for a total of three matchings to cover $\Pi$, whereas a better choice in the second row can accomplish the same in two. Note that the extra timeslots required can be made arbitrarily large for admissible arrivals by increasing the timescale (2.3) of admissibility.

The above insufficiency necessitates additional properties on the matching. A node in $G$ is referred to as *critical* if the corresponding line sum in $\Pi$ is maximum. Let $R_i = \sum_j \Pi_{i,j}$ denote the row sum corresponding to input node $i$, and $C_j = \sum_i \Pi_{i,j}$ denote the column sum corresponding to output node $j$. Then, the maximum line sum is given by $M = \max_{i,j}\{R_i, C_j\}$. Every input (output) node of $G$ for which the corresponding row (column) sum is equal to $M$ is critical. A matching that covers all the critical nodes is referred to as a *critical matching*. The second row of Fig. 3.5 satisfies this requirement. A critical matching can be obtained by using the Von Neumann augmentation procedure (e.g., see [7]) to generate a new graph $G'$, computing a maximum matching $\pi'$ on the latter, and retaining only those edges that belong to $G$. The procedure adds dummy entries to $\Pi$ so that each row and column sums up exactly to $M$. Any resultant maximum matching is guaranteed to be a *perfect matching*, i.e., every node of $G$ is covered. As no dummy entries were added to the lines corresponding to the critical nodes, the final matching is guaranteed to cover them. Since the Von Neumann procedure can be completed in $O(N^2)$ time, the run-time complexity of the above algorithm remains $O(N^{2.5})$.

An MSM that covers all the critical nodes is referred to as a *critical maximum-size matching* (C-MSM) [114, 52]. It is unclear whether it is possible to find a matching that is simultaneously maximum and critical in polynomial time. For example, a fully-connected graph has $N!$ different maximum matchings to inspect. An alternative approach, called *longest port first* (LPF) [79] matching, is similar in concept to the above. In LPF, each edge of $G$ is assigned a weight which equals the sum of the line sums of the connected vertices, and the algorithm systematically searches for a maximum-size matching with the highest total weight. The Edmonds-Karp algorithm was modified, without adding complexity, to conduct the search.

**Maximum-Weight Matching**

Given a matrix $w$ that assigns weights to each edge of $G$, a maximum-weight matching (MWM) $\pi$ maximizes $\sum_{i,j} w_{i,j}\pi_{i,j}$, the sum of the weights of connected edges. Note that this is not the same as an MSM with the maximum weight. In fact, the cardinality of an MWM need not be maximum. Such matchings have been put to use in switching theory (e.g., [29, 80], where $w$ is set to $Q$) in order to overcome the limitations of MSM. Solving for an MWM may be viewed as a linear programming problem, with constraints expressed by (2.5) and $\pi_{i,j} \geq 0$. The goal would be to minimize $\sum_{i,j} c_{i,j}\pi_{i,j}$, where $c_{i,j} = \max_{i,j}\{w_{i,j}\} - w_{i,j}$. The Hungarian simplex method [86] provides a feasible solution in $O(N^3)$ time, while ensuring integral values for $\pi_{i,j}$, however, it has been found to be too complex to implement in the fast path of a switch and infeasible to parallelize.

The best parallel algorithm to date [32] is based on finding augmenting paths to an existing weighted matching, and runs in $O(\sqrt{n}\log^2 n)$ time using $O(N^3)$ concurrent-read concurrent-write (CRCW) processors. Table 3.1 summarizes the existing techniques to compute various

| **Matching** | **Sequential** | **Deterministic Parallel** |
|---|---|---|
| Maximal | Greedy assignment: $O(N^2)$ | Israeli-Shiloach: $O(\log^3 N)$ CRCW |
| MSM | Hopcroft-Karp: $O(N^{2.5})$ | Fayyazi: $O(\sqrt{n}\log^2 n)$ CRCW None for EREW |
| Critical | Von-Neumann plus MSM: $O(N^{2.5})$ | |
| MWM | Hungarian method: $O(N^3)$ | |

Table 3.1: General matching techniques and their complexity

matchings and their complexity. It is worth noting here that neither MWM nor any flavor of MSM is commonly found in commercial switches today, the preference being given to maximal and various sub-maximal heuristic matchings.

### 3.2.3 Deterministic Properties

The combinatorial properties of a sequence of matchings may be applied to yield deterministic results on the departure process. Such results in the literature are primarily of two types. The goal of the first is to achieve exact 100% throughput for bounded admissible arrivals using an on-line matching, while the second aims to ensure virtual bandwidth trunks that satisfy an admissible rate matrix by generating off-line templates, which are then used in a pre-determined sequence.

#### Exact 100% Throughput

We say that a switch provides *exact* 100% throughput if every cell is guaranteed to depart with a bounded maximum delay. Weller and Hajek [114] showed that such a property can be achieved for bounded admissible[2] traffic, without explicit knowledge of the traffic rates, by computing suitable matchings in *batch-mode*. Specifically, let $T$ be the timescale of admissibility. The time axis is marked by epochs $\{T, 2T, \ldots, kT, \ldots\}$. The arriving cells are batched at every epoch, i.e., the aggregate request matrix $\Pi$ is set to $Q$ at the beginning of an epoch and never incremented within an interval. Clearly, if all the requests in $\Pi(kT)$ can be covered by the matching sequence in $(kT, (k+1)T)$, the maximum delay for each cell is upper bounded by $2T$. Furthermore, the

[2]The $(\alpha, S)$ traffic model in [114] generates bounded traffic whose constraint is identical to (2.3). For uniformity, we will continue to use our own definitions.

following inequality is ensured for $\Pi$:

$$\Pi(kT) = A[(k-1)T, kT) \quad \leq \lambda T \tag{3.4}$$

The task then is to partition $\Pi$ into a sequence of conflict-free configurations for the space element, to be completed before the next epoch. The choice of a suitable matching is governed by basic results in combinatorics [43] (similar to the basis for the Slepian-Duguid theorem). Recall that an integer matrix with maximum line sum $M$ can be decomposed as a sum of $M$ permutations, which is a direct consequence of Hall's theorem of the system of distinct representatives. Moreover, each of the $M$ permutations has to necessarily include the critical rows and columns. Since the maximum line sum of $\Pi$ never exceeds $T$, due to (3.4) and the admissibility of $\lambda$, a sequence of C-MSM is guaranteed to drain $\Pi$ in $T$ steps. We highlight this result as follows, paraphrased to suit our terminology.

**Theorem** (Weller-Hajek, C-MSM)**.** *A critical maximum-size matching is sufficient to ensure exact 100% throughput for an IQ switch without speedup, for bounded admissible arrivals.*

The same work [114] also explored the applicability of maximal matchings for exact 100% throughput. Let the maximum row sum of a matrix be denoted by $R$, and the maximum column sum by $C$. It can be easily shown (e.g., by contradiction) that a sequence of $(R + C - 1)$ maximal matchings can cover all the elements of the matrix. Therefore, $\Pi$ can be drained in $(2T - 1)$ steps by a maximal matching. Recognizing that those steps need to be completed in $T$ timeslots leads us to the following result.

**Theorem** (Weller-Hajek, Maximal Matching)**.** *A maximal matching is sufficient to ensure exact 100% throughput for an input-queued switch, for bounded admissible arrivals, as long as the internal speedup is 2.*

It is not clear why the Weller-Hajek C-MSM result insists on a maximum matching. In fact, we re-state this result in Chapter 5 by showing that any critical matching suffices, including a sequence that possibly contains sub-maximum cardinality matches. Note that Clos fitting, with a speedup of 2, and Slepian-Duguid fitting, without speedup, can also partition $\Pi$ into $(2T-1)$ and $T$ configurations, respectively, by viewing the entries in $\Pi$ as simultaneously admissible circuits

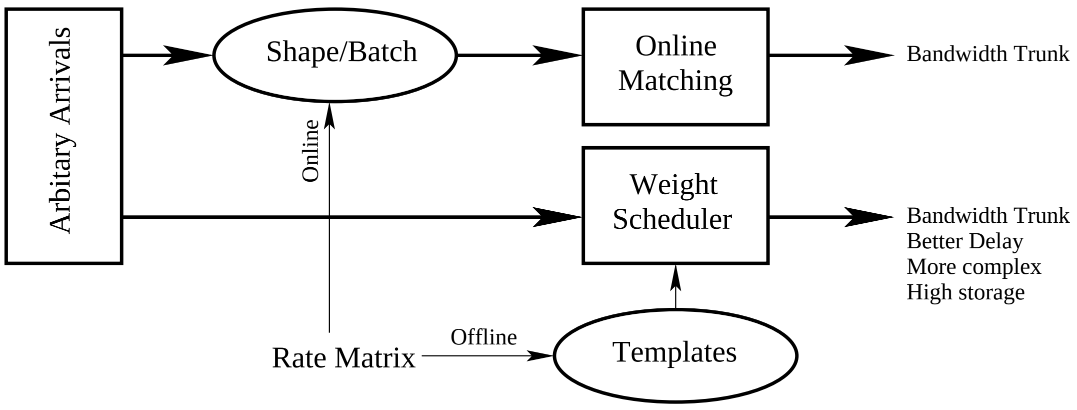


Figure 3.6: Bandwidth and delay guarantees: Exact 100% throughput versus templates

and the internal timeslots as separate space elements. Nevertheless, both these procedures do not operate in sequence, and hence cannot be used in lieu of on-line matchings. Li and Ansari [72] propose the store-sort-forward (SSF) method, which may be considered as a perfect matching algorithm on an augmented $\Pi$. However, instead of directly computing a perfect matching, the method uses a re-arrangement technique in each timeslot, reminiscent of the Slepian-Duguid re-arrangement, in order to arrive at a permutation.

None of the above works explore the possibility of translating the exact 100% throughput property into providing bandwidth and delay guarantees, based on an independent request matrix $R$, for arbitrary arrivals. We make this leap, also in Chapter 5, by recognizing that $R$ can be used to impose a timescale, and boundedness can be replaced by explicit shaping. Smiljanic [100] addresses this subject and provides a credit-based scheme for the allocation of bandwidth using on-line matching, which mirrors the Weller-Hajek maximal result. Fig 3.6 shows how the exact 100% throughput approach compares with rate-based templates in providing bandwidth and delay guarantees.

**Rate-based Templates**

Given an admissible required-rate matrix $R$, e.g., by the admission control procedure that negotiates QoS, similar matrix decomposition techniques can be used to generate a repeating sequence of matchings that ensure a virtual bandwidth trunk of $R_{i,j}$ to each input-output flow $(i, j)$. Note that such a trunk is ensured, providing isolated bandwidth guarantees to each flow, as long as there exists a finite interval $T$ such that in each interval, the sequence allocates $R_{i,j}T$ turns to the respective flows. If $T'$ is the number of matchings in the sequence, then the internal speedup required is equal to $T'/T$. The maximum waiting time for a flow, or the latency of scheduling, is guaranteed never to exceed $T$. Therefore, the arbiter essentially works as a two-dimensional latency-rate [3] (LR) [104] scheduler. The sequence of matchings, and hence the bandwidth trunk, is independent of the actual arrivals. In fact, they are generated off-line whenever $R$ changes, and stored as a set of templates. The switch may store one template for each matching in the sequence, for a total of $T'$, or preferably, a smaller set with associated weights that indicate the number of times each matching is to be used in a repeated sequence.

Such templates may be readily generated using Clos or Slepian-Duguid fitting on a request matrix based on $R$. This was applied to ATM switches in [46]. Let $r$ be a bandwidth value such that $R_{i,j}/r$ is an integer for all $(i, j)$. The general approach is to generate an aggregate request matrix given by

$$\Pi^{(r)} = \frac{1}{r}R.$$

Such a matrix can be decomposed into $(2/r - 1)$ conflict-free configurations by Clos fitting, and into $1/r$ by Slepian-Duguid fitting. If these configurations are used as a sequence completed in $1/r$ time, each flow is guaranteed $R_{i,j}/r$ turns in that interval. Consequently, the former requires a speedup of $(2 - r)$ and the latter does not require any speedup. Note that both the approaches require a storage of $O(1/r)$ templates. Furthermore, since there are at most $N/r$ entries in $\Pi^{(r)}$, template generation using Clos fitting runs in $O(N/r^2)$ time, while Slepian-Duguid fitting runs in $O(N^3/r + N/r^2)$ time.

The biggest drawback of the above is that both the storage and time complexity depend on $r$. To overcome this, Chang et al. proposed [7] the Birkhoff-Von Neumann (BVN) decomposition method to efficiently generate templates. It is based on the combinatorial result, due to G. Birkhoff, that a doubly stochastic matrix[4] $R'$ can be expressed as a weighted sum (convex

[3]A latency-rate server, with latency $L$ and service rate $R$, ensures service to a backlogged flow, lower bounded by a service curve of slope $R$ and $x$-intercept $L$, where the origin denotes the most recent time instant when the flow became newly backlogged.

[4]Note that a doubly stochastic matrix is defined as one in which the entries in each row and column add up to

combination) of $k$ permutations, i.e.,

$$\forall R' \ \exists k, \phi_k \ \text{ s.t. } R' = \sum_k \phi_k P_k, \tag{3.5}$$

where $P_k$ is a permutation, and $\sum_k \phi_k \leq 1$. The request matrix $R$ may be converted into a stochastic matrix $R'$ by augmenting each row and column, using the Von Neumann procedure. The task then is to decide the permutations and associated weights to use, in order to cover $R'$. This can be achieved systematically by repeated maximum matchings, which are also perfect, on the residue of $R'$ in order to generate a set $\{P_k\}$. The weight $\phi_k$ is set to the minimum value among the entries covered by $P_k$, and $R'$ is decremented in each step as follows:

$$R'_{k+1} = R'_k - \phi_k P_k$$

This procedure is guaranteed to terminate in $(N^2-2N+1)$ steps. Since each step is dominated by an MSM, the decomposition runs in $O(N^{4.5})$ time. The advantage of this approach, as opposed to performing Slepian-Duguid fitting or critical matching on $\Pi^{(r)}$, is that the number of templates generated is $O(N^2)$ and independent of $r$. The generated weights $\phi_k$ can be used in an on-line scheduler that chooses the respective $P_k$ in accordance with those weights. Note that the latency would depend on the order in which those permutations are picked, and in the worst case, remains $1/r$, comparable to Clos or Slepian-Duguid fitting.

**Theorem** (Chang et al., BVN Decomposition)**.** *The BVN decomposition procedure on an admissible required rate matrix $R$ is sufficient to ensure bandwidth guarantees, specified by $R$, to input-output flows in an input-queued switch.*

Towles and Dally propose [111] a few approaches to further reduce the number of templates, at the expense of some speedup. Referring to (3.5), observe that each flow $(i, j)$ will continue to receive its required allocation if $R'$ is *no greater* than $\sum_k \phi_k P_k$, the penalty being a wastage in some of the configurations and a speedup of $\sum_k \phi_k$. This fact may be used to generate fewer permutations, with a higher total weight. Since, in general, there are $N^2$ distinct entries in the request matrix, and each permutation covers exactly $N$ of them, the minimum number of

exactly 1. On the other hand, if the entries are less than 1, the matrix is referred to as doubly sub-stochastic. The Birkhoff result also applies for sub-stochastic matrices where each row and column sums up to the same value.

| **Matching** | **Complexity** | **Speedup** | **Property** |
|---|---|---|---|
| Maximal (Weller-Hajek) | $O(N^2)$ per slot | 2 | Exact 100% throughput |
| C-MSM (Weller-Hajek) | $O(N^{2.5})$ per slot | 1 | Admissible traffic only |
| SSF (Li-Ansari) | $O(N^3)$ per slot | 1 | |
| Clos fitting | $O(N/r^2)$ | 2 | Templates: $O(1/r)$ |
| Slepian-Duguid | $O(N^3/r + N/r^2)$ | 1 | No on-line scheduling |
| BVN (Chang et al.) | $O(N^{4.5})$ | 1 | Templates: $O(N^2)$ |
| MIN (Towles-Dally) | $O(N^{3.5})$ | $4(4 + \ln N)$ | Minimum templates: $N$ |
| GLJD (Keslassy et al.) | $O(N^3)$ | $2 \ln N + 1$ | Heuristic/Templates: $2N$ |

Table 3.2: Deterministic properties of popular matching algorithms

permutations required is $N$. The linear programming problem of minimizing the total weight (and hence the speedup) while generating exactly $N$ templates to cover or exceed $R'$ is known to be NP-hard. In [111], the authors recognize that the speedup required is $\Omega(\log N)$, and propose an algorithm called MIN that generates $N$ templates, with a speedup that is not minimum, yet is $O(\log N)$. In step $k$, MIN covers entries in the residual request matrix that exceed $1/2k$ using edge coloring on the corresponding bipartite graph. Clearly, there are $\log N$ number of steps and each step is guaranteed to return exactly $k$ permutations. Since each of the latter runs in $O(N^{2.5})$ time, MIN runs in $O(N^{3.5})$ time. The same work also proposes an algorithm called DOUBLE, which generates $2N$ templates and requires a speedup of 2. These are all variants of the classical open-shop problem in operations research.

Keslassy et al. [58] make the observation that a decomposition that returns exactly $N$ permutations can be applied to yield a latency much less than $1/r$ that is characteristic of template-based schemes. Notice that, in this case, an entry of the original request matrix is covered by exactly one permutation. As a result, an on-line scheduler that picks those templates in accordance with their weights can tightly control the delay of each flow $(i, j)$ by favoring the unique template that covers that flow. A heuristic, called *greedy low-jitter decomposition* (GLJD), is proposed in [58] for this minimum permutation problem, which runs in $O(N^3)$ time.

We conclude by summarizing some of the notable methods overviewed in this section in Table 3.2. In practice, the problem of providing virtual bandwidth trunks in an input-queued switch is usually solved by some variation of the BVN approach. Accordingly, we will consider it as the current basis for providing QoS in such switches.

### 3.2.4 Stochastic Stability

While the deterministic results presented above provide the means to ensure bandwidth trunks specified by an admissible $R$ independent of the offered $\lambda$, or equivalently, exact 100% throughput if the latter is admissible and bounded within a finite timescale, neither address maximization of throughput for unknown arbitrary $\lambda$. Indeed, if the entire traffic consists of flows for which the requested rates are specified, and furthermore the offered rates conform to those requests, then using such means immediately translate to maximizing throughput. However, since the best-effort component forms a significant majority of the offered traffic in most switches, stronger results are required that address arbitrary arrival processes with unknown rates.

There are several significant results in the literature that address this problem for stochastic arrival processes that yield an admissible (though unknown) $\lambda$. Since all the queues in a work-conserving OQ switch, our ideal reference, would remain stable under such arrivals, irrespective of their distributions, an asymptotic 100% throughput may be claimed if the VOQ system of an input-queued switch remains stable, for a given matching algorithm. An explicit determination of the invariant distributions of the queues is usually infeasible for all but the simplest of matchings, not to mention that such analysis would depend on the specific arrival distribution. Accordingly, two techniques in stochastic stability have been used to analyze matching properties, namely, exploring the fluid limit of the system of queues [29], and establishing a positive recurrence of the queue states (e.g., [70, 79, 80]) using control theoretic concepts.

Since no meaningful parameters are usually available for batching the offered traffic, the stability results are typically for matchings conducted in a continuous mode. In order words, $\Pi$ is the same as $Q$ in every timeslot, and the eligibility block in the control loop of Fig. 3.3 is skipped.

#### Maximal Matching and Stability Theory

Dai and Prabhakar [29] established a significant result that any maximal matching is sufficient for asymptotic 100% throughput, for admissible arrivals, as long as the speedup is at least 2. The result was proven using fluid limits. As we will use a similar technique to prove some of our original results in subsequent chapters, it is worth a digression here to see how such limits work.

Given a random variable $f(t)$, a fluid limit of $f$ may be obtained as follows:

$$\bar{f}(t) = \lim_{r\to\infty} \frac{1}{r} f(rt) \tag{3.6}$$

Strictly speaking, the variable may have an explicit dependence on randomness $\omega$, and the limiting parameter $r$ may be drawn from a monotonically increasing sequence $\{r_{n_k}\}$ with the limit converted to $k \to \infty$ (see Sec. A.1 in [29] for such specifics), but the above definition suffices for our purpose. If $f(t)$ is defined in discrete time, it may be converted to a continuous-time function before applying the limit, through appropriate linear interpolations. Then, $f(rt)$ is merely a scaled-time fluid model of $f(t)$. The idea is to analyze the scaled stochastic process, provided the results can be meaningfully applied back to the initially given process.

The above work characterizes the evolution of the switch state using the fluid limits $(\bar{A}, \bar{D}, \bar{Q})$, the only assumptions being that $A(n)$ obeys the strong law of large numbers (SLLN), i.e., $\lambda$ exists, and that $\lambda$ is admissible. The switch dynamics, in the limit, are governed by the following equations, where $(i, j)$ stands for the input-output pairs, and $t \geq 0$.

$$\begin{aligned}
\bar{Q}_{i,j}(t) &= \lambda_{i,j}t - \bar{D}_{i,j}(t) && (3.7)\\
\bar{D'}_{i,j}(t) &= \sum_{\text{all } \pi} \pi_{i,j}\bar{T'}_{\pi}(t), \quad \text{if } \bar{Q}(t) > 0 && (3.8)\\
\sum_{\text{all } \pi} \bar{T}_{\pi}(t) &= st && (3.9)
\end{aligned}$$

The first equation follows from the definition of the limits and (3.2), and the second from a re-formulation of (3.3) in terms of $T_\pi(n)$, which stands for the number of times a specific configuration $\pi$ has been used before and including slot $n$. The last equation just states that some configuration is in use at all times. The evolution of $T_\pi$ depends on the specific matching algorithm used, which may yield additional equations expressing its specific properties. The fluid model solution, once obtained, is meaningfully applied back to the initial process using the following result:

**Theorem** (Dai-Prabhakar, Fluid limit)**.** *A switch operating under a matching algorithm is rate-stable if in the corresponding limiting fluid model, every solution with* $\bar{Q}(0) = 0$ *gives* $\bar{Q}(t) = 0$

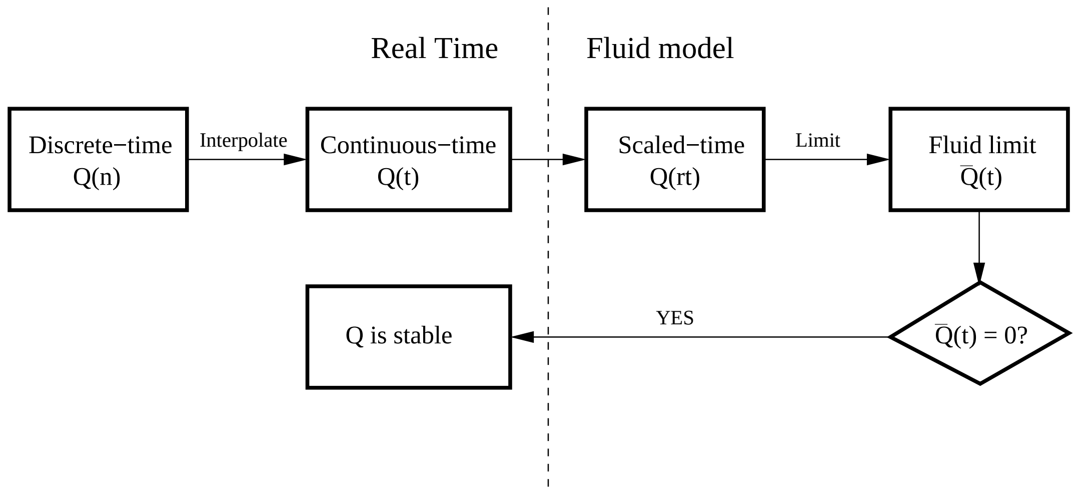


Figure 3.7: Fluid limits: Steps in analyzing queue stability

*for all* $t \geq 0$.

Note that rate-stability, being used here to specify asymptotic 100% throughput for all $(i, j)$ and admissible $\lambda$, is achieved when the departure processes satisfy

$$\lim_{n\to\infty} \frac{D_{i,j}}{n} = \lambda_{i,j}.$$

The above theorem is evident by setting $t = 1$ in (3.7) and expanding $\bar{D}(1)$ by using the fluid limit definition (3.6). Accordingly, the task in proving stability boils down to showing that every fluid limit solution yields $\bar{Q}(t) = 0$, as shown by the decision chart in Fig. 3.7. Conversely, if $\lambda$ is inadmissible, the average departure rate for a subset of queues will become strictly less than $\lambda_{i,j}$, resulting in a non-zero $\bar{Q}$ for that subset. While stability may be ascertained using this approach, it is worth noting that most of the other statistics disappear in the fluid model. For example, this technique cannot be used to determine average queue lengths and waiting times.

Remarkably, a number of combinatorial results on invariant matrices become directly applicable to the slow-changing world of fluid limits. For instance, let $C_{i,j}$ be defined as the sum of the row $i$ of $Q$ plus the sum of column $j$, i.e.,

$$C_{i,j}(n) = \sum_k Q_{i,k}(n) + \sum_k Q_{k,j}(n).$$

In [29], the authors recognize that a maximal matching decreases the above quantity by at least one in every internal timeslot, whenever $Q_{i,j} > 0$. Since $s$ matchings are computed for each external slot, and $\lambda$ is admissible, the following is the fluid limit derivative of $C$, and it cannot exceed zero if $s \geq 2$.

$$\bar{C}'_{i,j}(t) = \sum_k \lambda_{i,k} + \sum_k \lambda_{k,j} - s \qquad \text{if } \bar{Q}_{i,j}(t) > 0 \tag{3.10}$$

Using the above, and some basic algebra, the product $f(t) = \sum_{i,j} \bar{Q}_{i,j}(t)\bar{C}_{i,j}(t)$ is shown to be zero, by proving that $f(t) > 0$ immediately implies $f'(t) \leq 0$. This leads to the result that $\bar{Q}_{i,j}(t) = 0$ for all $(i, j)$, thus establishing rate stability for the switch under maximal matching, with a speedup that is at least 2. Paraphrased below, this result is the stochastic analog of the Weller-Hajek maximal matching theorem.

**Theorem** (Dai-Prabhakar, Maximal Matching)**.** *Any maximal matching algorithm ensures asymptotic 100% throughput for a CIOQ switch, under admissible arrivals, as long as the internal speedup exceeds 2.*

If the arrival processes are such that the evolution of the system of queues can be described by an irreducible discrete-time Markov chain (DTMC), an alternative approach to stability, and hence for asymptotic 100% throughput, is to establish positive recurrence of the system state. This goal may be achieved, without knowledge of the specific arrival distribution, by constructing a quadratic Lyapunov function [68] of the queue lengths that has a negative drift whenever the norm of the queue lengths is large.

Specifically, let the state of the system be described by $Z(n)$, an $N^2$-entry row vector of VOQ lengths, i.e., for all $n$ (ignoring the time argument for notational simplicity),

$$Z = (Q_{1,1}, Q_{1,2}, \ldots, Q_{1,N}, Q_{2,1}, Q_{2,2}, \ldots, Q_{N,N})$$

The norm of the vector, representing the cumulative level, is given by $\|Z(n)\| = \sqrt{\sum_{i,j} Q^2_{i,j}(n)}$. Using any suitable $N^2 \times N^2$ symmetric copositive matrix $L$, a Lyapunov function may be constructed from the queue state as follows:

$$V(Z(n)) = Z(n)LZ^T(n)$$

$$\text{where } L = L^T \text{ and } (x_1, \ldots, x_K)L(x_1, \ldots, x_K)^T \geq 0, \text{ whenever } x_i \geq 0 \text{ for all } i$$

Notice that $V(Z(n))$ is merely a weighted cartesian product of the vector $Z(n)$ with itself, the respective weights being assigned by $L$. The second line above expresses the symmetric copositive property of $L$. Note also that the identity matrix is a trivial instance of $L$. Recall that a single-queue DTMC with state variable $s(n)$, is positive recurrent, due to Foster's criterion [82], if there exists a positive function $f(s)$ with a negative drift. More precisely, if there exist $f(s)$, $\epsilon > 0$, and a set $B$ such that the following is satisfied, the queue is stable.

$$E[f(s(n+1))|s(n)=i] < \infty \quad \text{and}$$
$$E[f(s(n+1)) - f(s(n))|s(n)=i] < -\epsilon, \quad i \notin B.$$

This result can now be extended to a system of queues, by establishing a similar drift for a Lyapunov function of $Z(n)$. The choice to explore a Lyapunov function, as opposed to, say, a positive function of the norm of the queue-state vector, is due to the former's flexibility. Specifically, the copositive matrix may be chosen depending on the matching being analyzed. Several variations of this extension one of which is noted below, all inspired by [68], have been used to prove the stability of the VOQ system of an input-queued switch. Note that the right hand side of the second inequality below can strictly be just $\epsilon$, however, the following formulation is commonly preferred to exhibit that as the cumulative level increases, so does the negative drift.

**Theorem** (Lyapunov Stability)**.** *Given a system of queues with state vector $Z(n)$, whose evolution is described by a DTMC, if there exists a Lyapunov function $V(Z(n))$ such that*

$$E[V(Z(n+1))|Z(n)] < \infty \quad \textit{for all } \; Z(n)$$

*and if there exist $\epsilon > 0$ and $B > 0$, such that $\forall \|Z(n)\| > B$, the following condition holds*

$$E[V(Z(n+1)) - V(Z(n))|Z(n)] < -\epsilon\|Z(n)\|,$$

*then all the states of the DTMC are positive recurrent.*

Leonardi et al., in [70], comprehensively applied the above result to maximal matchings. Several specific maximal algorithms, driven by rates as well as queue lengths, were proven to yield asymptotic 100% throughput with $s \geq 2$, for admissible traffic. The arrival processes for the different input-output flows were assumed to be independent and identically distributed (i.i.d.),

| **Matching** | **Arrival Constraint** | **Speedup** | **Approach** |
|---|---|---|---|
| General Maximal (Dai-Prabhakar) | SLLN | 2 | Fluid limits |
| Specific Maximal (Leonardi et al.) | i.i.d. | 2 | Lyapunov stability |
| LPF (Mekkittikul-McKeown) | i.i.d. | 1 | |
| C-MSM (Iyer-McKeown) | Uniform Bernoulli | 1 | |
| MWM (McKeown et al.) | i.i.d. | 1 | |
| MWM (Dai-Prabhakar) | SLLN | 1 | Fluid limits |

Table 3.3: VOQ stability for admissible arrivals using popular matching algorithms

in order to enforce a DTMC on the queue state. These results are as expected and encompassed by the Dai-Prabhakar maximal matching result.

### Maximum Matchings

One of the earliest results on asymptotic 100% throughput for IQ switches is due to McKeown et al. [80], who showed that a maximum weight matching is sufficient for 100% throughput for admissible i.i.d arrivals, without the need for internal speedup. The weight matrix was set to $Q$, and the Lyapunov stability approach was used in conjunction with the Birkhoff result (3.5), with the copositive matrix set to identity. Subsequently, the i.i.d. assumption was relaxed by Dai and Prabhakar who established the same result using fluid limits [29].

**Theorem** (McKeown et al., MWM)**.** *A maximum weight matching, with the weights set to the respective queue lengths, is sufficient for asymptotic 100% throughput in an IQ switch, without speedup, for all admissible i.i.d. arrivals.*

Using the Lyapunov result with a simple copositive matrix, Mekkittikul and McKeown showed [79] that the less complex LPF matching, which is an MSM with maximum weight, where the weights are set to the row sum plus the column sum of the queue length matrix, also achieves asymptotic 100% throughput for admissible i.i.d arrivals. Iyer and McKeown, in [52], explore whether this result holds for a more general class of MSM, e.g., an analog of the Weller-Hajek C-MSM theorem for non-bounded traffic, and report that a C-MSM provides 100% throughput for Bernoulli i.i.d. arrivals, which are uniformly distributed among all outputs. Actually, we show in Chapter 5 that the analog does apply and prove, using fluid limits, that any critical matching indeed provides asymptotic 100% throughput for all admissible arrivals.

The Lyapunov function methodology, while restricted to i.i.d. arrivals, is superior in comparison to the fluid limit approach in some respects. Specifically, it becomes feasible to gather additional statistics about the queue lengths. Leonardi et al., in [71], use the former to derive bounds on average queue lengths and the variance in lengths for maximum weight matchings under uniform admissible traffic. Similarly, Shah and Kopikare [98] derive bounds on average delay for an approximate MWM under non-uniform Bernoulli traffic. Table 3.3 summarizes the stochastic stability results for cell-based input-queued switches in the literature.

### Packet-based Matching

A few of the above results on stability have been recently extended [77, 36] to packet-mode matchings. The authors of these works realize that it would be beneficial to serve all the cells belonging to a single packet in a consecutive fashion. This is because the delay of a packet depends on when its last cell is transferred by the space element. Marsan et al. [77] suggest a modification to the usual matchings, in which if the first cell of a packet belonging to input-output pair $(i, j)$ is matched in slot $k$, that connection is held for as many slots as the number of cells in the packet. While this connection is held, the pair is deemed *busy*, as opposed to *free*. The matching in each timeslot only considers free pairs. Clearly, a packet-mode maximal matching remains maximal in each slot, and hence continues to provide 100% throughput for admissible arrivals, with $s \geq 2$. However, it is evident that a C-MSM and an MWM operating on the free pairs do not retain their properties in the overall matching, because a few pre-existing connections are imposed in each slot.

Marsan et al. show that the Lyapunov stability result can be modified to consider only a sub-sequence of timeslots, or epochs $\{t_n\}$. Furthermore, the epochs, known as *renewal* points, are chosen such that all the pairs are free in those slots. Using such epochs, it is shown in [77] that MWM is stable for admissible Bernoulli arrivals, with finite average packet sizes. Ganjali et al. [36] demonstrate that it is easy to construct a more general admissible pattern for which such epochs do not exist (except $n = 0$). To overcome this limitation, a variation of MWM is presented in which the system stalls at periodic intervals, waiting for all the pairs to be free,

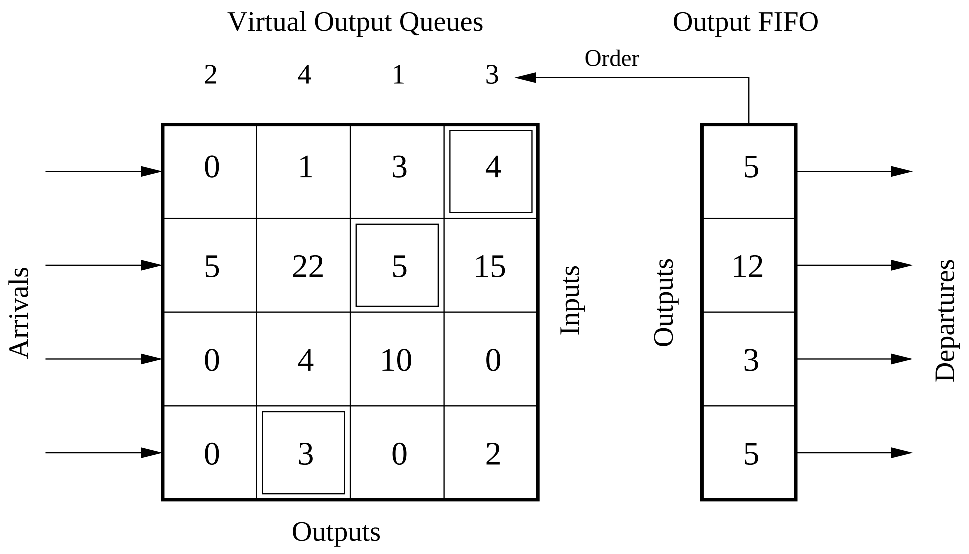


Figure 3.8: Lowest Output Occupancy First: A maximal matching

before proceeding. This variant is shown to ensure stability for all admissible traffic with bounded packet lengths, with a speedup of $(1 + \epsilon)$, where $\epsilon$ can be made arbitrarily small, the trade-off being the length of the forced renewal period.

### 3.2.5 Work Conservation and Exact Emulation

More sophisticated matching algorithms [65, 24, 107], than the ones covered so far, have been proposed to achieve better emulation of an OQ switch as opposed to mere long-term stability for admissible traffic. Such matchings, in general, use the backlog state of the scheduler of the output memory element to drive the matching for the space element. The intuition is that any suitable output scheduler, similar to the one deployed in a reference OQ switch, may be deployed in the output memory element, and the entire switch will behave like the reference switch as long as the cell departures from the switch are governed solely by the actions of the output scheduler. The latter goal may be achieved as long as the scheduler receives the necessary cells from the space element before they are due to be scheduled in the reference switch.

Krishna et al. proposed a specific maximal matching, called *lowest output-occupancy first* algorithm (LOOFA) [65], and proved that for $s \geq 2$, a switch operating under LOOFA for the space element, and a work-conserving scheduler at each output link, is overall work-conserving. This was proven by establishing that the backlog of an output scheduler can never go to zero when a cell destined to that output resides in a VOQ at the inputs. Let $X(n)$ be a column vector that denotes the backlog of respective FIFO schedulers at the output links. In LOOFA, the outputs are first sorted by $X(n)$, and a maximal matching visits the columns[5] of $Q(n)$ in increasing order of $X(n)$ (output selection). An arbitrary input, with a cell to offer, is chosen in each step (input selection), as shown in Fig. 3.8. For a cell $c_{i,j}$ residing at a VOQ $(i, j)$, an *input thread* (IT) is defined as the set of cells in the same input that are destined to a lower or equal-order output. The work-conservation property immediately follows due to the following inequality, which was established in [65] by induction on $n$.

$$X_j(n) \geq |IT(c_{i,j}, n)| \qquad \forall c_{i,j} \in Q_{i,j}(n)$$

**Theorem** (Krishna et al., LOOFA)**.** *A CIOQ switch with a work-conserving scheduler in the output elements, and a space element operating under LOOFA, is work-conserving as long as* $s \geq 2$.

Note that a work-conserving scheme such as LOOFA is superior to those that just ensure stable queues for admissible traffic in at least two aspects. It minimizes the average cell delay because an output link never idles when a cell destined to it resides anywhere in the system, and it isolates any instability, if it exists, i.e., if the offered traffic to output $j$ is inadmissible, it only affects the stability of the queues that contain cells destined to that output. Inspite of its advantages, LOOFA has two limitations. Firstly, the cells may depart in a different order with respect to a reference OQ switch with FIFO schedulers at its output links. Secondly, there is no control on the service allocated to an input-output flow within each output, which is meaningful to isolate instability (or offer fairness) on an input-output pair basis. Note that the latter can readily be offered by a flow-based scheduler in an OQ switch. The authors of LOOFA realize that such limitations may be overcome by improving input selection, e.g., by timestamping the cells to reflect their desired departure, and in each step, choosing an unmatched input with a cell that has the least timestamp. Specifically, the *oldest cell first* (OCF) criterion for input selection

[5]The algorithm described in [65] is implemented in a distributed fashion, with iterative steps of communication between the input and output elements. The centralized version is covered here for simplicity.

is suggested in order to better emulate an OQ switch with FIFO departures. For $s \geq 2$, LOOFA with OCF is shown to bound the difference in delay, with respect to the reference switch, by $2N$ timeslots, and for $s \geq 3$, exactly emulate the latter.

Chuang et al. [24], and independently, Stoica and Zhang [107], showed that it is possible to achieve exact emulation with an OQ switch employing *any* scheduler, with a speedup $s \geq 2$. The idea is to run a parallel simulation of the reference OQ switch (not the same as just running a scheduler in the output element that is identical to the OQ switch scheduler), and generate a timestamp for each cell, which equals the departure time of the cell in the reference switch. These timestamps are then used to create a priority list of outputs in each input, and vice-versa. Such a list can be created for an input $i$ (output $j$), for example, by first keeping the cells in each VOQ sorted by their timestamps, and then sorting the outputs (inputs) based on the timestamps of the head cells $c_{i,j}$. These priority lists are used to create a *stable matching*, between the inputs and outputs, by running the classical Gale-Shapely stable marriage algorithm, which runs in $O(N^2)$ time.

The main property of this algorithm is that if input $i$ is matched to $k$ in a timeslot, then all the outputs ahead of $k$ in the priority list of $i$ are guaranteed to have been matched to inputs better preferred by those outputs. In other words, if a head cell $c_{i,j}$ is not picked, then input $i$ is matched to an output $k$ ahead of $j$ in the input priority list, and/or output $j$ is matched to an input $l$ ahead of $i$ in the corresponding output priority list. Defining the input thread (IT) of $c_{i,j}$ as the cells ahead of it in the input priority list of $i$, and the *output cushion* (OC) as such cells in the output priority list of $j$, it is easy to see that exact emulation follows as long as

$$|OC(c_{i,j}, n)| \geq |IT(c_{i,j}, n)| \qquad \forall n \ \forall c_{i,j} = \text{Head } (Q_{i,j}(n))$$

The authors of [24, 107] show that the above inequality always holds for stable matchings, provided $s \geq 2$, using the property mentioned above. Note that both works assume monotonic scheduling, i.e., a discipline in which the relative order of cells destined to the same output does not change due to newly arriving cells. Presumably, this assumption was made in order to avoid having to re-sort cells in a VOQ in every timeslot. Note also that the VOQ structure was not used at all in [24, 107]. Instead, cells were inserted into a single queue at each input, sorted by timestamps. This queue also served as the input priority list, and a stable matching algorithm

| **Matching** | **Complexity** | **Property** | **Speedup** |
|---|---|---|---|
| LOOFA | $O(N^2)$ | Work conserving | 2 |
| LOOFA-OCF | $O(N^2)$ | Emulates OQ with FIFO | 3 |
| Stable | $O(N^2)$ plus shadow OQ plus VOQ sorting | Emulates OQ with any scheduler | 2 |

Table 3.4: Algorithms for emulation of a reference OQ switch

was run on the contents of the entire queue, as opposed to the heads of the VOQs as described here. We chose to expound the VOQ variation since the stable matching would run in $O(N^2)$ time in this case, compared to $O(B)$ time required for the single-queue version, where $B$ is the total number of enqueued cells. The above inequality holds in either case.

**Theorem** (Stable Matching)**.** *A CIOQ switch can exactly emulate a reference OQ switch using stable matching and a speedup $s \geq 2$.*

Exact emulation using stable matching remains a fundamental result in switching, however, practical implementation of such an algorithm has been hampered by the need to shadow the reference OQ switch in parallel. Recently, Giaccone et al. [38] explored whether a near-work conserving algorithm may be used at a lower speedup to achieve comparable delays as an OQ switch, for a restricted class of traffic. Specifically, the traffic is restricted to arrivals that are bounded admissible with timescale $nT$. A discipline is referred to as $T$-work conserving, if the latter property holds at epochs $\{T, 2T, \ldots, kT, \ldots\}$. It was shown that using essentially a batched critical matching with $n = 2$ (i.e., cells are batched faster than the timescale) and $s \geq 4/3$, the maximum cell delay is bounded by $\frac{3}{2}nT$. Similarly, $n = 3$ and $s \geq 3/2$ yields a delay bound of $\frac{4}{3}nT$. These compare to a delay bound of $nT$ observed in a work conserving OQ switch for similar traffic.

Table 3.4 summarizes the OQ emulation algorithms covered in this section.

### 3.2.6 Low-complexity Matchings

As mentioned in Sec. 3.2.2, several flavors of on-line maximal matching algorithms have been preferred for implementation over maximum size and weight matchings, due to their low complexity. In practice, even the $O(N^2)$ complexity of a sequential maximal matching is sometimes

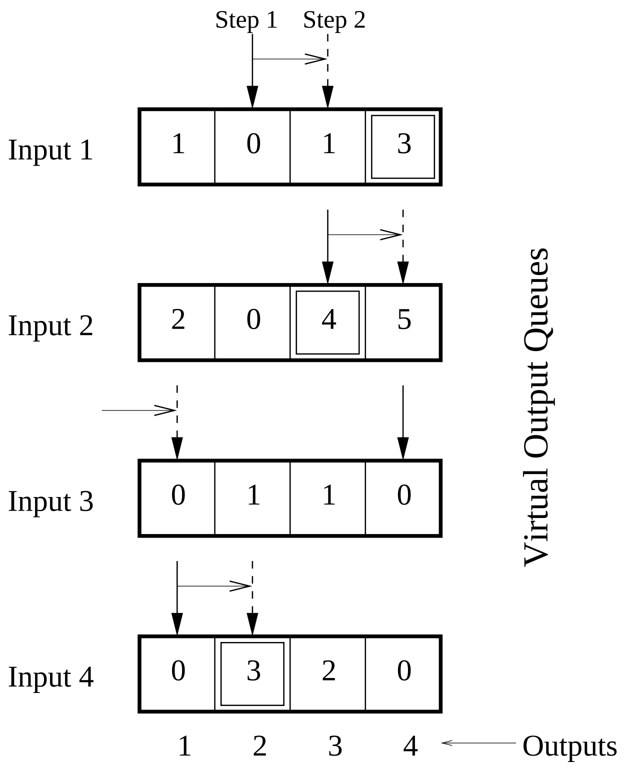


Figure 3.9: EREW Maximal Matching: A distributed implementation

prohibitive for a centralized arbiter of a high-speed switch. Two approaches are prevalent in the literature to address this issue, namely, a parallel or distributed maximal implementation, and the usage of larger data units and/or timeslots to switch.

**Parallel and Distributed Implementation**

One of the earliest approaches (e.g., used in PIM [1], iSLIP [78], LOOFA [65]) to generate a maximal matching is the so-called *request-grant-accept* (RGA) technique. RGA may be used for a parallel implementation in a centralized arbiter, or more beneficially, for a matching that is computed by the input and output elements in a distributed fashion. In general, each step of an RGA matching proceeds in three phases: (a) all the inputs, in parallel, generate a *request* to be connected to one or more outputs, based on the VOQ occupancy and other criteria; (b) from the set of inputs making a request, each output, in parallel, *grants* permission to a single input; and

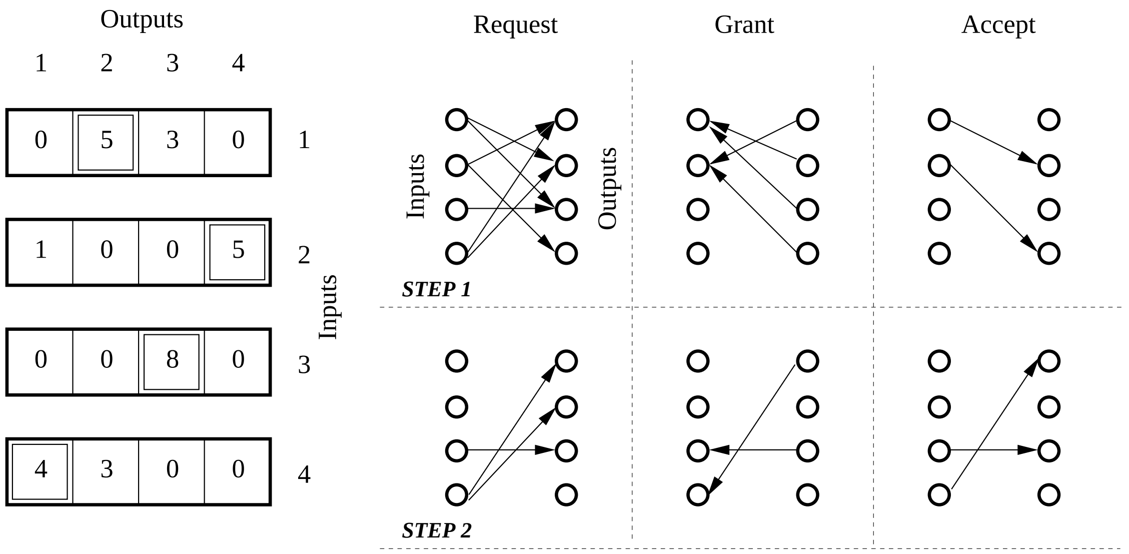


Figure 3.10: Parallel Iterative Matching: Example of an RGA algorithm

(c) if an input receives several grants, it *accepts* one of them and uses that as the match. The steps are iterated till a maximal matching is generated.

The EREW maximal matching described in Sec. 3.2.2, and shown in Fig. 3.9, lends itself to a straightforward RGA implementation. Each unmatched input requests a connection to a single output in each step, if it has a backlogged cell for that output. The pointers that the inputs use to generate the requests are staggered so that an output receives a request from at most one input in one step. If unconnected, the output grants the request, and the corresponding input always accepts the grant. Clearly, each step runs in $O(1)$ time, and a maximal matching is guaranteed in $N$ steps, since all the $N^2$ pairs are inspected. This remains one of the simplest RGA techniques to generate a general maximal matching. A matching may be deemed sub-maximal if the procedure stops after $k < N$ steps. The dispatch algorithms of the Atlanta [19] chip-set may be viewed as single-step ($k = 1$) sub-maximal variations. The advantage of a single step here is that no communication in necessary between the inputs and outputs, yet simulations show that the algorithm sustains *uniform* admissible traffic without speedup.

Parallel iterative matching (PIM) [1], iSLIP [78], and dual round-robin [11] are instances of iterative RGA matchings that have found commercial applications. In all these algorithms, each step (at the inputs and outputs, in parallel) runs in $O(N)$ time, however, the goal is to quickly converge to a maximal matching without requiring $N$ steps. The reasoning is that the communication overhead associated with a step dominates the required run-time, and hence should be minimized. In the first phase of PIM, each unmatched input sends a request to all the outputs for which it has a backlogged cell. An output makes a grant by choosing one requesting input at random. Similarly, an input randomly breaks ties to accept a connection from a granting output. Fig. 3.10 illustrates this technique, with two steps of the three phases. In iSLIP, the grants and accepts are performed using round-robin pointers at the output and input, respectively. Special care is taken, much as in EREW maximal, to ensure that the pointers at the different inputs (outputs) do not start at the same output (input), in each step.

In the worst-case, both the above algorithms need $N$ steps to converge, for a total run-time complexity of $O(N^2)$, which is extremely wasteful for a parallel algorithm. However, it was shown that, on average, it takes only $\log N$ steps to arrive at a maximal configuration. Furthermore, simulation results suggested that iSLIP with a *single* step and no speedup provides 100% throughput [78] for Bernoulli arrivals that are uniformly distributed among the outputs. This was attributed to the desynchronization of the pointers used at the outputs (inputs) to generate a grant (an accept). We show analytically, in a later section, that *less* is needed for *any* kind of uniform traffic. These happen to be so benign that a sub-maximal matching generated by a single step of the $O(1)$ EREW maximal technique suffices. For non-uniform admissible traffic, all such algorithms, with $N$ steps, can be expected to sustain 100% throughput with a speedup $s = 2$, in accordance with the Dai-Prabhakar maximal matching theorem.

**Reducing Configuration Frequency**

Reducing the frequency of space element configuration is desirable in practice due to two reasons. First, it allows the implementation of more complex matchings, and second, it provides the means to mitigate the overhead associated with each re-configuration, which happens to be large in optical space elements.

Recall that a matching needs to be completed once every internal timeslot, whose length equals $sC/L$, where $C$ is the external link capacity and $L$ is the length of the internal transfer

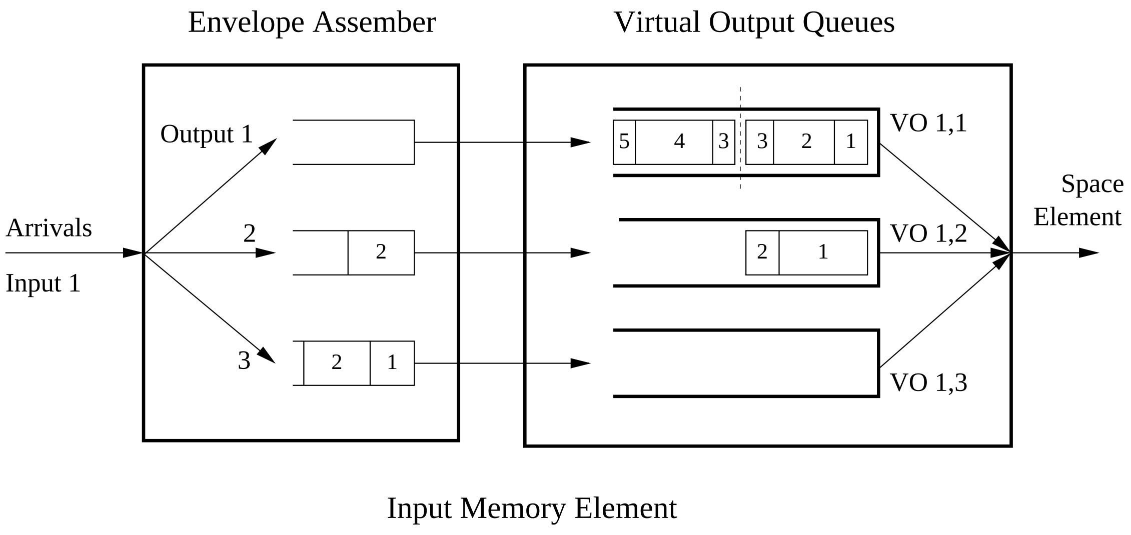


Figure 3.11: Assembling large envelopes for low-frequency matchings

unit (cell). Arbitrarily complex matchings can be implemented in each slot provided $L$ can be proportionately increased. Kar et al. [54] suggest *envelope matching* as a means to amortize matching complexity over a large internal timeslot. The SAR component in the port processors, responsible for chopping (re-assembling) packets into (from) fixed-size cells, is replaced by its dual. At the inputs, several packets are assembled into a large fixed-size envelope, as shown in Fig. 3.11. The arriving packets for each input-output pair are shown in the figure as numbered, using the order of arrival. Packets wait in this assembler until sufficient bytes are accumulated to fill an envelope. It is shown in [54] that the system remains stable without the need for additional speedup (similar in intuition to the packet-based matching results in Sec. 3.2.4). A good side-effect of this approach is that it handles variable-sized packets without the bandwidth overhead of the SAR, caused by fragmentation. However, the authors also realize that waiting for an envelope to fill adds to the packet delay. In some pathological cases, a partially filled envelope may remain unserved forever if no additional cells arrive for the corresponding flow. A heuristic is suggested to counter this, in which each matching is allowed to consider partial envelopes, in a separate phase (similar to renewal points in Sec. 3.2.4), after it has accounted for all the full envelopes.

In some optical space elements, even if the central arbiter computes matchings at a frequency of $sC/L$, it becomes infeasible to use those matchings at that frequency, as the re-configuration of the element is associated with a significant overhead. Towles and Dally propose [111] a combination of batch scheduling and efficient matrix decomposition to reduce the number of re-configurations. Specifically, the arbiter continues to compute matchings at a high frequency. The matchings are batched at epochs of period $T$, and then decomposed into $k \ll T$ permutations, each associated with the number of slots $N_k$ for which it needs to be held. The $k$ permutations are applied to the space element in the next batch interval. The MIN and DOUBLE algorithms, described in Sec. 3.2.3, are suggested to generate those permutations since $k$ only depends on $N$ in those schemes, and is independent of $T$. Li and Hamdi [73] have since proposed a method to measure the deterioration of average delay, which is one penalty in such batch scheduling schemes. Note that the additional speedup required for efficient decomposition is not a limiting constraint for optical elements.

## 3.3 Throughput and QoS in Packet Switching

### 3.3.1 Integrated Matchings

We reviewed a host of matchings for input-queued switches, each with its own specific properties and benefits, several contingent on arrival assumptions to meet the desired performance goal. The decision chart in Fig. 3.12 puts it all together, for a selected set of matchings, all assuming admissible[6] traffic. The task of a switch designer is to choose one or more algorithms, given the specific requirements and assumptions. For example, consider an ATM switch design, where the only requirement is to ensure bandwidth guarantees for traffic shaped by the required-rate matrix $R$, i.e., in-profile traffic. All that is needed in this case is a BVN template-based algorithm. Any auxiliary matching for the out-of-profile traffic would suffice, as maximizing the throughput is not a goal. In general, however, a switch may need to be designed for QoS satisfaction, specified by $R$, *and* throughput maximization for offered rates $\lambda$, which may be unknown and independent of $R$. Short of using a stable matching to emulate an OQ switch, which can ensure both simultaneously, this is typically achieved by integrating two or more matchings at the same time.

[6]Strictly speaking, LOOFA does not assume admissible traffic.

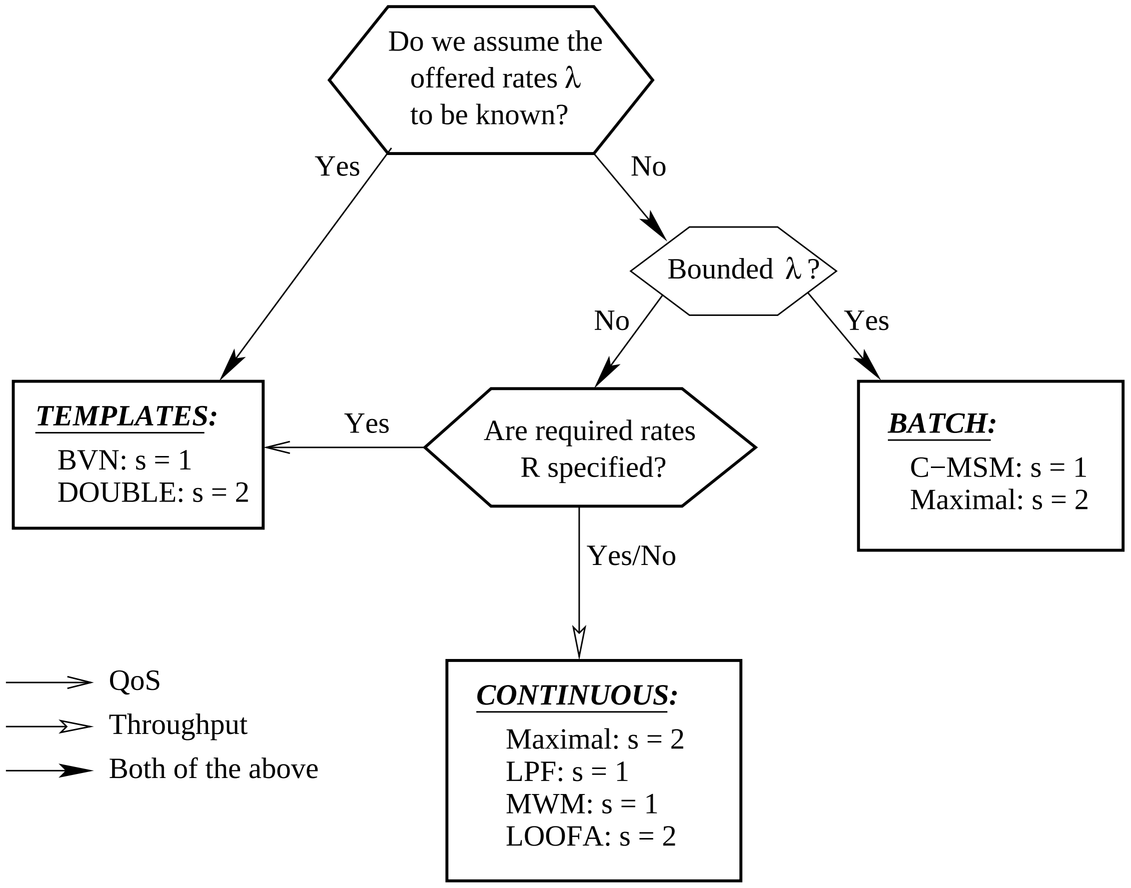


Figure 3.12: A decision chart to select matchings based on requirements

Let $M_1$ and $M_2$ be two policies chosen to satisfy two different performance criteria, specifically, QoS and throughput in this case, with respective speedup requirements of $s_1$ and $s_2$. We say that $M_2$ is *additive* with respect to $M_1$ if $M_2$ can add to the configuration generated by $M_1$ without losing its own properties. Note that this relationship need not be symmetric. If $M_2$ can add to $M_1$, these can be integrated into a single *multi-phase* matching $M$, with speedup

$$s_M = \max(s_1, s_2). \tag{3.11}$$

In each slot, $M_1$ is first applied to generate a partial configuration, and $M_2$, in a separate phase, augments the configuration. Let $T$ be an integer such that both $s_1T$ and $s_2T$ are also integers. If $s_1 > s_2$ (and vice-versa), then, out of every $s_1T$ internal timeslots, $M_2$ need not be applied for

$(s_1 - s_2)T$ timeslots on account of its lower required speedup. For example, a general maximal matching is additive with respect to any other matching algorithm. Therefore, we can use a multi-phase matching comprised of BVN templates for QoS, with $s = 1$, and maximal matching for throughput, with $s = 2$, in the second phase. The templates are applied only once in every two timeslots, and the overall required speedup remains 2.

We say that $M_1$ *conflicts* with $M_2$, if neither is additive with respect to the other. This is a symmetric relationship. In this case, the two matchings can be integrated only through an *exclusive* combination $M$, in which either one or the other algorithm is used by itself in each timeslot. The speedup required then is

$$s_M = s_1 + s_2. \tag{3.12}$$

Out of every $(s_1 + s_2)T$ internal slots, $M_1$ is applied for $s_1T$ slots, and $M_2$ for $s_2T$. As an example, an MWM, in general, conflicts with BVN template-based matching, as the latter does not consider queue lengths. Therefore, an integration of those two policies, for throughput and QoS, respectively, also results in an overall speedup requirement of 2, i.e., no better than the previous example. Clearly, the amenability of general maximal matchings to multi-phase integration is yet another of its many desirable properties.

Such integrations have not yet been studied extensively, though most switching systems use them in some form or the other. Recently, Chiussi et al. [15] proposed a distributed frame-definition algorithm (DFDA), which is an integration of rate-based templates and a single step of a maximal matching algorithm. Using simulations, the throughput is shown to be 100% for a class of admissible Bernoulli traffic, without speedup. This is likely because the templates require $s = 1$, and the additive single-step maximal matching also requires only $s = 1$ for such traffic.

### 3.3.2 Hierarchical Switch Scheduling

All the matching algorithms for input-queued switches, except the ones that exactly emulate an OQ switch, distribute bandwidth to coarse-grain input-output flows rather than to individual fine-grain ones that comprise the pairs. The QoS-enabling policies guarantee a trunk of $R_{i,j}$ to each input-output pair $(i, j)$, while the augmenting throughput-maximizing policies realize a portion

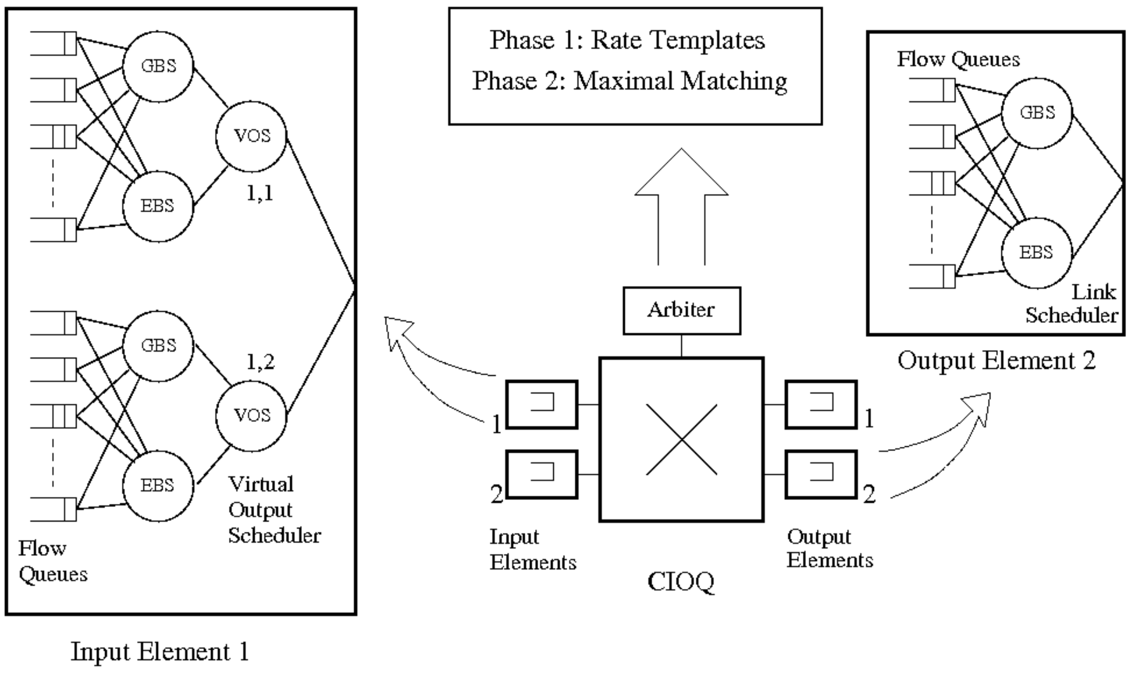


Figure 3.13: Throughput and QoS for input-queued switches using distributed scheduling

$\Phi^*_{i,j}$ ($R_{i,j} \leq \Phi^*_{i,j} \leq C$) of the output link capacity $C$ to each pair. $\Phi^*_{i,j}$ represents the long-term fairness in the allocation of output bandwidth. If the rates are admissible, then a policy that ensures 100% throughput essentially yields $\Phi^*_{i,j} = \lambda_{i,j}$ for all the pairs, and long-term fairness is moot. As it turns out, it suffices for the arbiter to consider only input-output flows, and fine-grain flows within them can be handled adequately using hierarchical link scheduling contained within the memory elements [103, 13, 14, 35].

The general approach is illustrated in Fig. 3.13. The output memory element contains a scheduling structure that is identical to the one in a reference OQ switch. Each output link $j$ is served by a combination of a guaranteed-bandwidth scheduler (GBS), ensuring a rate $R^{(k)}_{i,j}$ to flow $k$ in pair $(i, j)$, and an excess-bandwidth scheduler (EBS), which, together with GBS, determines the total service $\Phi^{(k)}_{i,j}$ observed by the flow. For example, the EBS could use a simple weighted allocation of the slots that are not used by GBS. Such a combination is referred to as a *fair-airport* [40] link scheduler. Let $\Phi_{i,j} = \sum_{k\in f(i,j)} \Phi^{(k)}_{i,j}$ be the total service observed by the pair $(i, j)$, where $f(i, j)$ is the set of flows for that pair. Let $\bar{\Phi}_{i,j}$ be the corresponding service observed by the flow in the above-mentioned reference OQ switch. These two quantities need not be equal, as $\Phi_{i,j}$ is dependent on the actions of the space element.

The space element employs a rate-based policy in its first phase that ensures a trunk of $R_{i,j} = \sum_{k \in f(i,j)} R_{i,j}^{(k)}$ to pair $(i, j)$. In other words, the trunk rate is exactly equal to the rate guaranteed by the output scheduler to the aggregate of flows in the pair. A second phase augments the above, resulting in a total observed rate of $\Phi_{i,j}^{*}$ for pair $(i, j)$. Clearly, $\Phi_{i,j} \leq \Phi_{i,j}^{*}$. Notice that $\Phi_{i,j} = \bar{\Phi}_{i,j}$ requires the space element to at least ensure $\Phi_{i,j}^{*} \geq \bar{\Phi}_{i,j}$. Then, we may say that the space element accords the same long-term fairness to the coarse-grain flows as an OQ switch. If the offered rates are admissible, then any integrated matching that provides 100% throughput is also long-term fair since $\Phi_{i,j} = \Phi_{i,j}^{*} = \lambda_{i,j} = \bar{\Phi}_{i,j}$. However, it is not straightforward to ensure the above equality for inadmissible traffic because the sum is not known beforehand, rather, it just results from the continued operation of the output link scheduler.

The service allocated by the space element to coarse-grain pairs can be translated back to per-flow service by employing a virtual output scheduler (VOS) for each output, as opposed to a FIFO VOQ, in the input memory element. Each VOS is also implemented as a fair-airport scheduler. The configurations generated in the first phase of the space-element matching algorithm are consumed by the GBS component. Since the rates required by the flows, within each pair, add up to the service allocated in this phase, each flow ends up receiving its per-flow reservation $R_{i,j}^{(k)}$. The penalty is in delay, which may be computed using hierarchical scheduling theory [4] as:

$$D_{i,j}^{(k)} \leq \; d_{i,j}^{(k)}(\text{GBS}) + \theta_{i,j}, \tag{3.13}$$

where $\theta_{i,j}$ is the *service burstiness* of the first-phase matching, which is defined as the maximum inter-service time observed by the GBS scheduler for pair $(i, j)$, and $d_{i,j}^{(k)}$ is the delay that would be experienced by the flow had it been served by the same GBS scheduler in a continuous fashion at capacity $R_{i,j}$.

The configurations generated in the second phase are consumed by the EBS component within the VOS. If the matching is long-term fair, each flow can indeed receive the same share of the output link capacity $\bar{\Phi}_{i,j}^{(k)}$ as in the reference OQ switch. For example, if the latter employs a weight-based EBS, identical shares can be achieved by using the same normalized weights for each flow in the EBS component. Note, however, that the system cannot enforce such fairness on a short timescale, as can be done in the reference switch. That continues to be dominated by the matching algorithm of the space element.

### 3.3.3 Memory Element QoS

The final task in providing per-flow QoS and throughput is to determine the specific policies to be used for the VOS in the input element, and the link scheduler in the output element. This choice may be drawn from the literature in OQ switching, independent of the matching algorithm. In other words, provided the above hierarchical scheduling framework is used, the memory elements may be considered as independent OQ switches for per-flow scheduling purposes. The input elements behave like a $1 \times N$ OQ switch, whose output links are really the respective VOS with capacity $\Phi^*_{i,j}$, and output elements behave like a $1 \times 1$ OQ switch, in which the input link capacity is $s$ times that of the output. Therefore, for completeness, we briefly review the OQ switch scheduling basics below.

#### Link Scheduling

Consider an $N \times N$ OQ switch with link capacity[7] $C$. If per-flow allocations are unimportant, a simple FIFO policy may be used at each output. In general, however, we assume per-flow control, where the flows may be defined at a desired granularity. The goal is to allocate $C$ among the flows so as to provide isolated rate guarantees for each flow, delay bounds for well-behaved traffic, and/or fairness in the overall allocation. A simple weight-based scheme is the Weighted Round Robin [56] (WRR) scheduler, in which each flow $i$ is associated with weight $w_i$. The flow is guaranteed a bandwidth of $(w_i / \sum_i w_i)C$. The worst-case latency between consecutive service may approach the size of the WRR frame. Note that bandwidth guarantees as well as allocation fairness are linked to the same weight. A popular credit-based implementation, which accounts for variable sized-packets, is the Deficit Round Robin [99] policy.

A classical rate-based scheme, which provides a framework for perfect flow isolation, is the fluid-model Generalized Processor Sharing (GPS) [87] policy. Each flow $i$ is associated with

[7]The contents of this entire section can be easily extended to account for a more general $N \times M$ OQ switch, with varying capacities on each output link. We are adopting a square switch with uniform output capacity merely for simplicity.

a required rate $r_i$, and receives an instantaneous service rate proportional to $r_i$, whenever it is backlogged. As long as the required rates are admissible, the service given to each flow meets or exceeds its requirement. If two flows $i$ and $j$ are continuously backlogged in interval $(t_1, t_2)$, the amount of service received by the flows are related as:

$$W_i(t_1, t_2)/r_i = W_j(t_1, t_2)/r_j$$

The GPS scheduler has several interesting properties. Firstly, as seen from the above equation, the rate-normalized service received by each backlogged flow is equal at every instant. Therefore, the available bandwidth in excess of the rate requirements is also distributed in proportion to each flow's required rate. This led to one of the first formal measures for *service fairness*, defined as the worst-case difference in normalized service between any two backlogged flows. Secondly, if $B(t)$ is the set of non-empty queues at instant $t$, the instantaneous service given to a queue $i$ is equal to $r_i C/\sum_{j \in B(t)} r_j$. Each flow receives such isolated service with zero latency. Thirdly, the observed delay depends only upon the flow's own arrival process, as shown in Fig. 3.14. For example, if the flow is token-bucket constrained [28] with a bucket size of $\sigma_i$, then the delay is bounded above by $\sigma_i/r_i$.

The packetized version of GPS, called Weighted Fair Queueing (WFQ) [30], works by essentially simulating a fluid system. This is achieved by keeping track of the normalized work done in the reference GPS scheduler, also called *virtual time* $v(t)$, at every timeslot. Note that, in GPS, $v'(t) = C/\sum_{i \in B(t)} r_i$. On arrival, the $k$th packet of flow $i$ with length $l_i^k$ is time-stamped using its finishing virtual time $F_i^k$, which is computed as:

$$F_i^k = l_i^k/r_i + \max(v(t), F_i^{k-1})$$

This policy simulates GPS with a worst-case added latency of $(l_i^k/r_i + l_{\max}/C)$. Several of the subsequent packet-based scheduling algorithms proposed the use of approximations for $v(t)$ that were easier to compute, but yielded different delay bounds and service fairness. For example, in Virtual Clock [117], $v(t)$ is set to real time $t$, which leads to delay bounds comparable to WFQ but unbounded service fairness. In Self-Clocked Fair Queueing (SCFQ) [39], the virtual time is set to the timestamp of the currently served packet, yielding the best fairness properties among GPS-related schedulers, but a delay bound which is $O(n)$, where $n$ is the number of

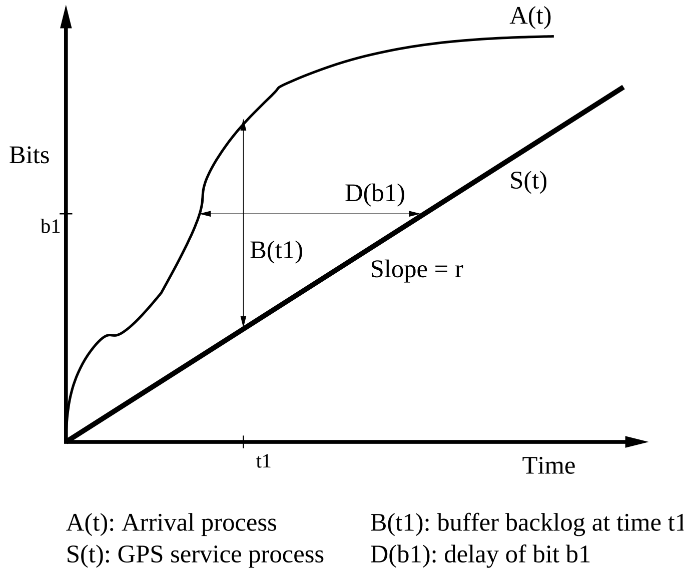
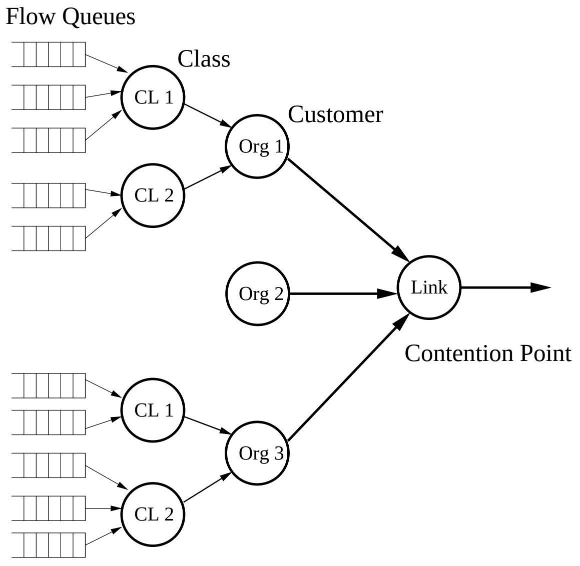


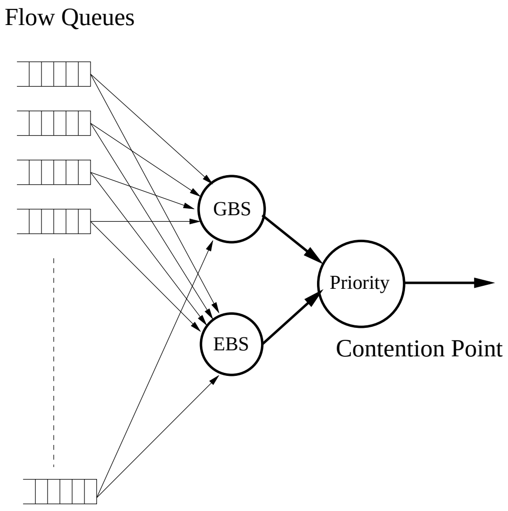
Figure 3.14: GPS: Delays and backlog for a single flow

flows. A general model for packet-based schedulers, called rate proportional [105] servers, was later devised, which showed that any scheme in which $v(t)$ grows at least as fast as real time, but always under-estimates it with respect to a GPS server, has the same added latency as WFQ, with the service fairness dependent only on the extent of under-estimation. Examples of such schemes include Virtual Clock and Frame-Based Fair Queueing (FFQ) [106].

The notion of *worst-case* fairness (which we use interchangeably with the term, service burstiness) was introduced in [3]. This is especially suited for hierarchical [4] (see Fig. 3.15(a)) and distributed schedulers, and is defined as the worst-case inter-service time of a scheduler. It was shown that the added latency, with respect to GPS, observed by a packet served by a hierarchical scheduler, is equal to the sum of the normalized worst-case fairness of the intermediate nodes. The inequality in (3.13) is essentially based on this result. In contrast to all the above schemes, in which fairness is tied to the corresponding required rate, the Fair Airport [40] algorithm allowed to decouple the scheduler into a non-work-conserving guaranteed bandwidth scheduler (GBS) (e.g., shaped Virtual Clock) to ensure rates and delays, and a work conserving excess bandwidth scheduler (EBS) (e.g., WRR), as shown in Fig. 3.15(b). Note that the results



(a) Hierarchical Scheduler



(b) Fair Airport Scheduler

Figure 3.15: Scheduler arrangements: (a) Hierarchical (b) Fair Airport

from hierarchical and fair airport scheduling were the ones that were used in the design of the VOS and output link schedulers in the memory elements of an input-queued switch in Sec. 3.3.2.

**Buffer Management**

Link scheduling policies and space-element matching algorithms focus on the finite bandwidth resource at contention points, with the goals of bandwidth isolation, fairness in service allocation and throughput maximization, essentially assuming an infinite-size attached memory. While we continue to make this assumption in the rest of this work, it is instructive to visit buffer management policies to keep the effects of finite memory in perspective. Given a finite buffer resource in a memory element, we may view buffer management as a mechanism to control the residue of the scheduler to support its *unhindered* operation. For example, if the scheduler is willing to allocate a long-term rate of $\Phi^{(k)}$ (not known beforehand) to flow $k$, the buffer manager must not inordinately drop packets belonging to that flow and bring the resultant service rate below $\Phi^{(k)}$.

The goal in buffer management is fairly well-defined for the guaranteed-QoS component of the offered traffic. Typically, a rate reservation $R^{(k)}$ is accompanied by a burstiness specification $\sigma^{(k)}$ (e.g., the token bucket size) during admission control. The task is to guarantee sufficient space so that the specified burstiness may be absorbed. Memory elements may be allowed to oversubscribe the buffer space if a small loss ratio is tolerated for the guaranteed component. Unfortunately, the goal is not as well-defined for the best-effort component. If the schedulers in the system offer long-term stability to flow $k$ ($\lambda^{(k)} = \Phi^{(k)}$, both unknown) assuming an infinite-sized memory, how should the buffer manager control admission into the finite memory in the short-term? Presumably, if the flow has non-conflicting access to a fixed share of the buffer space, and we assume a specific arrival process, techniques such as those based on Chernoff bounds [83] can be used to calculate the associated loss ratio. However, if two flows with guaranteed long-term stability compete for buffer space, how do their stability properties in the infinite-space virtual system translate to loss ratios in a finite-space system? More importantly, will an injudicious buffer management scheme hinder with the scheduling outcomes? There are some recent advancements (e.g., [96]) on this subject, but these questions are still in an exploratory stage of research.

In the absence of clear answers, the goal in maximizing throughput defaults to a combination of (i) allowing equal access to the available buffers; and (ii) sharing the buffers across flows to increase throughput when some of the flows are inactive. As a result, a flow with offered rate $\lambda^{(k)}$, and unspecified arrival profile, observes a bandwidth-buffer tuple of $(\Phi^{(k)}, B^{(k)})$, where the former is the long-term service rate and the latter is a time-varying resultant buffer space. The tail $(Q^{(k)}(t) > B^{(k)}(t))$ of the queue-length distribution will determine the observed loss ratio. The intuitive reasoning behind equal access is that, if all profiles are "similar", two stable flows, i.e., with $\lambda^{(k)} = \Phi^{(k)}$, will observe "similar" loss ratios, and the losses observed by unstable flows will be significantly "higher". Correspondingly, an unhindered scheduler may be assumed, which continues to consider the system as one with infinite-sized memory. It is not a coincidence that this redefined goal in buffer management is fairly well-addressed.

The classical *complete partitioning* [47] scheme may be used to generate a time-invariant $B^{(k)}$ that is perfectly isolated from other flows. This is a special case of *static thresholding* [34] in which the sum of the allocations is allowed to exceed the total buffer size. While the latter does not provide perfect isolation, it allows to share the buffers in a limited fashion by utilizing a knowledge of the inter-dependencies of arrival processes. The *dynamic thresholding* [23] scheme aims to equalize the queue lengths, while increasing throughput, by dynamically changing the allocation as a linear function of the free space. We have since proposed the *dynamic partitioning* [66] algorithm to simultaneously realize a static share to absorb the negotiated burstiness, and a dynamic share for the best-effort component.

All the above schemes fall into the category of arrival-drop policies in which the flow's buffer occupancy is checked against a threshold and the arriving cell dropped if the threshold is violated. In contrast, *push-out* policies [37] allow to maintain full buffer occupancy, and hence the lowest total cell loss, with exactly equal allocations to each backlogged flow in a congested buffer. When a cell arrives into a full buffer, a (different) cell from the longest queue is discarded. It was suggested in [108] that a cell from the head of the longest queue should be dropped to optimize TCP performance.

It should be noted here that many commercial switches do not queue packets on the basis of fine-grain flows. Instead, several flows are multiplexed into a common FIFO, and the link scheduler does not allocate bandwidth to each individual flow. In such cases, buffer management schemes have been adapted to control not just the residue of scheduling but also the per-flow bandwidth allocation itself. By regulating the entry of packets belonging to different flows into the same queue, such schemes essentially take up the role of a link scheduler. Popular examples in this category are Random Early Detection [33] and its multitude of variants, as well as some recent active queue management (AQM) schemes.

## 3.4 Summary

We reviewed several high-performance matching algorithms for input-queued (IQ and CIOQ) switches with a focus on providing QoS guarantees, specifically, bandwidth and delay, and optimizing throughput, independent of QoS. Note that most of the schemes in the literature address the latter only for admissible offered traffic. Special attention was paid to the formal methods from combinatorics and stochastic stability that were used to analyze and establish the properties of matching algorithms. We shall use similar methods in the analyses of our own contributions in later sections. We concluded the review of matching algorithms by outlining two schemes, envelope and batch matching, which can be used in conjunction with any specific algorithm to lower its amortized complexity. The presence of such add-on mechanisms is assumed in the rest of the dissertation without explicit elaboration.

We exposed a commonly used framework for integrating several matchings together, in order to simultaneously provide QoS and maximize throughput. We finally showed how a hierarchical scheduling structure may be combined with the rich results on link scheduling and buffer management, from OQ switching literature, to translate the coarse-grain properties of matchings to per-flow performance. Again, in the rest of the dissertation, we will assume the presence of these techniques (in memory elements) without specifically describing them. This allows us to concentrate on meeting the desired performance goals by merely focusing on the arrivals and departures on an input-output pair basis, and ignoring the effects on individual flows within them. For all practical purposes, such pairs are synonymous with flows in the rest of the work, unless otherwise specified.

We are now ready to extend the state of the art in two directions, namely, strengthening the performance results for input-queued switches so as to achieve different levels of equivalence with an ideal switch, and extending similar results to more general multi-module architectures.

# Chapter 4

# Buffered Clos Switches: A Framework

We introduce our taxonomy[1] of a class of three-stage multi-module switches in this chapter. The focus is on a constructive approach to build scalable switches using lower capacity logical elements, inspired partly by similar work in circuit switching that yielded the non-blocking Clos network. We call the resulting packet switches *buffered Clos switches* (BCS). This is accompanied by a description of feasible implementations of such three-stage switches, along with the design constraints that each resolves. We then introduce the performance framework of *functional equivalence* to accurately characterize the capabilities of the algorithms employed in these switches.

## 4.1 BCS Taxonomy

We defined two logical elements, in Sec. 2.4, that may be used to construct the forwarding path of a packet switch. These were defined in terms of their external behavior, rather than the internal implementations. Specifically, a memory element is a block in which an incoming packet is immediately presented to its output link, where it is queued and later scheduled for dispatch. In contrast, a space element is a memoryless interconnect that can realize a path for any given conflict-free configuration of input-output pairs (2.5). Referring again to the question we had asked ourselves as to what qualifies as the logical architecture of a multi-module switch, we

[1]The taxonomy of buffered Clos switches and the accompanying performance framework of functional equivalence were briefly introduced in a prior technical report [67].

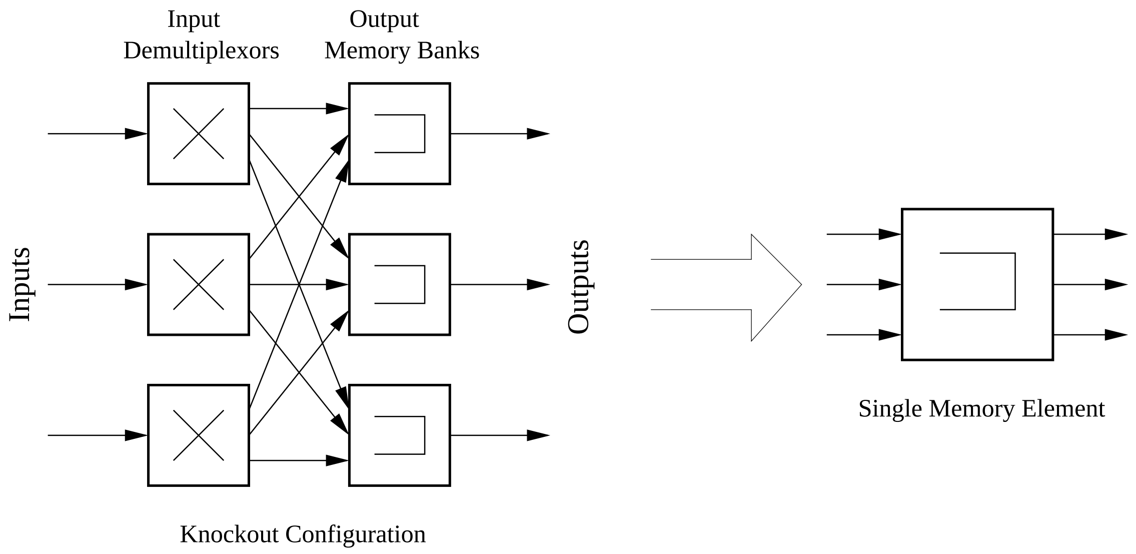


Figure 4.1: A Knockout configuration as a network of logical elements

suggest the following answer: any *irreducible* network of logical elements. This forms the basis of our proposed taxonomy.

**Definition 1.** *A network of logical switching elements is said to be* irreducible *if no subset of interconnected elements may be combined to form a single logical element.*

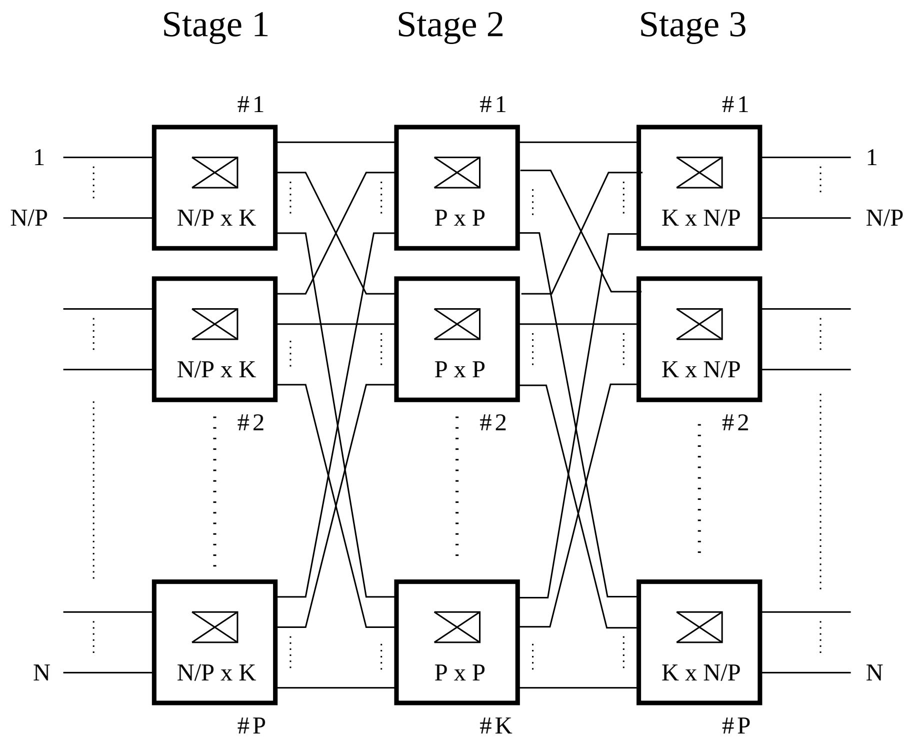
For example, consider the Knockout configuration shown in Fig. 2.4. This can be represented as a network of space and memory elements as shown in Fig. 4.1. If the interconnection between the first and second stages is not oversubscribed, i.e., a memory element can receive a packet simultaneously from all space elements, then the entire network can be trivially reduced to a single memory element, based on its external behavior. Similarly, a subset of interconnected space elements that form a non-blocking Clos network, as in Fig. 3.1, may be reduced to a single space element. An advantage of characterizing a switch in terms of its logical structure is that it allows us to isolate the *defining* algorithms of the switch, i.e., scheduling and arbitration, from implementation details. Given a switch, an inspection of the interconnection should yield its logical structure, which can then be meaningfully analyzed.

Let $S$ represent the set of elements that comprise a multi-module network. A network is said to be *multi-stage*, with $K$ stages, if $S$ can be partitioned into $K$ disjoint subsets, $A_k$, $k = 1, 2, \ldots, K$, such that the interconnections $I \subseteq \bigcup_{k=1}^{K-1} A_k \times A_{k+1}$. Note that this definition is similar to that of a layered network in graph theoretic terminology. When $I$ is exactly equal to the union of the above cartesian products, the $K$-stage network is referred to as *fully connected*. In other words, each element in $A_k$ is connected to every element in $A_{k+1}$. The input links of the elements in $A_1$ form the inputs to the overall system, while the output links of the elements in $A_K$ form the outputs of the system. The elements in $A_k$ are collectively referred to as the $k$th stage of the network.

**Definition 2.** *A stage $k$ of a multi-stage network is* uniform *if all the elements in $A_k$ are of the same type.*

For example, the Knockout configuration shown in Fig. 4.1 is a fully-connected two-stage network, where both stages are uniform. Furthermore, the network is said to contain *symmetric* stages if $|A_i| = |A_{K+1-i}|$, for all $i = 1, 2, \ldots, K$. Essentially, the overall network exhibits bilateral symmetry in terms of its interconnections. While our goal is to create a large switch as an irreducible network of memory and space elements, we will narrow our scope to fully-connected multi-stage networks with stages that are uniform and symmetric. A circuit-switching equivalent of such a network may be obtained by replacing all memory elements with space elements, essentially creating an isomorphic network of just space elements. Recall that a network is said to have a *non-blocking structure* if the circuit-switching equivalent is non-blocking. Guided by Proposition 2, which informally stated that a non-blocking structure is necessary for fitting admissible flows, we further restrict our scope to networks whose equivalents are non-blocking. Specifically, we consider the class of networks whose equivalent is the Clos network. While outside the scope of this work, we note here that such modeling can be extended to equivalents of other non-blocking circuit networks such as the Cantor network.

Given a requirement for a $N \times N$ switch with link capacities of $C$, the task of choosing a specific multi-stage configuration is driven by design and reuse constraints. Note that a monolithic OQ design would be limited by memory element constraints, namely, the memory bandwidth per output and the required full-mesh connectivity, and an IQ design by space element constraints, primarily, the arbitration frequency that limits the complexity of resident matching algorithms,



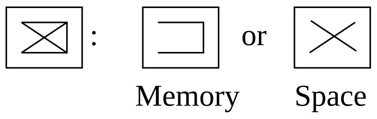


Figure 4.2: General model of a buffered Clos switch

which in turn typically depend on the switch size. Driven by the same limitations, a switch designer may use the largest such elements that can cost-effectively be implemented (or reused), and then create a multi-stage network using several instances of them.

### 4.1.1 Multi-module Architecture

The general model of a buffered Clos switch (BCS) is shown in Fig. 4.2. A BCS is obtained by augmenting selected stages of the circuit Clos network with memory elements. This class lends itself to a rich set of feasible multi-stage implementations, including CIOQ switches, sev-

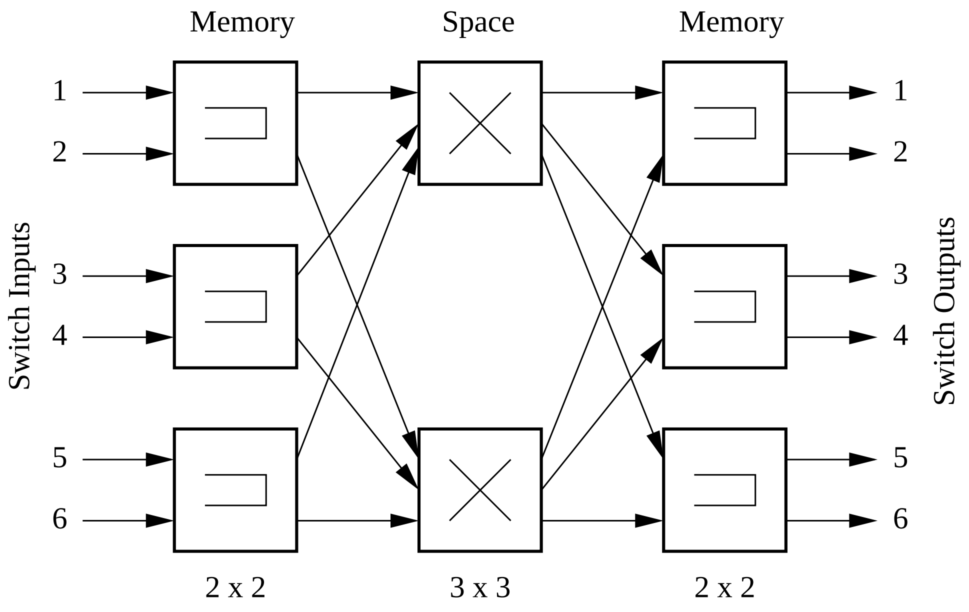


Figure 4.3: BCS example: $(6, [MSM], 3, 2, 2)$

eral variants of memory-space-memory switches, and load balanced switches such as PPS. More formally, we offer the following characterization.

**Definition 3.** *A buffered Clos switch is a fully-connected irreducible three-stage network of memory and space elements, with uniform and symmetric stages.*

For convenience, we will only consider multi-stage networks that realize *square* switches, i.e., those with an equal number of inputs and outputs. In practice, this is not a limiting assumption since the interfaces to most packet switches are full-duplex, each offering both an input and output link. Besides, all the architectures and analyses of square switches, presented in this work, can be easily extended to a general $N \times M$ model. The square, uniform and symmetric properties of the network allow us to represent a BCS using a compact 5-tuple notation:

$$\mathcal{S} : (N, X, P, K, s),$$

where $N$ is the number of inputs and $N \times N$ denotes the dimensions of the switch, $P$ is the number of first and third-stage elements, and $K$ is the number of second-stage elements. We assume that $N/P$ is an integer. Due to symmetry, the first stage elements have dimensions of

$N/P \times K$, the second $P \times P$, and the third $K \times N/P$. $X$ is a string that denotes the type of element in each stage, with $X[i] \in \{S, M\}, i = 1, 2, 3$. Here, $S$ denotes a space element, and $M$ a memory element. The parameter $s$ refers to the speedup between the first and second stages, and identically, between the third and second. The total input link bandwidth (nominal forwarding capacity) of the switch is $NC$. If the capacity of the links between the stages is $c$, the total nominal capacity between adjacent stages is equal to $cPK$. By definition,

$$c = \left(s\frac{N}{PK}\right)C \tag{4.1}$$

In other words, the given 5-tuple is sufficient to derive the normalized internal capacity $c/C$. For example, Fig. 4.3 shows the layout of $\mathcal{S} : (6, [MSM], 3, 2, 2)$. Here, $s = 2$ immediately implies that the internal links operate at twice the capacity of the external links. Note that $s = 1$ does not give a feasible implementation because the memory in the third stage becomes redundant. We used a similar configuration, as in this example, in the architecture of the Atlanta [19] and $\pi$ [21, 22] chipsets.

We define a BCS as *single-path* if each realizable path between any given input and output contains the same memory element(s). If two such paths contain different memory elements, the BCS is defined as *multi-path*. Note that we allow the paths between an input-output pair of a single-path switch to consist of different space elements. We propose such a definition, as opposed to a conventional one[2], because the algorithms in a single-path switch defined in this manner end up being markedly different from those in a multi-path switch. Specifically, as shall be seen in the subsequent chapters, the former is dominated by the matching algorithms in the space elements, and the latter by load-balancing scheduling algorithms. Furthermore, a cell experiences no difference in delay between two candidate paths that only differ in space elements. The BCS example in Fig. 4.3 is a single-path switch because each path between the input-output pair $(i, j)$ traverses through memory element $\lceil \frac{i}{N/P} \rceil$ in the first stage, and element $\lceil \frac{j}{N/P} \rceil$ in the third. The paths may traverse any of the central space elements.

The space elements $A_k$ in stage $k$ of a BCS are configured either by a co-operating single arbiter, or preferably, by $|A_k|$ separate arbiters. If the matchings generated by stage $k$ cause a contention at the inputs of that stage, then $k \neq 1$ and the previous stage has to necessarily

[2]Conventional refers to a graph-theoretic definition.

| $X$ | **Feasibility** | **Example** |
|---|---|---|
| $[SSS]$ | No, reduces to $[S]$ | Circuit Clos network |
| $[SSM]$ | No, reduces to $[SM]$ to $[M]$ | OQ switch: Knockout implementation |
| $[SMS]$ | Yes, if no contention at inputs | Birkhoff Von-Neumann switch |
| $[SMM]$ | Yes, if no contention at inputs | Parallel packet switch variant |
| $[MSS]$ | No, reduces to $[MS]$ | Simple IQ switch |
| $[MSM]$ | Yes | CIOQ switch |
| $[MMS]$ | Yes | None, so far |
| $[MMM]$ | Yes | Cluster of OQ switches |

Table 4.1: Feasibility of BCS implementations

consist of memory elements. The memory elements in stage $k$ contain scheduling algorithms, like the ones described in Sec. 3.3, with the requisite queueing structure. To prevent head-of-line blocking, there are at least as many virtual output queues as the number of outputs in stage $k+1$. Unless otherwise mentioned, we assume flows of the coarsest granularity, i.e., traffic belonging to the $N^2$ input-output pairs.

### 4.1.2 Feasible Implementations

The BCS taxonomy is obtained by varying $X$ and the relationship between $N$, $P$ and $K$. However, not all the eight permutations of $X$ lead to feasible implementations. Note that two fully-connected adjacent stages of space elements is always non-blocking and hence immediately reduces to a single space element. Therefore, $X \notin \{[SSS], [SSM], [MSS]\}$. The other five permutations may lead to feasible switches, as shown in Table 4.1. Additionally, some values of speedup may not make sense in some configurations. For example, $s = 1$ with $X = [MSM]$ makes the memory in the third stage redundant.

The feasible items we propose for the BCS taxonomy are illustrated in Fig. 4.4. Note that CIOQ-A, CIOQ-P and G-MSM are names we suggest, while the rest of the names are already found in the literature. Expressed in terms of the 5-tuple notation, the items are characterized by the following list.

```
Packet Switches
 ├─ OQ: Output Queued
 ├─ IQ : Input Queued
 └─ Buffered Clos Switches
      ├─Single Path
      │    ├─ CIOQ: Combined Input Output Queued switch
      │    ├─ CIOQ−A: CIOQ with Aggregation
      │    ├─ CIOQ−P: CIOQ with Pipelining
      │    └─ G−MSM: General Memory−Space−Memory switch
      └─Multi−Path
           ├─ PPS: Parallel Packet Switch
           └─ BVN: Load−balanced Birkhoff Von−Neumann switch
```

Figure 4.4: The taxonomy of buffered Clos switches

| | | |
|---|---|---|
| OQ | : | $(N, [M], 1, \o, \o)$ |
| IQ | : | $(N, [MS], N, 1, 1)$ |
| CIOQ | : | $(N, [MSM], N, 1, s)$ |
| CIOQ-A | : | $(N, [MSM], P, 1, s)$ |
| CIOQ-P | : | $(N, [MSM], N, K, s)$ |
| G-MSM | : | $(N, [MSM], P, K, s)$ |
| PPS | : | $(N, [xMx], N, K, s)$, $x$ is variable |
| BVN | : | $(N, [SMS], 1, K, s)$ |

Strictly speaking, IQ and OQ switches are not members of this class, but we use the same notation for uniformity. In subsequent analysis, we will continue to consider IQ as a special case of CIOQ, with $s = 1$. Furthermore, for the named items above, henceforth, we will use the names as short-form for the 5-tuple notations, as long as any constraints on $N$, $P$, $K$ and $s$ are clearly defined.

The CIOQ switch essentially forms the starting point of the taxonomy. As seen from the previous chapter, CIOQ is fairly well-studied and widely deployed. CIOQ-A, CIOQ-P and G-MSM switches are already found in practice (e.g., [19, 21]), but their performance is relatively less established. PPS and BVN switches are still mainly under theoretical investigation. We will briefly overview these architectures here. Specifically, we show how complex single-path designs can be obtained by applying aggregation and pipelining transformations to CIOQ switches, as shown in Fig. 4.5, and how path parallelization may be applied to generate a multi-path design.

**Aggregation**

A CIOQ switch with aggregation (CIOQ-A) is obtained by grouping $N/P$ consecutive interfaces of the former into the same memory element, where $N/P$ is an integer, and $1 < P < N$. More precisely, the aggregation transformation works as follows:

$$(N, [MSM], N, 1, s_1) \xrightarrow{a} (N, [MSM], P, 1, s) \tag{4.2}$$

A CIOQ-A switch is illustrated in Fig. 4.5(a). The value of $s$ may in general be different from $s_1$, and is determined by the performance of the resident matching algorithm. There are $P$ memory elements in the first and third stages, and a single space element in the second. From (4.1), the internal links operate at a capacity of $s\frac{N}{P}C$. Consequently, each memory element requires a bandwidth of $(1 + s)\frac{N}{P}C$. The space element requires an interface rate of $s\frac{N}{P}C$, and an arbitration frequency of $sN/P$ matchings per external timeslot. The last three values do not compare favorably with respective values for a CIOQ switch with the same speedup, namely, $(1 + s)C$, $sC$ and $s$, respectively. Therefore, the disadvantages of aggregation include a higher memory bandwidth and faster arbitration.

Nevertheless, the CIOQ-A offers three notable benefits. Firstly, it allows to decrease the dimensions of the space element, and with it the number of interconnecting links (enabling easier system design) and internal contention points. This is especially suitable for deploying optical space elements [115], which are available today at very high interface rates yet in small sizes. The problem of faster arbitration can be addressed by envelope matching as opposed to cell-based matching. Secondly, aggregation allows to re-use existing space elements to support multiple subports. In other words, the same space element can be used to interconnect memory elements with a varying number of subports, provided their rates sum up to $C$. Finally, the smaller size of the space element may be used to decrease the complexity of matching. For example, a

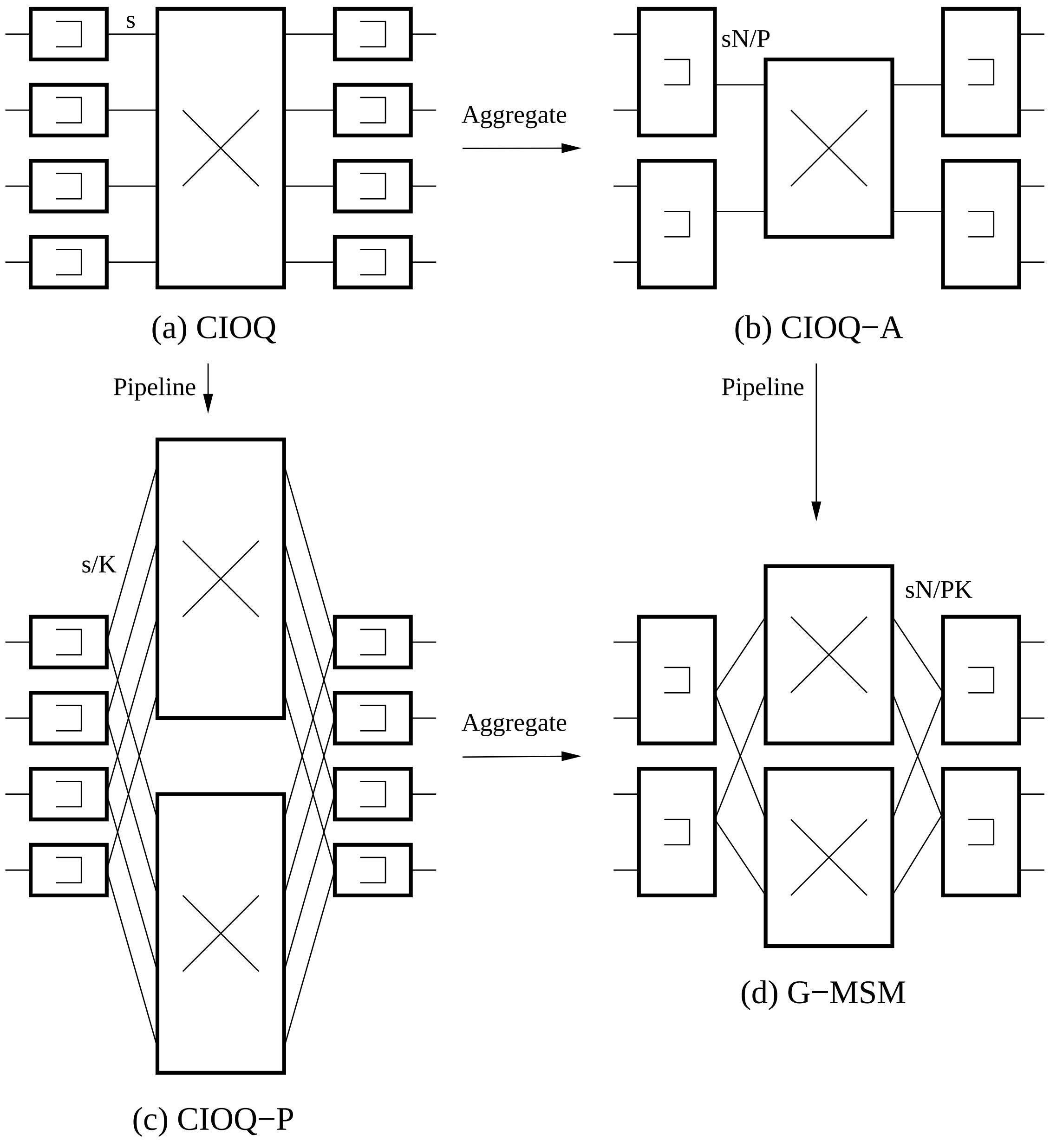


Figure 4.5: Single-path buffered Clos switches

sequential maximal matching on a CIOQ runs in $O(N^2)$ time, while the same algorithm runs in $O(P^2)$ time in a CIOQ-A. Accounting for the fact that $N/P$ matchings need to be computed for every one in a comparable CIOQ, the total run-time complexity of such matchings is $O(NP)$, still better than $O(N^2)$.

**Pipelining**

A CIOQ switch with pipelining (CIOQ-P) is obtained by replacing the single space element of CIOQ with $K$ instances of slower elements. This arrangement is very similar to the manner in which super-scalar pipelines are constructed in microprocessor and ASIC designs. More precisely, the pipelining transformation works as follows:

$$(N, [MSM], N, 1, s_1) \xrightarrow{p} (N, [MSM], N, K, s) \tag{4.3}$$

A CIOQ-P is illustrated in Fig. 4.5(b). Again, the value of $s$ may in general be different from $s_1$. The number of memory elements in the first and third stages remain equal to $N$, though their dimensions become $(1 \times K)$ and $(K \times 1)$, respectively. From (4.1), the internal links operate at a capacity of $sC/K$. Correspondingly, the memory bandwidth required in first and third stages remain comparable to a CIOQ. While the main design advantage of this approach is the usage of slower space elements, if the arbiters of the elements can be made to operate concurrently, the arbitration frequency goes down to $s/K$ matchings per external timeslot, thereby facilitating more complex policies. For example, the total run-time complexity of a sequential maximal matching goes down to $O(N^2/K)$. In fact, pipelining can be combined with aggregation to address the faster arbitration required for the latter. This leads us to general memory-space-memory (G-MSM) designs.

A G-MSM switch is obtained by simultaneously applying aggregation and pipelining transformations to a CIOQ switch, as shown in Fig. 4.5(d). Starting from (4.2) or (4.3), we generate a G-MSM as follows:

$$\begin{aligned} &(N, [MSM], P, 1, s_1) \xrightarrow{p} (N, [MSM], P, K, s) \quad \text{or} \\ &(N, [MSM], N, K, s_2) \xrightarrow{a} (N, [MSM], P, K, s) \end{aligned} \tag{4.4}$$

| | **CIOQ** | **CIOQ-A** | **CIOQ-P** | **G-MSM** |
|---|---|---|---|---|
| Number of elements in stage 1/3 | $N$ | $N/P$ | $N$ | $N/P$ |
| Number of elements in stage 2 | 1 | 1 | $K$ | $K$ |
| Memory Bandwidth | $(1+s)C$ | $(1+s)\frac{N}{P}C \downarrow$ | $(1+s)C$ | $(1+s)\frac{N}{P}C \downarrow$ |
| Matchings per external timeslot | $s$ | $sN/P \downarrow$ | $s/K \uparrow$ | $sN/PK \uparrow$ |
| Maximal matching complexity | $O(N^2)$ | $O(NP) \uparrow$ | $O(N^2/K) \uparrow$ | $O(NP/K) \uparrow$ |
| Native support for subports | No | Yes $\uparrow$ | No | Yes $\uparrow$ |

Table 4.2: Properties of single-path BCS designs

The internal links operate at a normalized capacity of $s\frac{N}{PK}$, from (4.1). There are $N/P$ memory elements in the first and third stages, each with a memory bandwidth of $(1+s)\frac{N}{P}C$, which remains the same as in CIOQ-A. However, the space elements now have lower interface rates of $s\frac{N}{PK}C$, and if they employ concurrent matchings, an arbitration frequency of $s\frac{N}{PK}$ per external timeslot. In addition, a G-MSM retains all the advantages of a CIOQ-A, namely, smaller space elements, support for multi-rate subports, and the possibility of less complex matchings. A sequential maximal matching, applied concurrently on the space elements, runs in $O(NP/K)$ time. Table 4.2 summarizes the properties of the various single-path BCS designs. The arrows indicate whether the respective property is an advantage or a disadvantage compared to a CIOQ switch.

The G-MSM architecture opens up an exciting possibility of building recursive BCS switches. A circuit Clos network (Fig. 3.1) can be recursively expanded [88] by converting each space element into a Clos network. In a similar fashion, we note that the memory elements of a G-MSM may be converted into CIOQ switches, leading to a five-stage (not fully-connected) multi-path design. Specifically, each $N \times K$ memory element may be replaced by a CIOQ switch of the same dimensions. This can be used to decrease the relatively high memory bandwidth of a G-MSM, which now approaches $(1+s\frac{N}{PK})C$, comparable to a stand-alone CIOQ switch. While we note this possibility, and expand it in a later section, we will not explore such five-stage switches in detail in this work.

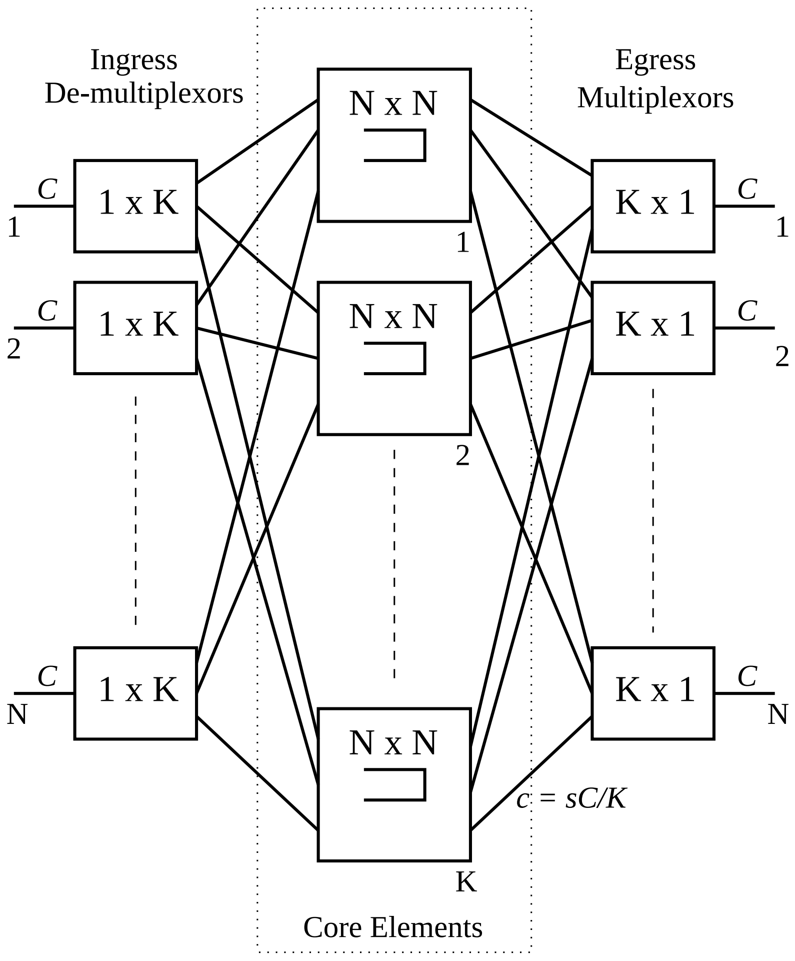


Figure 4.6: A parallel packet switch (PPS)

### Parallelization

A parallel packet switch (PPS) is a multi-path BCS that allows to pool the bandwidth resources on several switching paths, in order to construct a high capacity switch. As shown in Fig. 4.6, a $N \times N$ switch with interface rate $C$ is composed of $K$ parallel memory elements, each of size $N \times N$ and interface rate $c < C$. A central-stage element is referred to as a *core* memory element. The $1 \times K$ sized first-stage element is referred to as an ingress demultiplexor, and the $K \times 1$ third-stage element as an egress multiplexor. The $K$ logical core elements may be realized by a single physical $KN \times KN$ memory element yielding what we call an *inverse multiplexed* switch [17], or by $K$ separate physical planes [62, 49].

If $s$ is the internal speedup of the switch, then the internal links operate at a rate of $sC/K$, much as in a CIOQ-P switch. While memory elements in the first and/or third stages, each with memory bandwidth $(1+s)C$, lead to feasible PPS implementations, the main promise of a PPS lies in the possibility to make them both memoryless. If this can be achieved without a performance penalty, then the fastest memory in the entire system can be made to run *slower* than the external interfaces–a very attractive proposition given that memory capacities are not growing as fast as link capacities. Irrespective of this ability, a PPS at the very least allows for the reuse of lower capacity components in its core. The defining algorithms within the central memory elements continue to be the scheduling schemes typical of such elements. However, the performance of a PPS is dominated by the load-balancing policies at the demultiplexors, as opposed to matching algorithms in single-path designs. These policies distribute the offered flow traffic among the $K$ candidate paths available for each input-output pair.

With the basics of the BCS architecture in place, we now introduce our performance framework before embarking on a detailed analysis of each item in the taxonomy.

## 4.2 Functional Equivalence

We noted in Sec. 2.2 that the performance goals for a packet switch include successfully allocating admissible required rates using the given rate matrix $R$, and at the same time, optimizing throughput without the knowledge of the offered rates $\lambda$, which may or may not be admissible. For a switch with a non-blocking structure, any desired allocation may be given by ensuring them piece-wise in each component element. In a memory element this can be done using link scheduling policies, e.g., based on WFQ, and in a space element using deterministic properties of suitable matching algorithms, both of which were reviewed in the previous chapter. In addition, if the switch is also work-conserving, we noted that the resulting performance may be considered ideal.

Proposition 3 informally stated that queue stability may be used as a proxy for maximum throughput, and any switch that ensures stability in the same set of queues as a well-known ideal switch is also optimal in throughput. Short of establishing that a given design is ideal, such equivalence essentially allows for an asymptotic emulation of the ideal. Our performance framework formalizes this notion and provides a few meaningful levels of equivalence. We start with the ability to fit flows with known offered rates, and make it progressively stronger by first removing the necessity to know the rates, and then the constraints of admissibility. The goals of both rate allocation (QoS) and optimal throughput are subsumed within the levels of equivalence.

### 4.2.1 Levels of Emulation

We denote a switch $\mathcal{S}_1$ operating under a set of algorithms $\mathcal{A}_1$ to be functionally equivalent to a switch $\mathcal{S}_2$ operating under set $\mathcal{A}_2$ as follows:

$$(\mathcal{S}_1, \mathcal{A}_1) \overset{T,f}{\simeq} (\mathcal{S}_2, \mathcal{A}_2),$$

where $T$ is the class of arrival processes for which the equivalence holds, and $f$ denotes the level of equivalence. We propose the following levels:

$f_1$ Assuming all the average rates $\lambda_{i,j}$ between the input-output pairs $(i, j)$ are known and admissible, given an algorithm set $\mathcal{A}_2$ which ensures the stability of the queues in $\mathcal{S}_2$, we can find an algorithm set $\mathcal{A}_1$ which ensures the same in $\mathcal{S}_1$. In other words, $f_1$ denotes the ability to allocate known admissible rates, and therefore can be used to provide virtual trunks between input-output pairs.

$f_2$ Assuming the rates $\lambda_{i,j}$ are admissible (but not necessarily known), given an algorithm set $\mathcal{A}_2$ which ensures the stability of all the queues in $\mathcal{S}_2$, we can find a set $\mathcal{A}_1$ which ensures the same in $\mathcal{S}_1$ without explicit knowledge of the individual rates. We say that $\mathcal{S}_1$ is *relatively stable* with respect to $\mathcal{S}_2$ for *admissible traffic*.

$f_3$ Given an algorithm set $\mathcal{A}_2$ which ensures the stability of a subset of outputs $\{j\}$ for which $\sum_i \lambda_{i,j} < 1$, in switch $\mathcal{S}_2$, we can find a set $\mathcal{A}_1$ which enables the same in $\mathcal{S}_1$. We define an output $j$ to be stable if the system of queues which contain traffic destined to output $j$ is stable. In other words, $f_3$ represents the ability of a switch to isolate unstable outputs in the presence of inadmissible traffic, without knowledge of the offered rates. We say that $\mathcal{S}_1$ is relatively stable in the *wide sense* with respect to $\mathcal{S}_2$.

$f_4$ Given $\mathcal{A}_2$ which ensures the stability of a subset of input-output pairs in switch $\mathcal{S}_2$, we can find $\mathcal{A}_1$ which guarantees the same in $\mathcal{S}_1$ for the same subset. We define an input-output pair $(i, j)$ to be stable if the system of queues, which contain traffic belonging to the pair, is stable. In other words, $f_4$ represents the ability of the switch to isolate unstable input-output pairs within the same (unstable) output. This can immediately be seen as a measure of long-term fairness in the distribution of link bandwidth among the flows destined to that link. We say that $\mathcal{S}_1$ is relatively stable in the *strict sense* with respect to $\mathcal{S}_2$.

$f_5$ Given an algorithm set $\mathcal{A}_2$ for $\mathcal{S}_2$, we can find a set $\mathcal{A}_1$ which ensures that cells depart from $\mathcal{S}_1$ in an exact emulation of the departure from $\mathcal{S}_2$.

**Properties and Goals**

The methodology to prove the stability of a system of queues is left open, which might itself restrict the traffic class $T$. We establish two important properties of functional equivalence to aid in analysis. Notice that, by definition, each level of equivalence subsumes the previous one, i.e., if the equivalence holds at level $f_i$, then it holds for all $f_j, j < i$. This leads us to the *containment property*:

$$(\mathcal{S}_1, \mathcal{A}_1) \overset{T,f_i}{\simeq} (\mathcal{S}_2, \mathcal{A}_2) \Rightarrow \quad \forall j < i,\ (\mathcal{S}_1, \mathcal{A}_1) \overset{T,f_j}{\simeq} (\mathcal{S}_2, \mathcal{A}_2) \tag{4.5}$$

Also, it is easy to see that each level of equivalence lends itself to the *transitivity property*. That is, if a set of equivalences of switch $\mathcal{S}_2$ with respect to $\mathcal{S}_3$ is well known, it suffices to show the equivalence of a new switch $\mathcal{S}_1$ with $\mathcal{S}_2$, in order to establish the same equivalence with respect to $\mathcal{S}_3$.

$$\begin{aligned} \forall f \ \ & (\mathcal{S}_1, \mathcal{A}_1) \overset{T,f}{\simeq} (\mathcal{S}_2, \mathcal{A}_2) \\ \wedge \ \ & (\mathcal{S}_2, \mathcal{A}_2) \overset{T,f}{\simeq} (\mathcal{S}_3, \mathcal{A}_3) \ \ \Rightarrow (\mathcal{S}_1, \mathcal{A}_1) \overset{T,f}{\simeq} (\mathcal{S}_3, \mathcal{A}_3) \end{aligned} \tag{4.6}$$

Since the optimality of work-conserving OQ switches is well established, we can unambiguously characterize the performance of any given switch $\mathcal{S}$ by finding algorithms $\mathcal{A}$, such that it is functionally equivalent at level $f_i$ with the OQ switch, for the largest possible value of $i$ and the broadest traffic class $T$, i.e.[3],

$$\textbf{Goal}: \ (\mathcal{S}, \mathcal{A}) \overset{T,f_i}{\simeq} (\text{OQ, \{Work Conserving (WC)\}+ })$$

Note that we can deduce exactly which queues remain stable in a work-conserving OQ switch at each level. Specifically, if $\lambda$ is admissible, meaningful for levels $f_1$ and $f_2$, every queue remains stable. If $\lambda$ is partially admissible, meaningful for $f_3$, only those queues that contain traffic destined to outputs $j$ for which $\sum_i \lambda_{i,j} \geq C$ become unstable in the OQ switch. Finally, even within such outputs in the reference OQ switch, only those queues $(i, j)$ become unstable for which the offered traffic $\lambda_{i,j}$ is greater than the share of the output link bandwidth $\bar{\Phi}_{i,j}$ accorded to that pair. As is evident, the last is meaningful for level $f_4$. We chose to define optimal throughput not directly in terms of the stability of those queues but as a comparison between two switches, because the latter approach ends up being more flexible. More precisely, given a switch $\mathcal{S}$, it allows us to analyze throughput in comparison with another switch $\mathcal{S}_1$ with known properties, instead of always having to establish asymptotic emulation of the ideal switch.

### 4.2.2 Meaning of Equivalence

We now provide some intuition and examples to crystallize the meaning of equivalence levels, and see how existing measures in the literature compare with them. Recall that $f_1$ equivalence ensures bandwidth trunks of $\lambda$ to each input-output pair, provided those rates are known and admissible. However, a switch becomes QoS-capable only if trunks of required-rates $R$ can be guaranteed in short timescales, independent of $\lambda$. The length of the timescale determines the delay bound, while ensuring $R$ translates to guaranteed bandwidth for each flow. While $f_1$ by itself does not make a switch QoS-capable, it can be easily applied to QoS guarantees. The approach is to shape or mark incoming traffic so that the algorithms only consider a portion of $\lambda$ that confirms to $R$. This was shown in Fig. 3.6 and will be methodically demonstrated for a CIOQ switch in Chapter 5.

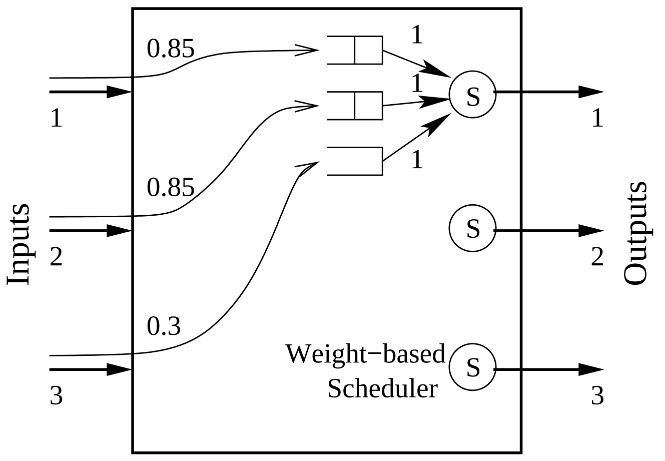
All the schemes in the literature that ensure 100% throughput for admissible arrivals essentially provide $f_2$ equivalence with a work-conserving OQ switch. As noted before, overall admissibility is assumed due to end-to-end flow control means such as TCP. Presumably, if the

[3] $B = A+ \Rightarrow A \subseteq B$

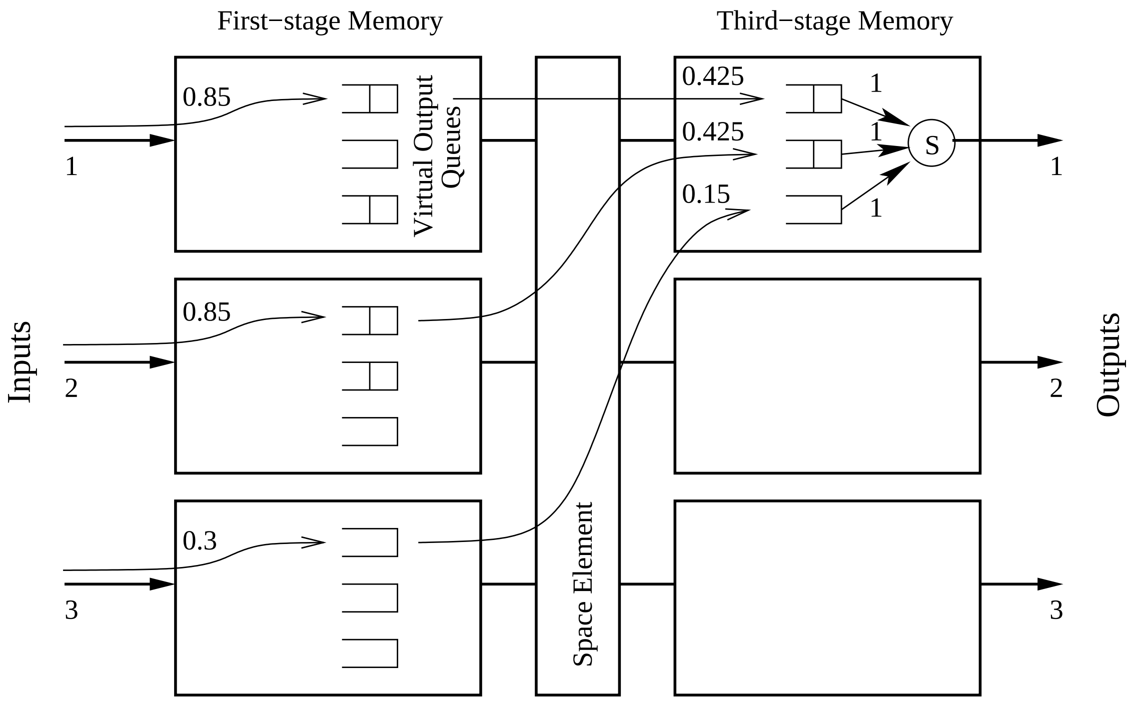
offered rates remain inadmissible over some timescale $T$, packets belonging to *some* flows will be dropped in the finite buffers, resulting in a throttling of the offered rates so that they become admissible. Our main criticism of such logic is that it is circular. The algorithms in the switch, which assume admissibility, effectively take an active part in determining which flows experience huge backlogs, and therefore need to scale back in order to ensure the same admissibility. Note that a work-conserving OQ switch needs no such circular reasoning for throughput optimization. If some output $j$ continues to receive traffic in excess of the link capacity, and remains that way over any timescale, it does not affect the queue state of other outputs.

The levels $f_3$ and $f_4$ essentially address the above shortcoming of the results in the literature that claim 100% throughput. This limitation, of course, has not been completely lost on the switching community. The qualitative notion of starvation prevention (e.g. [78]), i.e., guaranteeing that no input-output flow is starved of service, is commonly used as an add-on criterion to 100% throughput. For admissible arrivals, however, such a notion offers no measurable benefits to the long-term queue state. For inadmissible arrivals, the long-term benefits are unclear. In contrast, $f_3$ and $f_4$ equivalence allow to isolate instability, and make additional measures like starvation prevention unnecessary, at least for long-term throughput. The former isolates instability on an output basis, and the latter on a finer grain, namely, on an input-output pair basis. Correspondingly, the offered traffic of one flow will not affect the long-term throughput of other flows. Only those flows that persistently offer traffic in excess of what can be served by the output links will experience huge backlogs, much as in an ideal switch.

Note that any scheme that ensures work-conserving behavior on a per-output basis, e.g., LOOFA for CIOQ switches, immediately implies $f_3$ equivalence, though the converse is not necessarily true. In other words, WC is a stronger property than $f_3$. Accordingly, we will prefer non-work conserving algorithms that ensure $f_3$ only if their implementation is simpler. Clearly, a WC algorithm does not necessarily translate to $f_4$ equivalence with an ideal switch. A counter-example is shown in Fig. 4.7. In the reference switch, flows are served using a weight-based output link scheduler. If the weights given to flows (1,1), (2,1) and (3,1) are equal, the last flow remains stable. On the other hand, in a work-conserving CIOQ switch, the space element may ensure that output 1 never idles, but may give different allocations to each of the flows. In this



(a) Reference OQ Switch



(b) A work−conserving CIOQ switch

Figure 4.7: Example: Work-conservation does not lead to strict relative stability

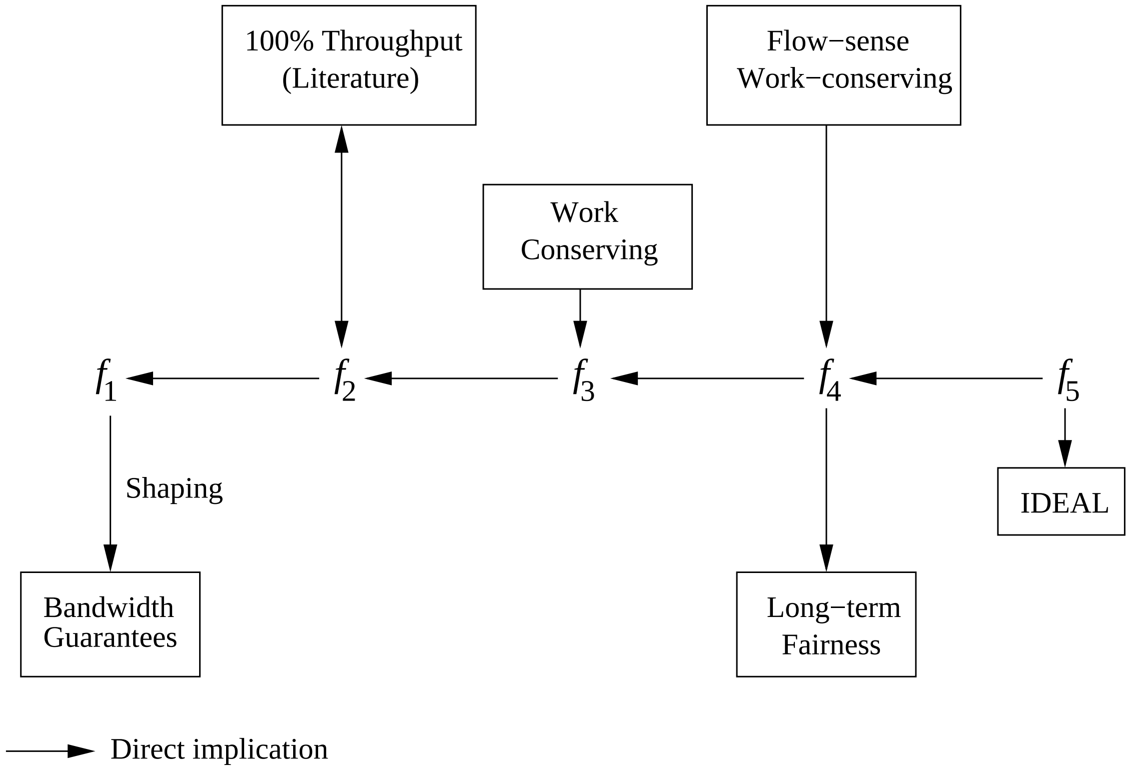


Figure 4.8: Relationship between various performance measures

case, all the virtual output queues can become unstable. Clearly, this situation arises because the space element does not take into account the shares given to each flow by the output link scheduler. The isolation of instability on an input-output pair basis, addressed by $f_4$, may be viewed as a form of long-term fairness in the allocation of the output link bandwidth. Atleast from the point of view of stable flows, the switch provides the same long-term share of the output link bandwidth as a reference switch with a fair output-link scheduler.

We define a system to be *flow-sense* work-conserving (FWC) if, for all $j$, the link scheduler of output $j$ has access to a cell belonging to each pair $(i, j)$, if such a cell exists anywhere in the system, in its scheduling decisions. Furthermore, if the scheduler chooses $(i, j)$ in a timeslot, a cell belonging to that pair must be dispatched on the output link. A WC scheme merely offers any cell belonging to an output to the respective link scheduler, if such a cell resides anywhere in the system, without regard to its originating input. Consequently, it might not offer a cell originating from a specific input that might have been better preferred by the scheduler. It is clear that an FWC scheme, on the other hand, directly implies $f_4$ equivalence with an ideal switch, though the converse is not necessarily true. Unfortunately, no such scheme exists even for CIOQ switches, short of exactly emulating an ideal switch using stable matching.

It might be tempting to provide $f_4$ equivalence using weight-based templates in space elements, using the same weights of the output link scheduler. However, that approach does not work because such templates are static and do not consider the offered traffic. For example, in Fig. 4.7, let there be a fourth input and let the output scheduler allocate the link bandwidth using equal weights to (1,1), (2,1), (3,1) and (4,1). Clearly, if flow (4,1) only offers a normalized rate of 0.1, flow (3,1) continues to remain stable in the reference switch. On the other hand, a weight-based template in the CIOQ would ensure a rate of 0.25 to each flow, and the unused portion of flow (4,1) may be arbitrarily distributed, leading possibly to (3,1) becoming unstable. It suffices to note here that the actual share of the link bandwidth given to each flow by an ideal switch cannot be surmised by the static parameters used by its scheduler. Fig. 4.8 summarizes the relationships between our levels of equivalence and the measures found in the related art.

We conclude this discussion by noting that equivalences $f_1$–$f_4$ suffer from the same limitations as any measure that relies on long-term stability in an infinite-queue system. In practice, of course, the traffic offered to a scheduler in a switch is always admissible, due to the actions of the finite-memory buffer manager. Do such equivalences at least ensure that the packet-loss ratio for a flow is a function only of its own long-term average rate and profile? Does the burstiness in service observed by a flow that is guaranteed to be long-term stable translate to higher losses, resulting in lower-then-optimal throughput? These are some of the issues, related to the interaction of scheduling and buffer management, and the timescale of offered service, that are not resolved in this work. For our purpose, we merely offer the following arguments in defense of our approach. To address the first question, we claim that if a queue is stable in the infinite-memory system, the observed loss ratio in a corresponding finite-memory switch will indeed depend only on its own profile, which determines the tail of its queue length distribution, at least for complete-partitioning buffer management schemes. To address the second, we claim that the burstiness in observed service might indeed lead to a heavier tail, and a higher loss ratio in a finite-memory system, but the ratio will not be materially affected by the other offered traffic. We realize that this issue is related to the timescale of service, and probably, some additional measure like ensuring that a backlogged queue does not starve (!) within a finite interval might be in order.

In summary, a stronger performance framework would indeed address the emulation of a finite-memory OQ switch with the goal of providing comparable loss ratios to each flow. Another alternative would be to provide for short-timescale OQ emulation as opposed to an asymptotic one. Nevertheless, our framework of functional equivalence addresses the isolation of flows that become congested in the reference switch, without the presumption of admissibility. The rationale is that if the algorithms in the switch do not presume admissibility, it is unlikely to affect flows that do not experience congestion in the reference.

### 4.2.3 Existing Results

We can now relate the specific results in input-queued switching, reviewed in the previous chapters, using the relationships shown in Fig. 4.8. The theorems on matching seen so far yield the following equivalences:

$$\text{(IQ, \{BVN Decomposition\})} \overset{T,f_1}{\simeq} \text{(OQ, \{Rate Scheduler\})}$$
$$\text{(CIOQ, \{DOUBLE\})} \overset{T,f_1}{\simeq} \text{(OQ, \{Rate Scheduler\})}, \quad s \geq 2$$
$$\text{(IQ, \{MWM\})} \overset{T,f_2}{\simeq} \text{(OQ, \{WC\})}, \quad T\text{: SLLN}$$
$$\text{(IQ, \{LPF\})} \overset{T,f_2}{\simeq} \text{(OQ, \{WC\})}, \quad T\text{: i.i.d.}$$
$$\text{(IQ, \{C-MSM\})} \overset{T,f_2}{\simeq} \text{(OQ, \{WC\})}, \quad T\text{: Bernoulli, uniform}$$
$$\text{(CIOQ, \{Maximal Matching\})} \overset{T,f_2}{\simeq} \text{(OQ, \{WC\})}, \quad s \geq 2, \ T\text{: SLLN}$$
$$\text{(CIOQ, \{LOOFA\})} \overset{T,f_3}{\simeq} \text{(OQ, \{WC\})}, \quad s \geq 2$$
$$\text{(CIOQ, \{LOOFA, OCF\})} \overset{T,f_5}{\simeq} \text{(OQ, \{WC, FIFO\})}, \quad s \geq 3$$
$$\text{(CIOQ, \{Simulate OQ, Stable Matching\})} \overset{T,f_5}{\simeq} \text{(OQ, \{WC\}+)}, \quad s \geq 2$$

Clearly, the performance of input-queued switches are well-represented in the literature, except for practical algorithms to provide $f_4$ equivalence with an OQ switch. Accordingly, the latest interest in such switches revolve around providing similar properties using less-complex matchings [52], and to characterize the quality of stability in terms of queue-length distributions [71, 98] and the effects of finite memory [96]. The other multi-module architectures presented here are less well-studied.

## 4.3 Summary

We demonstrated a constructive approach in this chapter to build three-stage switches using lower capacity memory and space elements. The buffered Clos switches (BCS) taxonomy was formally introduced, along with the architectures of several feasible implementations. We paid special attention to the design constraints that each item addressed, so that switch designers may use the largest building blocks that can cost-effectively be acquired, and interconnect them to realize a larger switch. While several items in the taxonomy are already in existence, such a formal model is unprecedented. The model may be extended in the future to build more sophisticated multi-module designs, such as a recursive BCS or a buffered Cantor network.

We provided a performance framework to compare and contrast the algorithms residing in such switches. Several levels of functional equivalence with an ideal switch, inspired by the similarly tiered framework of blocking in circuit switching, were proposed as means for emulating the ideal switch in QoS and throughput. We provided intuition and examples to see how existing measures relate to this approach, and to outline the shortcomings of some of them. Most of the levels of equivalence do rely on queue stability in an infinite-memory system, and admittedly, additional work would be needed to characterize their applications to finite-memory systems, specifically, to understand how buffer management would interact with switch scheduling and to analyze the service burstiness of long-term stable schemes. Nevertheless, such functional equivalence provides a good first step in isolating instability and providing optimal throughput to an input-output flow, independent of the offered traffic to other flows in the system.

The rest of the dissertation essentially deals with detailed analysis of the individual items in the BCS taxonomy. The goal is to design switching algorithms for those items in order to achieve equivalence with an ideal switch.

# Chapter 5

# Combined Input-Output Queueing

You see, you know how to take a reservation, you just don't know how to *hold* the reservation. That's really the most important part of the reservation, the holding. Anybody can just take them...

– *Jerry Seinfeld*, Episode 28, NBC, 1991

Input-queued switches (IQ and CIOQ), which form the starting point of the BCS taxonomy, continue to be the most popular designs for the forwarding path of a high-capacity packet switch, primarily because the memory bandwidth in such switches does not depend on the dimensions of the switch. The rich set of results in input-queueing literature have been leveraged in several practical implementations. Bandwidth and delay guarantees are provided in practice using some variation of the BVN decomposition technique [7, 111], and throughput by maximal matchings [29] as well as sub-maximal heuristics [1, 78, 11]. Due to the plethora of well-understood results, the switching community has, for the most part, deemed input-queueing as a closed topic. We report here that several of the well-known matching algorithms, or minor variations thereof, lend themselves to stronger performance results than is currently accepted. It seems that many of these results have eluded the community, despite the impressive research output, mainly because we have not been asking the correct questions.

Specifically, we report the following results in this chapter. First, we present a simple extension[1] of the Weller-Hajek maximal matching theorem [114] for bounded admissible traffic,

[1]Sec. 5.1, which presents this extension, is an outcome of several discussions with Dimitrios Stiliadis, Bell Labs,

and show that the same can be used to achieve bandwidth and delay guarantees in the presence of arbitrary traffic. Second, we show using the Dai-Prabhakar fluid-limit approach [29] that a critical matching suffices for 100% throughput to admissible arrivals, without speedup. Comparable results in the literature are based on more complex matchings like LPF [79] and MWM [80]. Next, we expose the fact that a simple sub-maximal matching ensures 100% throughput for any uniform admissible arrival pattern. This could (yet again) explain the similar simulation-based observations for popular algorithms such as iSLIP. Finally, we show that maximal matchings hold up well even in the presence of inadmissible traffic. More precisely, we present a specific maximal matching, called shortest output-queue first, which ensures strict relative stability with an OQ switch. To conclude, we put these QoS and throughput results together and present the switched fair-airport framework to enable their combined operation.

We contend that the above results constitute a significant theoretical advancement to the art of input-queued switching. In fact, the throughput results of critical and maximal matchings may be considered fundamental, and belong alongside the various theorems reviewed in Chapter 3. More beneficially, most of the material presented here are based on simple matchings, and can be considered for practical implementations.

## 5.1 Maximal Matching: Application to QoS

Recall that the task in providing bandwidth guarantees in a switch is to ensure sufficient service turns to each input-output pair so that the service rate meets or exceeds a reservation, specified by a rate matrix $R$, independent of the offered arrivals $\lambda$. The maximum waiting time for an input-output pair, or the latency of scheduling, together with the profile of a flow itself, end up determining the delay bound for that flow. Given an admissible $R$, matrix decomposition using Clos fitting [46] or the more efficient BVN [7] and DOUBLE [111] techniques can be readily used to generate a repeating sequence of matching templates to satisfy $R$. The latency in all such approaches, except for the GLJD algorithm [58], which uses a significant speedup, depends on $r$, the greatest common divisor (GCD) of the values in $R$ (see Sec. 3.2.3 for details).

The off-line template approach has two disadvantages. It imposes a high storage over-

and hence should be considered as joint work.

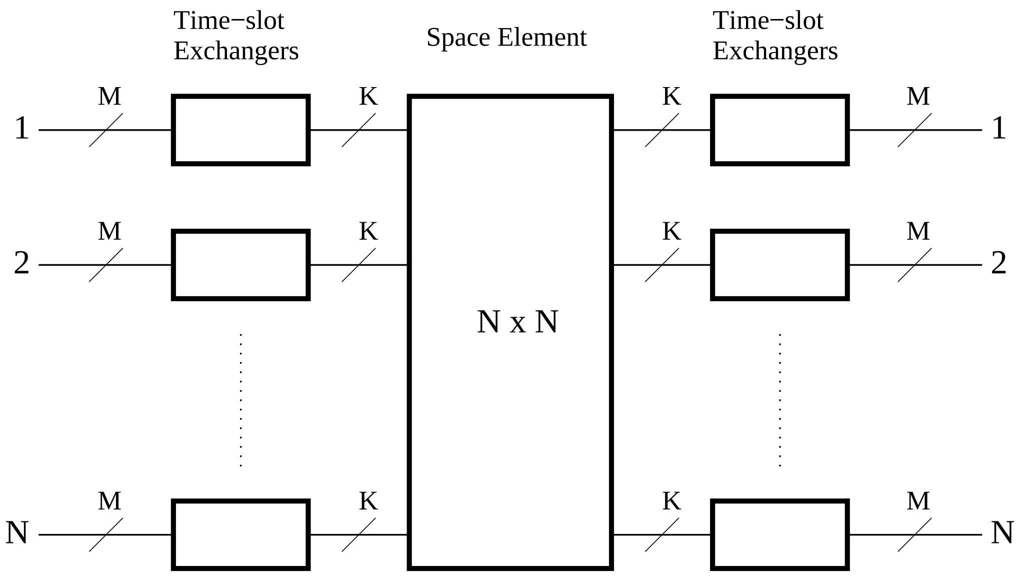


Figure 5.1: A circuit Clos network: Time version

head, as the templates need to be stored in high-speed memory attached to the central arbiter of the switch. Furthermore, the entire set of templates and their associated weights need to be re-computed (e.g., at a run-time complexity of $O(N^{4.5})$ for BVN) every time $R$ changes, on the addition or deletion of a fine-grain flow. A comparable online matching algorithm without the storage overhead and a distributed processing of rate changes would certainly be preferable. The Weller-Hajek result, which states that any batch-mode maximal matching with a speedup $s \geq 2$ is sufficient for exact 100% throughput to bounded admissible arrivals, provides us an online alternative. We show here that the constraint of bounded admissibility on $\lambda$ may be removed by explicitly shaping the arrivals to conform with the independent $R$. This may also be viewed as an application of the Dai-Prabhakar maximal matching result to finite timescales.

We emphasize that the results in this section are but an extension of the Weller-Hajek theorem, our additional contributions being the removal of its main constraint, delay bounds for independently regulated sources, and a new proof methodology that proves the often conjectured [1] analogy between CIOQ packet switches and circuit Clos networks. In some sense, these are also expected results, e.g., by following the implication arrows in Fig. 4.8 starting from $f_2$, since maximal matchings have already been proven to provide $f_2$ equivalence with an OQ switch. The closest similar results to the ones presented here are due to Smiljanic [100] who offers a specific maximal matching, based on rate-based credits instead of rate-shaping, to achieve the same goal. The Clos analogy is also argumentatively established in that work.

### 5.1.1 Clos Networks: Space-Time Duality

Consider a $NM \times NM$ circuit switch, implemented as a Clos network similar to the one shown in Fig. 3.1. Each link in the network has a capacity of one circuit. There are $N$ instances of $M \times K$ space elements in the first stage, $K$ instances of $N \times N$ elements in the second, and $N$ instances of $K \times M$ elements in the third.

A particularly relevant property of Clos networks is that the links, internal and external, may interchangeably be in space or time. Specifically, Fig. 5.1 shows a *time version* of such a network. The $NM \times NM$ switch with unit link capacities is replaced by a $N \times N$ switch with a capacity of $M$ circuits on the external links and $K$ on the internal ones. A *frame* of an external link consists of $M$ timeslots, each corresponding to an available circuit, and a frame of an internal link consists of $K$ timeslots, each available to fit exactly one circuit. A circuit request between an input-output pair $(i, j)$ of a time-version Clos network is *admissible* if input $i$ and output $j$ both contain idle, not necessarily coinciding, timeslots.

Let the $N \times N$ matrix $\pi^{(k)}$ represent the configuration of the central space element for internal timeslot $k$, $1 \leq k \leq K$. The space element mandates that $\pi^{(k)}$ is conflict-free (2.5). A request for an admissible circuit is satisfied by finding an idle slot $m$ on the given input link $i$, a similar slot $n$ on the output $j$, and an internal slot $k$ such that $\pi^{(k)}[i, j] = 0$. If such a slot exists, then the first-stage element $i$ merely exchanges the traffic unit arriving at slot $m$ to internal slot $k$, the central element switches the unit from $i$ to $j$ in timeslot $k$, and output $j$ exchanges the unit to the correct external slot $n$. The timeslot exchange is accomplished by accumulating[2] all the traffic units belonging to a single frame and rearranging them. The existence of an idle internal slot to fit an admissible circuit continues to be governed by the Clos theorem (pg. 31). Recognize the duality of the $M$ slots on the external links with $M$ separate unit-capacity links, and the $K$

[2]The timeslot exchange operation is sometimes referred to as *grooming*.

slots of the space element with $K$ physical unit-capacity elements. For the switch of the above dimensions, Clos theorem requires $K \geq 2M - 1$.

### Aggregate Circuit Requests

Consider a contrived circuit-arrival scenario, in which a number of circuit requests arrive at the same instant to an idle time-version Clos network. Let the $N \times N$ matrix $\Pi$ represent the aggregate of simultaneous circuit requests, such that $\Pi[i,j]$ equals the number of requested circuits between the input-output pair $(i,j)$. An aggregate request matrix $\Pi$ is admissible as long as *all* the circuits in $\Pi$ can be accommodated on the external links. That is,

$$\forall j \ \sum_i \Pi[i,j] \leq M \quad \text{and} \quad \forall i \ \sum_j \Pi[i,j] \leq M \tag{5.1}$$

Let $\pi^{(k)}$ be initialized to the zero matrix for all $k$. Consider a *sequential* Clos fitting algorithm, which operates as follows. We pick each circuit in $\Pi$, in an arbitrary order, and perform Clos fitting by finding the smallest timeslot $k$ for which $\pi^{(k)}[i,j] = 0$ and setting it to 1. Notice that irrespective of the order in which the circuits are picked, they remain individually admissible as long as $\Pi$ is admissible. Consequently, the Clos theorem may be extended as follows (a re-statement of the corollary on pg. 32):

**Lemma 1.** *An admissible aggregate-request matrix $\Pi$ (5.1) is guaranteed to be satisfied by a sequential Clos fitting algorithm on a time-version Clos network, with $K$ internal timeslots, as long as $K \geq 2M - 1$.*

As there are up to $NM$ circuits to be fitted, the total run-time complexity can be found as $O(NM^2)$. Since every circuit in $\Pi$ is assigned to some configuration, the sequential Clos fitting algorithm essentially partitions $\Pi$ into $\pi^{(k)}$, $1 \leq k \leq K$. Now, consider the operation of a maximal matching algorithm on an admissible $\Pi$. For the purposes of the matching, $\Pi[i,j]$ is viewed as the length of a virtual output queue in a $N \times N$ input-queued packet switch. A sequence of $K$ matchings yields the corresponding configurations $\pi^{(k)}$. From basic combinatorial theory, the condition on $K$ in order to satisfy the aggregate requests is identical to the one in Lemma 1. We highlight this fact in the following result, the proof of which is based on the same pigeon-hole principle as that of the Clos theorem.

**Lemma 2.** *A sequence of $K$ maximal matchings is guaranteed to satisfy an admissible aggregate-request matrix $\Pi$ (5.1), as long as $K \geq 2M - 1$.*

*Proof.* We prove the lemma by contradiction. Let there be a circuit request in $\Pi$ for some input-output pair $(i, j)$, which is not assigned to any of the $K$ configurations. Let $T_{i,j}$ denote the sum of the total requests from input $i$ and the total requests destined to output $j$. Then,

$$\begin{aligned} T_{i,j} &= \sum_l \Pi[i,l] + \sum_l \Pi[l,j] - \Pi[i,j] \\ &\leq 2M - 1 \end{aligned} \tag{5.2}$$

The first two terms above are no greater than $M$ (5.1) and the last term is at least 1 due to the presence of the unassigned element. Let $T'_{i,j}(k)$ denote the sum of assigned circuits from input $i$ and destined to output $j$, in slot $k$. Then,

$$\begin{aligned} T'_{i,j}(k) &= \sum_l \pi^{(k)}[i,l] + \sum_l \pi^{(k)}[l,j] - \pi^{(k)}[i,j] \\ &\geq 1 \quad \forall k. \end{aligned} \tag{5.3}$$

The above is due to the fact that a maximal matching ensures that if there is a pending request for pair $(i, j)$, one of the following is true. Either (a) $\pi^{(k)}[i,j] = 1$, or (b) either $i$ is connected to another output or $j$ is connected to another input, or both. From (5.2) and (5.3), and since $K \geq 2M - 1$,

$$\sum_k T'_{i,j}(k) \geq K \geq T_{i,j},$$

which contradicts the assumption that there is an unassigned circuit for $(i, j)$. □

A sequential maximal matching (Sec. 3.2.2) has a total run-time complexity of $O(N^2M)$. Notice that while a sequential Clos fitting algorithm operates in circuit-sequence, i.e., it finds an internal slot $k$ for each circuit in $\Pi$, a maximal matching operates in timeslot-sequence, i.e., it fills $\pi^{(k)}$ until no more circuits from $\Pi$ can be added to it, in increasing order of $k$. Indeed, the latter is a specific instance of a sequential Clos fitting, where the matching determines the order in which circuits are picked from $\Pi$. Clearly, it is a *strategy* for sequential Clos fitting so that the fitting may be conducted in a timeslot-sequence.

### 5.1.2 Packet-switching Equivalent

Consider now a $N \times N$ CIOQ packet switch (Fig. 2.6). Let the offered average rates be denoted by the matrix $\lambda$. Furthermore, let the rates be admissible, normalized to the external link capacity ($C = 1$), and bounded in timescale $M$, i.e.,

$$\forall i, j, n \;\; \frac{A_{i,j}[n, n+M)}{M} \leq \lambda_{i,j} \tag{5.4}$$

Assume that time is split into intervals of duration $M$ by epochs $kM$, $k \geq 0$. In addition, let the switch operate under batch-mode arbitration by assembling an aggregate request matrix $\Pi^*$ at each epoch, comprised of all the cells that arrived in the preceding interval. More precisely,

$$\forall i, j, k \;\; \Pi^*_{i,j}(kM) = A_{i,j}[(k-1)M, kM) \quad \leq \lambda_{i,j} M \tag{5.5}$$

Since $\lambda$ is admissible, $\Pi^*$ is constrained by the same inequalities as $\Pi$ in (5.1). Therefore, from lemmas 1 and 2, sequential Clos fitting, and equivalently, a sequence of maximal matchings, can both partition $\Pi^*$ into $K$ configurations, as long as $K \geq 2M - 1$. Both the methods behave in exactly the same fashion irrespective of whether the aggregate matrix consists of simultaneous circuit requests in a time-version Clos network, or a batch of cells from a preceding fixed interval in a CIOQ packet switch. In essence, we just proved the Weller-Hajek maximal matching result (paraphrased below) as a direct consequence of the analogy between Clos fitting and maximal matching. Note that all the cells in $\Pi^*(kM)$ will be served before the next epoch as long as the above $K$ configurations can be completed in $M$ external timeslots, leading us to the following:

**Lemma 3.** *Any maximal matching algorithm guarantees exact 100% throughput to bounded admissible arrival traffic, with timescale $M$, as long as the internal speedup $s \geq 2 - 1/M$.*

Due to the bounded nature of the arrivals, the maximum cell delay is no greater than $2M$. This is because every cell that arrived in $[(k-1)M, kM)$ is guaranteed to depart by epoch $(k+1)M$. Notice that as $T \to \infty$, the minimum value of $s$ approaches 2. The above lemma then may be viewed as a short-timescale batch-mode matching version of the Dai-Prabhakar asymptotic-throughput result for maximal matchings. Finally, it should be noted that the maximal matching strategy is more suited for online implementation than Clos fitting because it operates

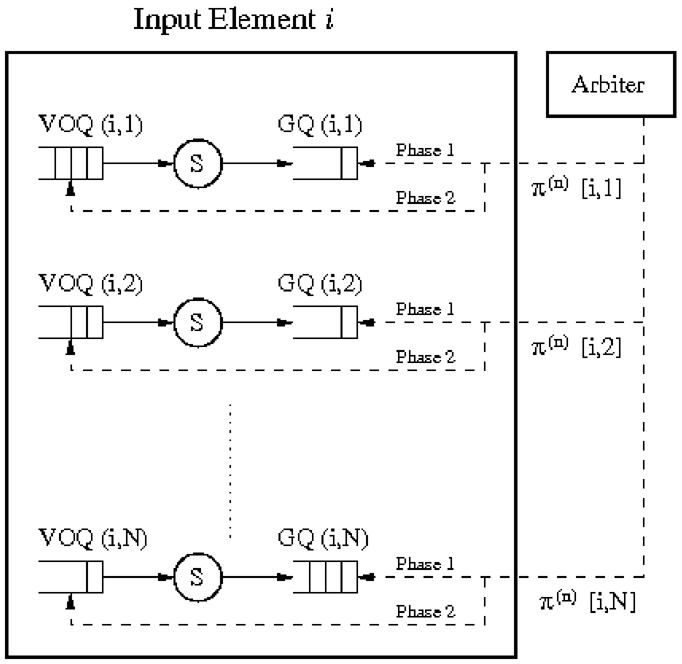


Figure 5.2: CIOQ: Virtual output queues and guaranteed queues

in timeslot sequence on the aggregate matrix. Clos fitting requires the computation of the entire sequence, of $K$ matchings, to be completed before the first cell can be dispatched.

### 5.1.3 Bandwidth and Delay Guarantees

We now remove the constraint on $\lambda$, and apply the above result to ensure bandwidth and delay guarantees. Let the offered arrivals be arbitrary, and let matrix $R$ contain the requested rates for each input-output pair. Let $r$ be the GCD of the values in $R$, and define a timescale $M = 1/r$. We introduce a *rate shaper* in the architecture that schedules the backlogged cells in VOQ $(i, j)$ by transferring up to $R_{i,j}M$ cells into a *guaranteed queue* (GQ) $(i, j)$, as shown in Fig. 5.2 (phase 1), at epochs $kM$, $k > 0$. In other words, using the terminology in Sec. 3.2.1, we batch the arriving cells into an eligible queue-length matrix $\Pi$ at each epoch, where

$$\forall i, j, k \quad \Pi_{i,j}(kM) = \min(R_{i,j}M, Q_{i,j}(kM)) \tag{5.6}$$

We make the following observations. The arrival traffic into the guaranteed queues is bounded-admissible with timescale $M$. From lemma 3, all the cells in $\Pi(kM)$ are guaranteed to be dispatched before epoch $(k+1)M$ by a maximal matching operating on $\Pi$, as long as $s \geq 2 - 1/M$. In summary, the rate shaper ensures that each input-output pair $(i, j)$ is guaranteed the requested rate $R_{i,j}$ into the guaranteed queue, irrespective of the arrivals, while the maximal matching with the specified speedup ensures that every cell in a GQ departs with an additional delay bounded above by $M$ slots. This gives us the main result of this section:

**Theorem 1.** *Any maximal matching, combined with rate shaping, is sufficient to provide isolated bandwidth guarantees to input-output pairs of an input-queued switch, as long as the internal speedup $s \geq 2 - r$.*

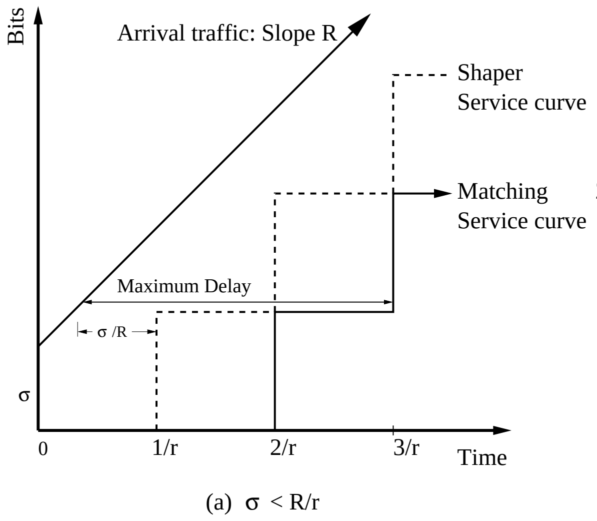
For delay analysis, the input memory element of the CIOQ switch may be viewed as a latency-rate server [104], where the shaper has a latency of $1/r$ due to the burst transfer at the beginning of epochs, and the maximal matching also has a latency of $1/r$. The actual delay bound will depend on the profile of the arriving traffic. For example, if the traffic flow between the input-output pair $(i, j)$ is leaky-bucket [28] constrained with a token-bucket size of $\sigma_{i,j}$, then the worst-case delay[3] of a cell belonging to the flow is given by

$$D_{i,j} \leq \frac{\sigma_{i,j}}{R_{i,j}} + \frac{2}{r} \tag{5.7}$$

This is illustrated in Fig. 5.3. Here, $t = 0$ corresponds to the beginning of a busy period for the considered flow. We use the worst-case arrival curve for a leaky-bucket constrained source, and the worst-case service curve for the batch-mode matching operation. Consider two cases: (a) Let $\sigma_{i,j} \leq R_{i,j}/r$. In this case, basic geometry on the service curves in the figure leads to (5.7). (b) Let $k$ be the largest integer such that $\sigma_{i,j} \geq k\frac{R_{i,j}}{r}$. Then the maximum delay is given by

$$\begin{aligned} D_{i,j} &\leq \frac{k+2}{r} \\ &\leq \frac{\sigma_{i,j}}{R_{i,j}} + \frac{2}{r}, \end{aligned}$$

[3]We refer the reader to [28] for properties of leaky-bucket constrained traffic, and to [104] for the theory of latency-rate servers. Here, it suffices to note that a source is said to conform to a leaky bucket with parameters $(\sigma, R)$ if $\forall t, \tau\ A[t, t+\tau) \leq R\tau + \sigma$. If a server guarantees a trunk of $R$ with a maximum initial waiting time (or latency) of $L$ to such a source, then the delay is upper bounded by $\sigma/R + L$.



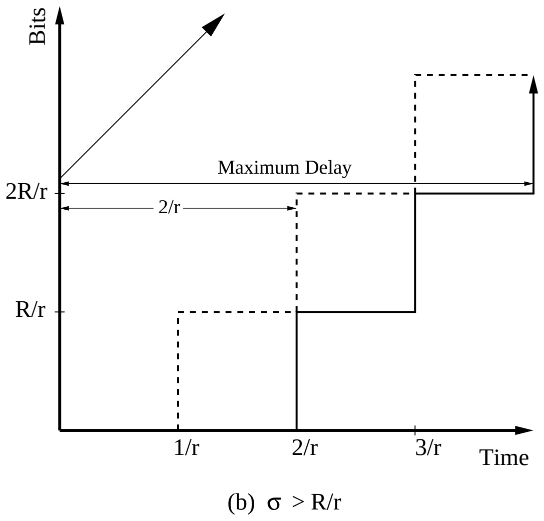


Figure 5.3: Maximum delay for leaky-bucket constrained traffic under maximal matching

which is also covered by (5.7). This delay bound for leaky-bucket constrained traffic, using the online maximal matching, is comparable to the bound provided by a sequence of templates using off-line BVN decomposition of the rate matrix $R$.

We can easily extend the delay analysis to determine the bounds for each fine-grain flow within the input-output pairs. Recall the hierarchical scheduling framework, from Sec. 3.3.2, used to provide QoS guarantees to fine-grain flows in a CIOQ switch. Let $R_{i,j}^{(k)}$ denote the requested rate for flow $k$ belonging to pair $(i, j)$. The shaper merely uses the aggregate rates $R_{i,j}$ to transfer cells to the respective GQ at each epoch, where,

$$R_{i,j} = \sum_k R_{i,j}^{(k)}$$

The GBS component of VOS $(i, j)$ consumes the service turns accorded to pair $(i, j)$ by the batch-mode maximal matching. Let $\alpha_{i,j}$ be the latency in scheduling experienced by a fine-grain flow served by the GBS. For example, if the GBS is a weighted fair-queueing scheduler, then $\alpha_{i,j} \leq 1/R_{i,j}$ for cell-based traffic. If flow $k$ is itself leaky-bucket constrained with token-bucket size $\sigma_{i,j}^{(k)}$, then the per-flow delay bound may be calculated as

$$D_{i,j}^{(k)} \leq \frac{\sigma_{i,j}^{(k)}}{R_{i,j}^{(k)}} + \alpha_{i,j} + \frac{2}{r} \tag{5.8}$$

**Algorithm 1.** *Maximal matching for QoS*

```
Initialize:            For all $(i, j)$
                       1. $Q_{i,j} \leftarrow 0$, $\Pi_{i,j} \leftarrow 0$
                       2. $R_{i,j} \leftarrow$ Rate request for $(i, j)$
                       $M \leftarrow 1/\text{GCD}(R)$

Cell Enqueue:          If cell belongs to $(i, j)$
                       1. Increment $Q_{i,j}$

Every Epoch, $n = kM$: For all $(i, j)$
                       1. $\Pi_{i,j} = \min(R_{i,j}M, Q_{i,j})$

Every Timeslot $n$:    $\pi \leftarrow$ Multi-phase matching
                       If $\pi_{i,j} = 1$
                       1. If $\Pi_{i,j} > 0$, decrement $\Pi_{i,j}$
                       2. Decrement $Q_{i,j}$
```

Figure 5.4: Maximal matching for QoS: Emulating a GQ using counters

Note that, in practice, the batch-mode maximal matching for QoS guarantees operates in the first phase of a multi-phase combination. This phase uses the residual GQ lengths, while additional phases (shown in Fig. 5.2 as phase 2) may augment the matching matrix $\pi(n)$ in each timeslot $n$, in order to optimize throughput. These additional phases continue to operate using the total VOQ lengths. Recognize that if a flow $(i, j)$ is granted service in a subsequent phase, then GQ $(i, j)$ is necessarily empty. Otherwise, it would violate the maximal matching property in phase 1. Therefore, those service grants may be consumed by the EBS component of VOS $(i, j)$ without affecting the bandwidth guarantees.

### Shaping: Implementation

The main implementation advantage of online maximal matchings for QoS guarantees, with respect to template-based approaches, is that it does not need to pre-compute and store configurations. Furthermore, the proposed methodology can actually be implemented without designing rate-shapers or maintaining a separate physical GQ for each input-output pair. This was merely a logical concept, useful for the purpose of analysis. In practice, all that is needed is a counter $\Pi$ per input-output pair, which keeps track of the occupancy of an imaginary GQ. This is in addition to the total queue-length counter $Q$ for the VOQ. The updates of the counters, in order to emulate the desired rate-shaping, is shown in the algorithm in Fig. 5.4.

The added processing required for rate-shaping is simply one counter update in every timeslot. Notice that when a fine-grain flow is added or deleted, it affects the value of $R_{i,j}$ for the respective input-output pair, and possibly the value of $M$. The former is distributed among the different input elements, while a change in the latter is used to reset the epoch interval. In other words, the processing of rate changes is accomplished in a distributed fashion.

## 5.2 Critical Matching Algorithms

Critical matching algorithms (pg. 39), while more complex than maximal matchings, are sometimes more desirable as they provide comparable performance without the need for speedup. Specifically, Weller and Hajek established [114] that a C-MSM (critical with maximum cardinality) provides exact 100% throughput for bounded-admissible arrivals. Here, we will first re-state this result to remove the constraint of maximum cardinality. It is in fact unclear why the authors insist on a critical matching that also maximizes the matching size, a combined constraint that does not lend itself to any known polynomial-time algorithm. It might well have been a small oversight, or possibly a different implied definition of C-MSM. Furthermore, we will demonstrate the application of rate-shaping, seen in the previous section, to extend this result to bandwidth and delay guarantees.

Much as the Dai-Prabhakar maximal matching result, for unbounded traffic, is an analog of the Weller-Hajek batch-mode maximal-matching one, we establish a similar fundamental performance result for critical matchings. We show that a critical matching is sufficient for asymptotic 100% throughput for any admissible arrivals. We prove this using the Dai-Prabhakar fluid-limit theorem, whose only constraint is that the arrivals obey SLLN.

### 5.2.1 Deterministic Properties

Consider again a $N \times N$ CIOQ packet switch (Fig. 2.6). To begin with, let the offered average rates, matrix $\lambda$, be admissible and bounded in timescale $M$ (5.4). Let $\Pi$ be an aggregate request matrix assembled at epochs $kM, k \geq 0$ as shown in (5.5). Then, a C-MSM, operating on the aggregate, is guaranteed to drain all the cells in $M$ slots, according to the Weller-Hajek C-MSM theorem, paraphrased below:

**Lemma 4.** *A critical maximum-size matching guarantees exact 100% throughput to bounded admissible arrival traffic, without internal speedup.*

The worst-case latency of scheduling remains $2M$, much as in the batch-mode maximal matching seen before. Every cell that arrived in $[(k-1)M, kM)$ is guaranteed to depart by epoch $(k+1)M$. We now show that a critical matching, not necessarily maximum, operating on $\Pi$ suffices for the same result. Recall that a line (row or column) of $\Pi$ is deemed *critical* if its sum is the maximum. A critical matching $\pi$ is one that covers all critical lines. Fig. 5.5 shows an algorithm for generating such a matching in the batch-mode. It is based on first applying the Von Neumann augmentation procedure on $\Pi$, at each epoch, to generate a matrix $\Pi'$, in which all the line sums are equal. Then, in each timeslot, an MSM is generated using the bipartite graph corresponding to the residual $\Pi'$, and only those edges $(i, j)$ are retained for which the residual $\Pi_{i,j} > 0$. Since the Von Neumann procedure terminates in $2N - 1$ steps, the complexity of the algorithm is dominated by the MSM for a run-time of $O(N^{2.5})$.

**Lemma 5.** *The matching algorithm in Fig. 5.5 covers* $\Pi$ *in no more than* $M$ *timeslots.*

*Proof.* Notice that $\Pi'$ has equal line sums, equal to $m$. Due to the admissibility of $\lambda$ (5.5), $m \leq M$. From basic combinatorics [43], there exists a matching algorithm $\mathcal{A}$ that covers $\Pi$ in $m$ timeslots[4]. Since there are $Nm$ elements in $\Pi'$, each matching $\pi^{\mathcal{A}}$ generated by $\mathcal{A}$ is perfect.

Consider slot 0 within an interval. Since $|\pi^{\mathcal{A}}| = N$, and since step 3 in each timeslot, in Fig. 5.5, is a maximum size matching, we must have $|\pi'| = N$. In the next slot, the residual $\Pi'$ has equal line sums, equal to $m - 1$. Correspondingly, $\pi^{\mathcal{A}}$ would again be perfect. Using

[4]For example, Slepian-Duguid fitting would partition $\Pi'$ into no greater than $m$ configurations.

**Algorithm 2.** *Batch-mode critical matching*

```
Initialize:                 For all $(i,j)$
                            1. $Q_{i,j} \leftarrow 0$, $\Pi_{i,j} \leftarrow 0$
                            $M \leftarrow$ Timescale of arrivals

Cell Enqueue:               If cell belongs to $(i,j)$
                            1. Increment $Q_{i,j}$

Every Epoch, $n = kM$:      1. $\Pi \leftarrow Q$, $Q \leftarrow 0$     (Batch cells at epoch)
                            2. $\Pi' \leftarrow$ Von Neumann augmentation of $\Pi$

Every Timeslot $n$:         1. For all $(i,j)$, $G_{i,j} = 1_{\{\Pi_{i,j}>0\}}$
                            2. For all $(i,j)$, $G'_{i,j} = 1_{\{\Pi'_{i,j}>0\}}$
                            3. $\pi' \leftarrow$ Maximum size matching on $G'$
                            4. $\pi \leftarrow \pi' \bigcap G$     (Retain only edges in $G$)
                            5. $\Pi \leftarrow \Pi - \pi$
                            6. $\Pi' \leftarrow \Pi' - \pi'$

Von Neumann Augmentation:   1. Define $R_i = \sum_k \Pi_{i,k}$, and $C_j = \sum_k \Pi_{k,j}$
                            2. $\Pi' \leftarrow \Pi$, $m \leftarrow \max_{i,j}\{R_i, C_j\}$
                            3. Find $(i,j)$ s.t. $R_i < m$ and $C_j < m$
                            4. $\epsilon \leftarrow m - \max(R_i, C_j)$
                            5. Increment $\Pi'_{i,j}$ by $\epsilon$
                            6. Increment $R_i$ and $C_j$ by $\epsilon$
                            7. Go back to step 3, until $\forall i,j,\ \ R_i = C_j = m$
```

Figure 5.5: Algorithm for generating a batch-mode critical matching

induction on the sequence of timeslots, each matching $\pi'$ covers $N$ elements. Since $m \leq M$, the algorithm in Fig. 5.5 covers $\Pi'$, and hence $\Pi$, in no more than $M$ timeslots. □

**Lemma 6.** *The matching $\pi$ generated in each timeslot by the algorithm in Fig. 5.5 is critical.*

*Proof.* The algorithm in Fig. 5.5 generates $\pi'$ that has $N$ entries in each timeslot. Therefore, $\pi'$ covers every row and column. Since the critical lines of $\Pi$ were not augmented, $\pi' \bigcap G$ covers those lines in every timeslot. Therefore, $\pi$ is a critical matching. □

Note that since $\pi'$ is perfect, all the line sums of $\Pi'$ decrease by one in every timeslot. Correspondingly, the maximum line sums of $\Pi$ also decrease by one in every timeslot. In other words, if the critical lines were re-determined in each slot (instead of only at the epochs), $\pi$ is still guaranteed to cover them. Furthermore, while $\pi'$ is also a maximum-size matching on the augmented aggregate matrix, $\pi$ need not have maximum cardinality with respect to the original matrix, in each slot. A counter-example is shown in Fig. 5.6. The dotted entries are the ones added by the augmentation procedure. The matchings $\pi$ in slots 1–3 do not have maximum cardinality, with respect to the residual $\Pi$. A different sequence (in the example, the exact reverse of the shown sequence) might result in each matching being maximum, but such a property is not necessary.

As a critical matching covers $\Pi$ in no more than $M$ timeslots (lemma 5), only $M$ matchings are required to service an entire batch, which itself contains cells accumulated over $M$ slots. Therefore, lemma 4, i.e., the Weller-Hajek C-MSM theorem, can be refined as follows:

**Theorem 2.** *A critical matching guarantees exact 100% throughput to bounded admissible arrival traffic in an input-queued switch, without internal speedup.*

Recall that Slepian-Duguid fitting is sufficient to partition $\Pi$ into $M$ conflict-free configurations. Such an algorithm would operate in circuit-sequence, i.e., it finds a slot $k$ for each entry in $\Pi$, possibly rearranging the existing configurations in each step. On the other hand, a critical matching also performs a similar partitioning of $\Pi$, but operates in timeslot-sequence, i.e., it fills $\pi(k)$ in increasing order of $k$, possibly rearranging the current configuration so that it meets the critical criterion. The previously computed configurations remain untouched. In other words,

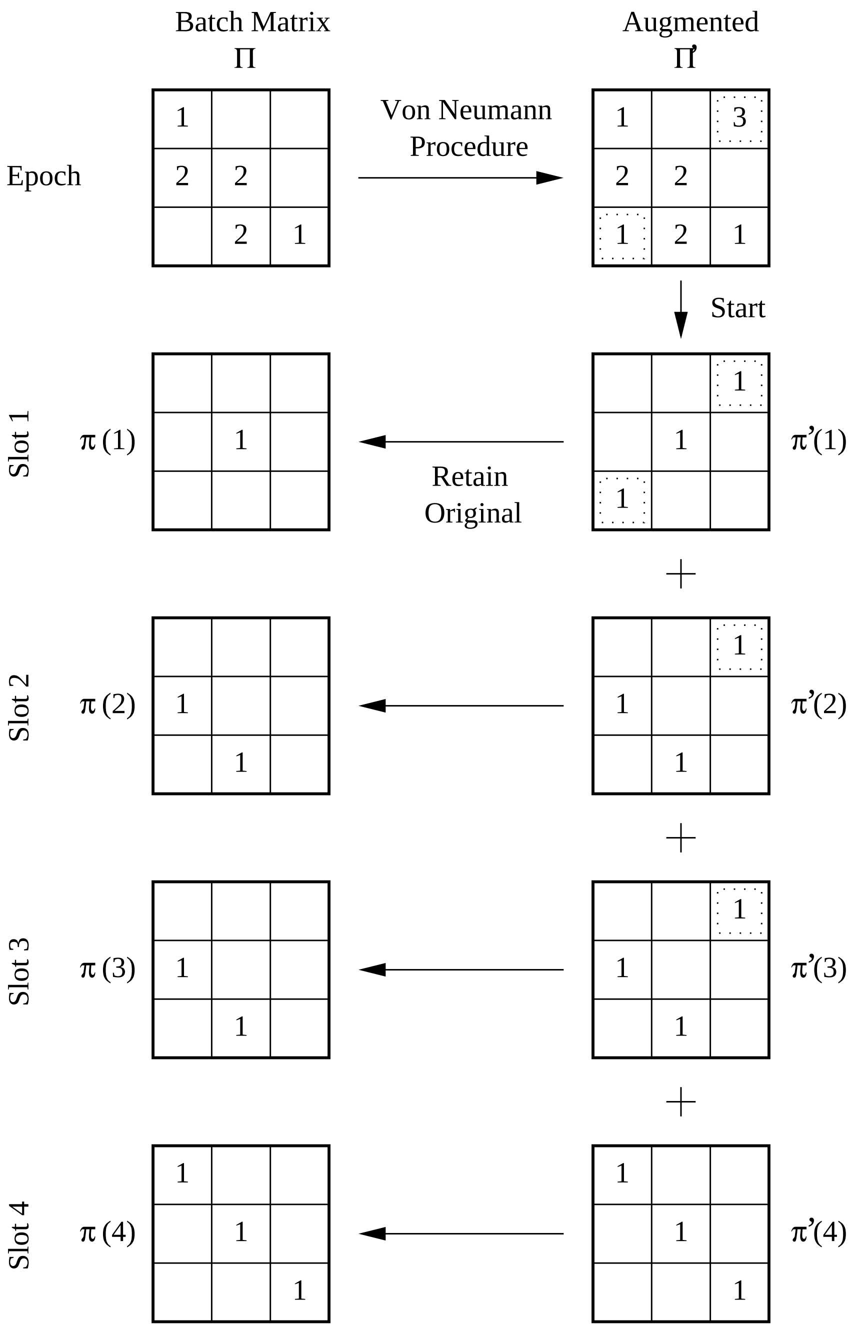


Figure 5.6: Example of a sequence of non-maximum critical matchings

**Algorithm 3.** *Critical matching for QoS*

```
Initialize:              For all $(i, j)$
                         1. $Q_{i,j} \leftarrow 0, \Pi_{i,j} \leftarrow 0$
                         2. $R_{i,j} \leftarrow$ Rate request for $(i, j)$
                         $M \leftarrow 1/\text{GCD}(R)$

Cell Enqueue:            1. If cell belongs to $(i, j)$, increment $Q_{i,j}$

Every Epoch, $n = kM$:   1. For all $(i, j)$, $\Pi_{i,j} = \min(R_{i,j}M, Q_{i,j})$   (Shaper)
                         2. $\Pi' \leftarrow$ Von Neumann augmentation of $\Pi$

Every Timeslot $n$:      1. For all $(i, j)$, $G_{i,j} = 1_{\{\Pi_{i,j}>0\}}$
                         2. For all $(i, j)$, $G'_{i,j} = 1_{\{\Pi'_{i,j}>0\}}$
                         3. $\pi' \leftarrow$ Maximum size matching on $G'$
                         4. $\Pi' \leftarrow \Pi' - \pi'$
                         5. $\pi \leftarrow \pi' \bigcap G$   (Retain only edges in $G$)
                         6. $\pi \leftarrow$ Augment matching (second-phase)
                         7. $\Pi \leftarrow \Pi - \pi$, $Q \leftarrow Q - \pi$

Von Neumann Augmentation: See Fig. 5.5
```

Figure 5.7: Critical matching for QoS: Emulating a shaper using counters

a critical matching may be considered a strategy for Slepian-Duguid fitting on an aggregate request matrix, so that the latter may operate in timeslot-sequence without rearrangements of prior configurations.

### Bandwidth and Delay Guarantees

As with maximal matchings, the exact 100% throughput provided by critical matchings to bounded-admissible traffic can be applied to ensure bandwidth and delay guarantees under arbitrary traffic, given a rate reservation matrix $R$. A rate-shaper, as shown in Fig. 5.2, may be used to create batches $\Pi$ at epochs $kM$, where $M = 1/\text{GCD}(R)$. A critical matching, operating only on $\Pi$, is used in the first phase of a multi-phase combination, and a suitable augmenting matching, operating on $Q$, may be used to simultaneously optimize throughput. The algorithm shown in Fig. 5.5 needs to be slightly modified in order to implement rate-shaping (using counters $\Pi$), and is shown in Fig. 5.7.

**Theorem 3.** *A critical matching, combined with rate shaping, is sufficient to provide isolated bandwidth guarantees in an input-queued switch, without internal speedup.*

*Proof.* Referring to Fig. 5.7, the rate-shaper guarantees $R_{i,j}M$ service turns to pair $(i, j)$ in every $M$ timeslots. The critical matching ensures that all the (shaped) entries in $\Pi$ are drained within an additional $M$ timeslots. Therefore, each input-output pair is guaranteed a virtual bandwidth trunk of $R_{i,j}$. $\square$

The advantage of using critical matchings for QoS is that it needs no internal speedup, much as with offline BVN decomposition. The penalty with respect to the latter is the online complexity of $O(N^{2.5})$. The latency of scheduling is the same as with batch-mode maximal matching, namely, $2M$ timeslots. Therefore, the delay bounds remain exactly the same for leaky-bucket constrained traffic, i.e., the bounds given by (5.7) and (5.8) continue to hold. All the advantages of the batch-mode maximal matching are retained, namely, no template storage overhead and a distributed processing of rate changes on addition and deletion of flows.

### 5.2.2 Stability without Speedup

While the QoS capability of maximal and critical matchings may be viewed as a direct consequence of the work by Weller and Hajek on bounded admissible traffic, providing asymptotic 100% throughput for unbounded admissible arrivals has presented some interesting challenges. The Dai-Prabhakar maximal-matching theorem (pg. 51) establishes the latter, provided the internal speedup is at least 2. The quest for 100% throughput without speedup has been sufficiently addressed by LPF [79] and maximum-weight matching algorithms [80]. To date, these remain the best results in the literature for such performance. A long unanswered question has been whether 100% throughput can be achieved using less complex matchings. A recent exploration of this question is due to Iyer and McKeown [52], who enquire whether the result holds for a general class of maximum-size matchings. Inspired by the Weller-Hajek C-MSM theorem, the authors explore the use of C-MSM for unbounded traffic, and report that it is indeed sufficient for 100% throughput, but for a restricted class of admissible traffic, namely, uniform i.i.d. Bernoulli arrivals.

We posit that asking whether a matching has maximum cardinality is an irrelevant question. While any MSM does maximize instantaneous throughput, it does not necessarily seem to translate to long-term maximum throughput. Instead, a fruitful exploration concerns the ability of a critical matching, not necessarily maximum in size. We prove here that any critical matching, which does not even maximize instantaneous throughput, ends up providing asymptotic 100% throughput for admissible arrivals, without speedup. The proof is based on analyzing the fluid limit of the system of queues.

### Continuous-mode Critical Matching

Consider a $N \times N$ input-queued switch, with admissible offered rates $\lambda$ (2.2). This implicitly also assumes that the average $\lambda_{i,j}$ exists for each input-output pair, i.e., the arrivals conform to SLLN. Let the rates be normalized to the external link capacity, i.e., $C = 1$. As explained in Sec. 3.2.1, let $A$, $D$ and $Q$ be discrete-time $N \times N$ matrices that keep track of the cumulative arrivals, cumulative departures and the current VOQ length, respectively, for each input-output pair. These are related as:

$$\forall i, j, \;\; Q_{i,j}(n) = A_{i,j}(n) - D_{i,j}(n) \tag{5.9}$$

A continuous-mode (as opposed to batch-mode) critical matching $\pi$ is generated in every timeslot, which ensures that all the critical lines of $Q$ are connected. An algorithm to generate the matching is shown in Fig. 5.8. As discussed before, this need not generate a maximum cardinality matching in each slot. The main difference in the continuous-mode algorithm with respect to Fig. 5.5 is that the Von Neumann augmentation procedure is repeated on $Q$ in every timeslot, and there are no epochs. If desired, for some other performance goal, e.g., to decrease the average delay, $\pi$ may be further augmented (not shown) to make it maximal by greedily adding

**Algorithm 4.** *Continuous-mode critical matching*

```
Initialize:                 For all $(i,j)$, $Q_{i,j} \leftarrow 0$
Cell Enqueue:               If cell belongs to $(i,j)$, increment $Q_{i,j}$

Every Timeslot $n$:         1. $Q' \leftarrow$ Von Neumann augmentation of $Q$
                            2. For all $(i,j)$, $G_{i,j} = 1_{\{Q_{i,j}>0\}}$
                            3. For all $(i,j)$, $G'_{i,j} = 1_{\{Q'_{i,j}>0\}}$
                            4. $\pi' \leftarrow$ Maximum size matching on $G'$
                            5. $\pi \leftarrow \pi' \bigcap G$    (Retain only edges in $G$)
                            6. $Q \leftarrow Q - \pi$

Von Neumann Augmentation:   1. Define $R_i = \sum_k Q_{i,k}$, and $C_j = \sum_k Q_{k,j}$
                            2. $Q' \leftarrow Q$, $m \leftarrow \max_{i,j}\{R_i, C_j\}$
                            3. Find $(i,j)$ s.t. $R_i < m$ and $C_j < m$
                            4. $\epsilon \leftarrow m - \max(R_i, C_j)$
                            5. Increment $Q'_{i,j}$ by $\epsilon$
                            6. Increment $R_i$ and $C_j$ by $\epsilon$
                            7. Go back to step 3, until $\forall i,j,\ \ R_i = C_j = m$
```

Figure 5.8: Algorithm for generating a continuous-mode critical matching

connections, which does not affect the critical criterion. Note that the algorithm in Fig. 5.8 can be made more efficient by avoiding the re-computation of $R_i$ and $C_j$, the row and column sums of $Q$, respectively, in each slot. In either case, the run-time complexity is $O(N^{2.5})$, dominated by the MSM operation.

To analyze the stability of the VOQ system, we first need to establish how $D(n)$ evolves under critical matching. Let a $2N$-entry vector $Z(n)$ represent *buckets* that keep track of the cumulative queue length of an entire line of $Q(n)$. The first $N$ entries correspond to rows of $Q(n)$, and the next $N$ to columns. In other words,

$$Z_k(n) = \sum_j Q_{k,j}(n), \quad k = 1, \ldots, N$$

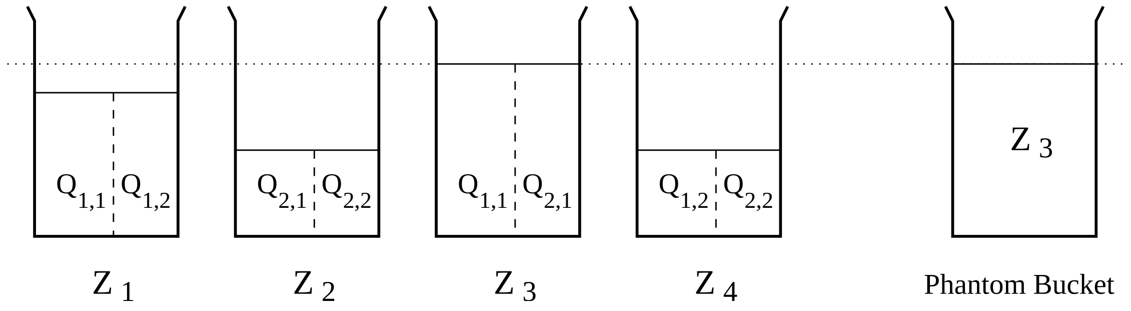


Figure 5.9: Critical matching: Buckets of line sums

$$Z_{N+k}(n) = \sum_i Q_{i,k}(n), \quad k = 1, \ldots, N \tag{5.10}$$

Let $D_{Z_k}(n)$ denote the cumulative departures from bucket $Z_k$ until timeslot $n$. Note that $D_{Z_k}$ is just the sum of the departures from the queues in line $k$ of $Q$. Then, (5.9) and (5.10), together with the critical criterion satisfied by the matching, lead to the following evolution of $Z$:

$$Z_k(n) = \sum_j A_{k,j}(n) - D_{Z_k}(n), \quad k = 1, \ldots, N \tag{5.11}$$

$$Z_{N+k}(n) = \sum_i A_{i,k}(n) - D_{Z_{N+k}}(n), \quad k = 1, \ldots, N \tag{5.12}$$

$$D_{Z_k}(n+1) = D_{Z_k}(n) + 1_{\{Z_k(n)=\sup_i\{Z_i(n)\}\}}, \text{ if } Z_k(n) > 0, \; k = 1, \ldots, 2N \tag{5.13}$$

The last equation merely states that whenever a bucket is non-empty and has the maximum value, it is guaranteed to receive one service turn by the critical matching.

Let $\mathcal{Z}$ be a phantom bucket that always has the maximum line sum. In other words, $\mathcal{Z}$ is a bucket that keeps track of each real bucket, and sets its own length to that of a bucket $k$ that has the maximum length. The identity of $k$ changes at a discontinuity, when the maximum-length bucket changes. Fig. 5.9 illustrates this concept using an analogy of buckets with fluid, for a $2 \times 2$ switch. A critical matching makes sure that a cell is always dispatched from the phantom bucket. To ensure this, all the buckets with the same length as the phantom bucket are ensured one service turn in each timeslot. The specific VOQ that consumes this service is irrelevant. Since a VOQ is represented in two different buckets, if a specific VOQ is starved of service in one bucket, the other bucket will eventually catch up with the maximum. Since the average rate of arrival into the phantom bucket does not exceed one over the long-term, it has a negative drift and cannot grow in an uncontrolled fashion. This is the intuition behind the sufficiency of critical matching for system stability.

Let $D_{\mathcal{Z}}(n)$ denote the cumulative departures from $\mathcal{Z}$ until timeslot $n$. Then, the above equations for $Z_k(n)$ can immediately be applied to $\mathcal{Z}(n)$ as follows:

$$\begin{aligned} \mathcal{Z}(n) &\leq \max_{i,j}\{\sum_k A_{i,k}(n), \sum_k A_{k,j}(n)\} - D_{\mathcal{Z}}(n) &(5.14)\\ D_{\mathcal{Z}}(n+1) &= D_{\mathcal{Z}}(n) + 1, \quad \text{if } \mathcal{Z}(n) > 0 &(5.15)\end{aligned}$$

**Asymptotic 100% Throughput**

We first state the following basic result from calculus:

**Lemma 7.** *Let $f(t)$ be a continuous function of $t$, with $f(0) = 0$. If $f'(t) \leq 0$ for almost every $t$, with respect to Lebesgue measure, such that $f(t) > 0$ and $f$ is differentiable at $t$, then $f(t) = 0$ for almost every $t \geq 0$.*

Here $f'(t)$ is a short-form for $df(t)/dt$. This is the same as lemma 1 in [29], and the reader may refer to that work for a simple proof. For our purposes, given $f(0) = 0$, we shall conclude $f(t) = 0$ whenever it can be established that $f(t) > 0$ implies $f'(t) \leq 0$.

As explained in Sec. 3.2.4, a fluid model may be generated from a continuous-time function $f(t)$ by scaling the time axis. Specifically, the fluid-limit of a random variable $f(t)$ may be obtained as follows (3.6):

$$\bar{f}(t) = \lim_{r\to\infty} \frac{1}{r} f(rt)$$

Assume that the definitions of $(A, D, Q, Z, \mathcal{Z}, D_{\mathcal{Z}})$ are extended so that they become continuous functions in time, e.g., by piece-wise linear interpolation between $(n, n+1)$ for all $n$. Let $(\bar{A}, \bar{D}, \bar{Q}, \bar{Z}, \bar{\mathcal{Z}}, \bar{D}_{\mathcal{Z}})$ be the respective fluid limits. Recall the Dai-Prabhakar fluid-limit theorem, which states that a system of queues is rate-stable if every solution of $\bar{Q}$, with $\bar{Q}(0) = 0$, gives $\bar{Q}(t) = 0$ for all $t \geq 0$. We use this result below to establish the sufficiency of critical matching for asymptotic 100% throughput, for admissible arrivals.

**Theorem 4.** *A critical matching is sufficient, without speedup, for asymptotic 100% throughput in an input-queued switch, for admissible arrivals, i.e.,*

$$\text{(IQ, \{Critical Matching\})} \overset{T,f_2}{\simeq} \text{(OQ, \{WC\})}, T\text{: SLLN}$$

*Proof.* The evolution of $\mathcal{Z}$ (5.14) and $D_{\mathcal{Z}}$ (5.15) translate to the following in the fluid limit:

$$\bar{\mathcal{Z}}(t) \leq t - \bar{D}_{\mathcal{Z}}(t) \tag{5.16}$$

$$\bar{D}'_{\mathcal{Z}}(t) = 1, \text{ if } \bar{\mathcal{Z}}(t) > 0 \tag{5.17}$$

The first inequality is because $\bar{A}_{i,j}(t) = \lambda_{i,j}t$ from the fluid-limit definition of $\bar{A}$, and $\lambda_{i,j}$ (2.1) exists due to SLLN. Furthermore, for all $(i,j)$, both $\sum_i \lambda_{i,j}$ and $\sum_j \lambda_{i,j}$ are no greater than one due to the admissibility constraint. The above equations immediately lead to $\bar{\mathcal{Z}}'(t) \leq 0$ whenever $\bar{\mathcal{Z}}(t) > 0$. From lemma 7, the only solution for $\bar{\mathcal{Z}}$ yields $\bar{\mathcal{Z}}(t) = 0, t \geq 0$.

Notice that $Z_k(n) \leq \mathcal{Z}(n)$ for all $k, n$. From the fluid-limit definition, this also implies $\bar{Z}_k(t) \leq \bar{\mathcal{Z}}(t)$. Therefore, $\bar{Z}_k(t) = 0$ for all $k$. Since the length of VOQ $(i,j)$ at timeslot $n$ is no greater than $Z_i(n)$ and $Z_{N+j}(n)$, we get $\bar{Q}_{i,j}(t) = 0$ for all $(i,j)$. Therefore, using the fluid-limit theorem, all the virtual output queues are rate-stable, thus establishing the sufficiency of critical matchings for asymptotic 100% throughput. □

The above is the strongest result so far regarding throughput of input-queued switches. It subsumes the C-MSM result in [52], which addressed a significantly restricted class of traffic. Also, critical matchings are less complex than LPF [79] and MWM [80], which have been shown to provide similar throughput performance and, to date, were the best results for that purpose. Theorem 4 mirrors the Weller-Hajek (and our refinement) C-MSM theorem, much as the Dai-Prabhakar result, which states that a maximal matching with a speedup of two ensures 100% throughput, mirrors the Weller-Hajek maximal-matching theorem for bounded traffic. In fact, frequent such analogs lead us to the following conjecture, which we have not been able to prove yet in its generality.

**Conjecture 1** (Batch-Continuous Hypothesis)**.** *Given a matching algorithm $\pi$, if for any finite timescale $T$, $\pi$ ensures exact 100% throughput for admissible arrivals bounded by $T$, using a speedup $s$ in the batch-mode, then, $\pi$ ensures asymptotic 100% throughput for unbounded admissible traffic, also with speedup $s$, in the continuous mode.*

In other words, for an arbitrarily bounded traffic, batch the cells arriving in $[(k-1)T, kT)$ into matrix $\Pi$ at epochs $kT$, and let the matching algorithm operate on $\Pi$. If all the cells in $\Pi$ are drained before the next epoch, using a speedup of $s$, then the algorithm also provides asymptotic $100\%$ throughput for unbounded admissible traffic. It does so by operating directly on the VOQ lengths $Q$ (without batching) in every timeslot, also using a speedup of $s$. If the above is true, then, for any matching algorithm, it would suffice to merely inspect its combinatorial properties as to how it decomposes a matrix, and the asymptotic result would follow without needing to study queue stability in each instance. We actually suspect that there is a stronger form of the result, namely, if $\pi$ ensures exact $100\%$ throughput for a specific VOQ in the batch-mode, for bounded traffic, i.e., $\Pi_{i,j}$ is completely drained for some $(i, j)$, then it likely ensures stability of the same VOQ in the continuous-mode, for unbounded traffic, regardless of the overall admissibility.

## 5.3 Uniform Traffic: Sub-maximal Perfect Sequence

An interesting challenge in input-queued switching has been whether it is possible to ensure $100\%$ throughput for admissible arrivals using simple matchings (e.g., simpler than LPF or even a critical matching) without speedup, for arrival patterns that are further restricted. Simulation results have shown that algorithms such as iSLIP [78] and dual round-robin [11] seem to suffice for traffic that is uniformly distributed among the outputs. In fact, iSLIP (pg. 60) has been shown to provide $100\%$ throughput for Bernoulli uniform[5] traffic, with a *single* iteration. The intuition supplied was that a single iteration converges to a maximum-size matching for uniform Bernoulli arrivals, due to the desynchronization of the pointers used by the RGA algorithm.

Due to the popularity of iSLIP, since these results were first shared, there have been several explorations to analytically establish the above. Most (e.g., [52, 97]) have taken it for granted that it must be the fast convergence to a maximum that explains the results, and hence focused on analyzing how an MSM behaves under restricted arrivals. Again, we report here that the cardinality of a matching is an irrelevant property, except maybe for instantaneous throughout and lower average delay. We show that a single iteration of a parallel maximal matching (and hence,

[5]We refer to a traffic pattern as uniform if the arrivals at each input are equally distributed to each of the outputs, in the long term.

**Algorithm 5.** *Sub-maximal Perfect Sequence*

```
Initialize:                 1. For all $(i, j)$, $Q_{i,j} \leftarrow 0$
                            2. Generate perfect sequence $\{\phi_k\}, k = 1, 2, \ldots, N$

Cell Enqueue:               If cell belongs to $(i, j)$
                            1. Increment $Q_{i,j}$

Every Timeslot $n$:         1. $k \leftarrow n \bmod N$
                            2. For all $(i, j)$, $G_{i,j} = 1_{\{Q_{i,j}(n)>0\}}$
                            3. $\pi \leftarrow \phi_k \bigcap G$
                            4. $Q \leftarrow Q - \pi$

Generate Perfect Sequence:  1. For all $(i, j)$, $U_{i,j} \leftarrow 1$
                            2. For $k = 1$ to $N$ do steps 2a, 2b:
                            2a. $\phi_k \leftarrow$ Maximum-size matching on $U$
                            2b. $U \leftarrow U - \phi_k$
```

Figure 5.10: Uniform traffic: A repeating sequence of perfect templates

also iSLIP) is sufficient for 100% throughput for all uniform arrivals. In light of this, optimizing long-term throughput is definitely not a reason to run several iterations of iSLIP, without speedup. Besides, for non-uniform traffic, iSLIP does need $N$ iterations and a speedup of 2, the complexity of which is no better than a centralized maximal matching algorithm, which provides the same performance.

### 5.3.1 SPS Matching

Consider a $N \times N$ input-queued switch, with admissible offered rates $\lambda$ (2.2). Let the rates be normalized to the external link capacity, i.e., $C = 1$. Further, let the arrivals be uniformly distributed to each of the outputs. Therefore,

$$\forall i, j \quad \lambda_{i,j} = \frac{\sum_k \lambda_{i,k}}{N} \quad \leq \frac{1}{N} \tag{5.18}$$

Let the space element of the switch operate under the matching algorithm shown in Fig. 5.10. Notice that this is a template-based approach. We pre-compute a set of $N$ templates $\{\phi_k\}$ that cover a unit matrix. Such a set exists because each line sum of $U$ is equal to $N$, and hence $U$ can be decomposed as a sum of $N$ permutations. Then, each such permutation has $N$ entries, and therefore is a *perfect* matching (see Sec. 3.2.2). Correspondingly, the MSM technique shown at the bottom of Fig. 5.10 will generate a set of $N$ permutations that cover $U$. The templates $\phi_k$ are used in a repeating sequence, such that in $N$ consecutive timeslots, $\phi_k$ is used exactly once, for all $k$. In general, the resultant matching is not even maximal, because there might not be any cells in one or more queues served by $\pi$, and at the same time, there might be cells for other idle pairs $(i, j)$, which remain unconnected. We call such a sequence a *perfect sequence*, and the resultant matching as a *sub-maximal perfect sequence* (SPS) matching.

**Theorem 5.** *An SPS matching is sufficient, without speedup, for asymptotic 100% throughput in an input-queued switch, for uniform admissible arrivals, i.e.,*

$$\text{(IQ, \{SPS Matching\})} \overset{T,f_2}{\simeq} \text{(OQ, \{WC\})}, T\text{: SLLN, Uniform}$$

*Proof.* Each element in $\{\phi_k\}$ is used once every $N$ timeslots. For every $(i, j)$, $U_{i,j} = 1$. Therefore, for some $k$, $\phi_k$ covers $(i, j)$, and every pair receives one service turn in every $N$ timeslots. Since $\lambda_{i,j} \leq 1/N$ (5.18), every queue is rate-stable. □

Note that, strictly speaking, strong stability of the queues, i.e., $E[Q_{i,j}(n)] < \infty$, requires the line sums to be strictly less than 1, i.e., $\sum_k \lambda_{i,k} < 1$. In reality, the above theorem is a direct consequence of the Birkhoff result (3.5) that was used in BVN decomposition. Uniform arrivals are so benign that even if we do not precisely know the offered rates, we may behave as if each $\lambda_{i,j}$ is equal to $1/N$, and use a pre-computed set of templates. Also, it just so happens that for uniform traffic, such a set is among the simplest.

Theorem 5 may easily be extended towards deterministic properties and QoS. Specifically, let the arrivals be bounded in timescale $M$. Let $M^*$ be the least common multiple (LCM) of $M$ and $N$. Batch cells at epoch $kM^*$ to generate $\Pi$. Clearly, $\Pi$ will be drained before the next epoch by a batch-mode SPS, since each pair receives $M^*/N$ (an integer) turns in every batch-interval, which is no less than the number of cells accumulated ($\lambda_{i,j} M^*$) over the previous interval. Thus, SPS provides exact 100% throughput, with a latency of twice the LCM$(M, N)$. Next, if a given rate-reservation matrix $R$ is uniform, or merely, if $R_{i,j} \leq 1/N$ for each $(i, j)$, SPS with rate-shaping may be used to provide isolated bandwidth and delay guarantees, in a similar fashion as seen in the previous sections. The shaping interval is just $N$ timeslots, as all the rates are considered equal. Therefore, the worst-case delay is determined by a scheduling latency of $2N$ slots. For example, for leaky-bucket constrained traffic, the delay bounds in (5.7) and (5.8) continue to hold with $2/r$ replaced by $2N$. These extensions are summarized in the following corollaries.

**Corollary 1.** *An SPS matching guarantees exact 100% throughput to uniform bounded-admissible arrivals in an input-queued switch, without internal speedup.*

**Corollary 2.** *An SPS matching, combined with rate shaping, is sufficient to provide isolated bandwidth guarantees in an input-queued switch, without internal speedup, provided $R_{i,j} \leq 1/N$, $\forall i, j$.*

### 5.3.2 Online Variants

A perfect sequence[6] may be generated rather easily using online means, instead of pre-computing templates. A single step of the EREW maximal matching (see Sec. 3.2.6) may be used to yield such a sequence, as shown in Fig. 5.11. Note that this is a parallel matching, without the conventional request-grant-accept phases. Each input memory element uses an output-pointer, which is desynchronized during initialization, and remains so for the lifetime of the switch. The element merely inspects the VOQ $(i, P_i(n))$, and dispatches a cell to output $P_i(n)$ if the VOQ is non-empty. The algorithm runs in $O(1)$ time.

**Theorem 6.** *The EREW sub-maximal matching is sufficient, without speedup, for asymptotic 100% throughput in an input-queued switch, for uniform admissible arrivals.*

*Proof.* Referring to Fig. 5.11, notice that $P_i(n) \neq P_j(n)$, for all $i, j, n$. Therefore, $\pi(n)$ is a perfect matching in every timeslot. Furthermore, $P_i(n)$ covers all the outputs in any $N$ consecu-

[6]The usage of such sequences is not new. For example, online perfect sequences have been used elsewhere in the design of load-balanced BVN switches [9].

**Algorithm 6.** *EREW sub-maximal matching*

```
Initialize:         1. For all $(i, j)$, $Q_{i,j} \leftarrow 0$
                    2. For all $i$, $P_i = i$ (Initialize pointers)

Cell Enqueue:       If cell belongs to $(i, j)$
                    1. Increment $Q_{i,j}$

Every Timeslot $n$: For all $i$ in parallel
                    1a. $\pi_{i,j} = 1$ iff $j = P_i$
                    1b. If $Q_{i,P_i} > 0$, decrement (dequeue)
                    2. Increment $P_i$ mod $N$
```

Figure 5.11: EREW sub-maximal matching: Online generation of a perfect sequence

tive timeslots. Therefore, the resultant matching belongs to the class of SPS. From theorem 5, an EREW sub-maximal matching is sufficient for 100% throughput. □

This is one of the simplest practical matchings for input-queued switches. The concurrent-dispatch algorithm in the Atlanta chipset [19] is a variant of such an algorithm. Above, we have analytically established what had been observed in simulation studies in that work. This possibly also explains similar observations for iSLIP, with a single iteration, for Bernoulli uniform traffic. It remains unclear why iSLIP needs $N$ iterations, leading to a maximal matching, for 100% throughput to bursty uniform traffic. We venture that this is because iSLIP loses its perfect matching property for bursty traffic, as its main priority is to converge to a maximal match. Each parallel iteration in iSLIP is actually an $O(N)$ operation, in which $P_i$ is used merely as a starting point to scan all the outputs. As we have seen, however, maximality is not required to provide 100% throughput for uniform arrivals.

We conclude by noting that any SPS algorithm seems to have limited applicability for non-uniform traffic, even with speedup. However, we can make the following straight-forward extension for such traffic, with some restrictions. If, for all $(i, j)$, $\lambda_{i,j} \leq k/N$, for some real $k$, $1 < k < N$, then, an SPS matching with speedup $k$ will provide 100% throughput. Other than for non-integer values of $k < 2$, this algorithm offers no performance benefits with respect to a maximal matching, for such traffic.

## 5.4 Maximal Matching for Inadmissible Traffic

We now focus on the performance of an input-queued switch under inadmissible traffic. The goal is to ensure that instability is isolated to outputs that receive more traffic than can be handled by the output link. More precisely, we wish to find matching algorithms that can provide wide-sense relative stability with a work-conserving OQ switch, or equivalently, $f_3$ equivalence. Consider a $N \times N$ input-queued switch, with (unknown) normalized offered rates $\lambda$ (2.1). Furthermore, let $\lambda$ be inadmissible, i.e., it violates (2.2). Let $A$ be the set of oversubscribed outputs. Then, the following is true for $\lambda$ and $A$:

$$\begin{aligned} \forall i \ \sum_j \lambda_{i,j} &\leq 1 \\ \forall j \notin A \ \sum_i \lambda_{i,j} &\leq 1 \\ \forall j \in A \ \sum_i \lambda_{i,j} &> 1 \end{aligned} \tag{5.19}$$

Then, $f_3$ equivalence with a work-conserving OQ switch may be achieved if all the virtual output queues $(i, j)$ for $j \notin A$ remain stable. Such a treatment of inadmissible traffic puts us in unchartered territory with respect to the literature.

Note that an algorithm such as LOOFA [65], which ensures work-conservation in a CIOQ switch, with a speedup no less than 2, would ensure $f_3$ equivalence. We report here that *any* maximal matching with the same speedup also suffices. This strengthens the Dai-Prabhakar maximal matching theorem, by extending it to inadmissible arrivals. At first glance, this seems like a counter-intuitive result. For example, if a VOQ $(i, j)$ for $j \in A$ becomes unstable, as should be the case, one might expect a bad matching to relentlessly connect $(i, j)$ at the expense of some other queue $(i, k)$, $k \notin A$, thereby also making the latter unstable. Indeed, this situation would arise with an MWM that uses the queue lengths as weights, or even with a critical matching. However, since an input can never be oversubscribed due to physical link limitations (first row of (5.19)), maximal matching ends up isolating instability to oversubscribed outputs.

### 5.4.1 Bounded Arrivals and Statistical Multiplexing

To provide intuition, and to demonstrate yet another instance of the batch-continuous hypothesis, we first explore the behavior of the system for bounded traffic. Let the arrivals be bounded in timescale $M$. Then the offered-rate matrix satisfies (5.4) and (5.19). Let cells be batched at epochs $kM$, $k > 0$, as shown in (5.5). The aggregate matrix $\Pi$ is served by a maximal matching. The following result states that $\Pi_{i,j}(kM)$ is completely drained before epoch $(k+1)M$ for $j \notin A$.

**Theorem 7.** *For arrivals bounded in timescale $M$, in an input-queued switch, any maximal matching provides exact 100% throughput to flows $(i, j)$ for which output $j$ is not oversubscribed, as long as the internal speedup $s \geq 2 - 1/M$.*

*Proof.* The proof is similar to the one for lemma 2. From (5.5) and (5.19), $\Pi$ satisfies the following:

$$\begin{aligned} \forall i, k \quad \sum_j \Pi_{i,j}(kM) &\leq M \\ \forall j \notin A, \forall k \quad \sum_i \Pi_{i,j}(kM) &\leq M \end{aligned}$$

In other words, even in the presence of inadmissible traffic, each row sum of $\Pi$ does not exceed $M$, and the same is true for each column sum of $\Pi$ corresponding to a non-oversubscribed output. Let $T_{i,j}$ denote the number of cells originating from $i$ plus the number destined to $j$. Then, for $\Pi_{i,j}(kM) > 0$,

$$\begin{aligned} \forall j \notin A, \forall k \qquad T_{i,j}(kM) &= \sum_l \Pi_{i,l}(kM) + \sum_l \Pi_{l,j}(kM) - \Pi_{i,j}(kM) \\ &\leq 2M - 1 \end{aligned}$$

No cells are added to $T_{i,j}$ in $[kM, (k + 1)M)$. Furthermore, assume $\Pi_{i,j} > 0$ at the end of the batch-interval. Then, a maximal matching would have ensured that $T_{i,j}$ decreases by 1 in each internal slot in the interval. If the number of internal timeslots is no less than $2M - 1$, then the above assumption is contradicted for $j \notin A$. Therefore, for such outputs, $\Pi_{i,j}$ is completely drained before the next epoch, as long as $s \geq 2 - 1/M$. □

An example is illustrated in Fig. 5.12(a), for a $4 \times 4$ switch and $M = 10$. The respective line sums are shown next to the rows and columns. Here, outputs 2 and 4 are oversubscribed. For

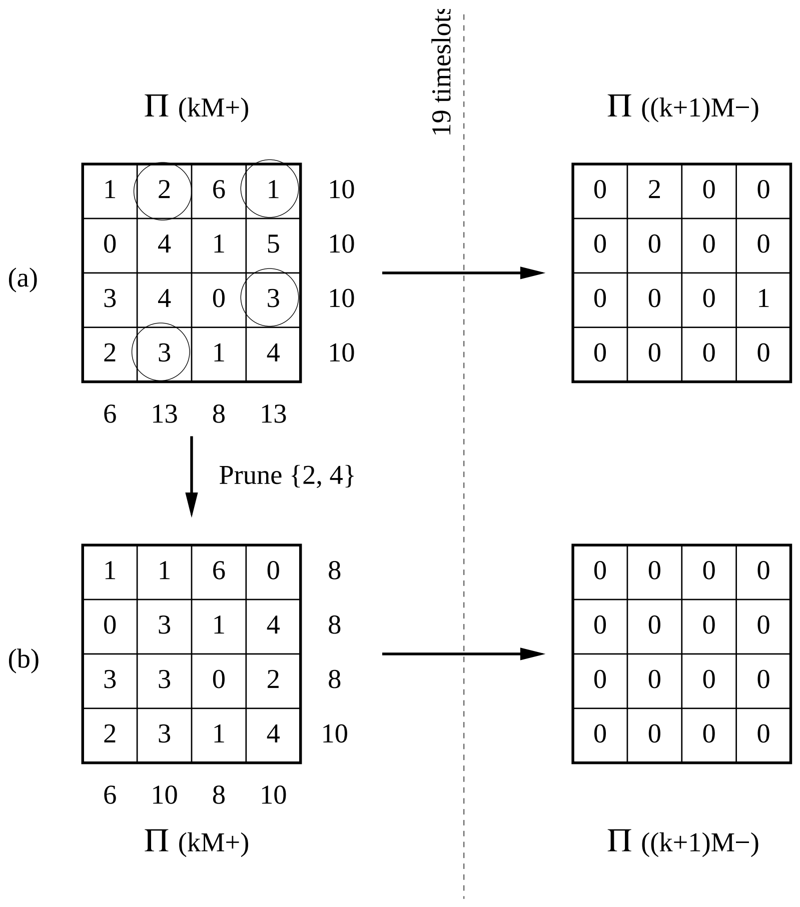


Figure 5.12: Example: Batch-mode matching for inadmissible traffic

any $(i, j)$, as long as $\Pi_{i,j}$ remains non-zero, either row $i$ or column $j$ is served by the maximal matching. For $(i, j)$ such that $j \notin \{2, 4\}$, since both the row sum and the column sum do not exceed $M = 10$ at the beginning of an interval, $\Pi_{i,j}$ cannot remain non-zero after $2M - 1 = 19$ internal timeslots. Therefore, all the cells destined to non-oversubscribed outputs $\{1, 3\}$ are served before the next epoch, if $M$ external slots map to $2M - 1$ internal ones.

## Residue of Matching

In the previous example, the residue of $\Pi$ at the end of the batch-interval contains some unserved cells from $\{2, 4\}$. Even within an oversubscribed output, a maximal matching has special affinity

**Algorithm 7.** *Residue-aware maximal matching*

```
Initialize:            For all $(i,j)$, $Q_{i,j} \leftarrow 0$, $Q'_{i,j} \leftarrow 0$, $\Pi_{i,j} \leftarrow 0$
Cell Enqueue:          If cell belongs to $(i,j)$, increment $Q_{i,j}$

Every Epoch, $n = kM$: 1. $Q' \leftarrow Q' + \Pi$   (Update residue)
                       2. $\Pi \leftarrow Q - Q'$   (Arrivals in preceding interval)
                       3. Transfer entries from $Q'$ to $\Pi$ s.t. $\forall i$, $\sum_j \Pi_{i,j} \not> M$

Every Timeslot $n$:    1. For all $(i,j)$, $G_{i,j} = 1_{\{\Pi_{i,j}>0\}}$
                       2. $\pi \leftarrow$ Maximal matching on $G$
                       3. $\Pi \leftarrow \Pi - \pi$
                       4. $Q \leftarrow Q - \pi$
```

Figure 5.13: Batch-mode maximal matching with residue management for inadmissible traffic

for pairs $(i,j)$ for which $T_{i,j} \leq 2M-1$ at the beginning of an interval, and all the cells for such pairs are served anyway. Among pairs for which $T_{i,j} > 2M-1$, circled in the figure, some cells remain unserved as long as a cell is dispatched from input $i$ or output $j$ in each internal timeslot. Notice that an adversarial maximal matching, which always gives preference to a pair with a higher $T_{i,j}$, can in fact completely drain $\Pi$ in this example. It is easy to see that the strict[7] conditions for a non-zero residue at epoch $(k+1)M$ are:

$$\begin{aligned} \sum_{j\in A} \sum_{i\in B_j(kM)} \Pi_{i,j}(kM) &> 2M-1 \text{ and} \\ \exists j \text{ s.t.} \sum_{i\in B_j(kM)} \Pi_{i,j}(kM) &> M, \end{aligned}$$

where $B_j(kM) = \{i | T_{i,j}(kM) > 2M-1\}$. If both inequalities hold, even a maximal matching that gives strict priority to pairs $(i,j)$, such that $j \in A$ and $i \in B_j$, might not prevent a residue for some pair in output $j$ that satisfies the second condition. Minimizing the residue is a moot point, however, since serving more than $M$ cells for an output will lead to instability in the respective

[7]These conditions are not sufficient, but are strictly necessary. Otherwise, an adversarial maximal matching can completely drain $\Pi$.

third-stage memory-element queue.

It is important that a residue is not added to a future batch without special care, otherwise a *row sum* might exceed $M$, and theorem 7 might not hold anymore. There are two options to address this issue: (a) Cells that are not served at the end of an interval may be immediately discarded, thereby ensuring that a VOQ only contains arrivals from the preceding interval, at each epoch; (b) Alternatively, the switch may keep track of the cumulative residue $Q'_{i,j}$, and selectively admit them in future batches, as long as the row sums are not violated. The latter is illustrated in Fig. 5.13. Note that $\forall j \notin A$, $Q'_{i,j}(n) = 0$, $\forall n$. Therefore, a residue admission will not affect column sums for $j \notin A$. In summary, flows $(i, j)$ with $j \notin A$ receive exact 100% throughput, and those with $j \in A$ receive 100% throughput with some statistical multiplexing losses.

### 5.4.2 Matching on Pruned Requests

Referring to Fig. 5.12(b), a desired residue may be pre-computed by *pruning* an aggregate request matrix before matching. More precisely, $\Pi$ may be set to $Q$ at each epoch, after a sufficient number of cells are discarded from $Q$ to ensure $\sum_i Q_{i,j} \leq M$, for all outputs $j$. In this case, the row and column sums of $\Pi$ do not exceed $M$, and every entry will be covered by a maximal matching before the next epoch. Notice that, at each epoch, $Q$ contains only the cells that arrived in the preceding interval, and no cells need to be dropped for $j \notin A$.

This approach, in contrast to the residue-management options (a) and (b) above, maximizes the discarded residue. However, it opens up the possibility of using other matching algorithms, preferably with lower required speedup, which could otherwise not have ensured 100% throughput for non-oversubscribed outputs. For example, a critical matching cannot be used in lieu of a maximal one in Fig. 5.13. The addition of request pruning, shown[8] in Fig. 5.14, allows the application of a critical matching, leading to the following corollary. Note that simpler matchings such as SPS for restricted arrivals do not seem to find similar applications. In fact, SPS itself is rendered irrelevant since no arrival pattern can be uniform and inadmissible at the same time.

**Corollary 3.** *For arrivals bounded in timescale $M$, in an input-queued switch, any critical matching with request-pruning provides exact 100% throughput to flows $(i, j)$, such that output $j$ is not*

[8]The pruning operation in Fig. 5.14 can be made more efficient by discarding several cells in the same step.

**Algorithm 8.** *Critical Matching with Pruning*

```
          Initialize:  For all $(i, j)$, $Q_{i,j} \leftarrow 0$, $\Pi_{i,j} \leftarrow 0$
        Cell Enqueue:  If cell belongs to $(i, j)$, increment $Q_{i,j}$

  Every Epoch, $n = kM$:  1. Prune $Q$: For all $j \in A$
                            1a. $i \leftarrow 1$
                            1b. While $\sum_i Q_{i,j} > M$ do steps 1c, 1d
                            1c. If $Q_{i,j} > 0$, decrement $Q_{i,j}$   (Discard cell)
                            1d. Increment $i$ mod $N$
                          2. $\Pi \leftarrow Q$
                          3. $\Pi' \leftarrow$ Von Neumann augmentation of $\Pi$

      Every Timeslot $n$:  Same as in Fig. 5.7
Von Neumann Augmentation:  Same as in Fig. 5.7
```

Figure 5.14: Batch-mode critical matching with pruning for inadmissible traffic

*oversubscribed, without internal speedup.*

The flows destined to oversubscribed outputs experience statistical multiplexing, due to the pruning operation, with losses given by the following mass function.

$$\forall j, k \;\; p(X_{j,k} = X) = \text{Prob}\{\sum_i A_{i,j}[kM, (k+1)M) > M + X\},$$

where $X_{j,k}$ denotes the number of cells arriving in $[kM, (k+1)M)$ that will be pruned and discarded at epoch $(k+1)M$. With pruning, there should be no losses in a work-conserving third-stage element. Even for inadmissible traffic, $p(X)$ could end up being rather small if not all sources are expected to be active at the same time. The exact computation of the above sum will depend on the joint distribution of the arrival processes. If we assume that the pruning is fair, i.e., each flow destined to $j \in A$ experiences equal discards, then the expected loss for any flow $(i, j)$ may be computed as follows:

$$E[L_{i,j}] = \frac{1}{N}\{E[\sum_i A_{i,j}[kM, (k+1)M)] - M\} \tag{5.20}$$

Further exploration of such statistical multiplexing is beyond the scope of this work, but presents an interesting challenge.

**Bandwidth Guarantees**

The results for maximal and critical matching algorithms, for bounded inadmissible traffic, may be easily extended to bandwidth and delay guarantees. Let $R$ be a rate-request matrix, with one or more oversubscribed outputs. In other words, the admission control procedure allows $R$ to violate output-link capacities, in order to take advantage of statistical multiplexing. A batch-mode maximal matching, with $M = 1/\text{GCD}(R)$, combined with rate-shaping, which ensures that no more than $RM$ cells are considered in each batch, ensures a bandwidth trunk of $R_{i,j}$ to each $(i, j)$ in non-oversubscribed outputs. If request pruning is added, then the pairs $(i, j)$ for oversubscribed outputs are provided trunks with an expected loss given by (5.20). With pruning, the latency of scheduling remains $2M$ for all pairs.

**Corollary 4.** *Any maximal matching, combined with rate shaping, provides isolated bandwidth guarantees to pairs $(i, j)$, such that $j$ is not oversubscribed, with speedup $s \geq 2 - GCD(R)$.*

**Corollary 5.** *Any critical matching, combined with request pruning and rate shaping, provides isolated bandwidth guarantees to pairs $(i, j)$, such that $j$ is not oversubscribed, without internal speedup.*

### 5.4.3 Stochastic Stability

Explicit residue management, and request pruning, for oversubscribed outputs is not feasible for unbounded inadmissible traffic, because $\lambda$ is not known, neither is a suitable timescale available. The only information given is (5.19), and the goal is to ensure wide-sense relative stability with an OQ switch, by guaranteeing VOQ stability for $j \notin A$. For such outputs, recall that a batch-mode maximal matching did not rely on any residue management. We show here that continuous-mode maximal matching provides asymptotic 100% throughput to non-oversubscribed outputs even for unbounded traffic, with a speedup of at least 2. We use the same combinatorial property of maximal matching as in the bounded case, namely, if $Q_{i,j}$ remains non-zero, either input $i$

or output $j$ is guaranteed a service turn in each timeslot. The proof, based on fluid limits, is similar to the one in [29], with minor modifications to only consider non-oversubscribed outputs. The following result strengthens the Dai-Prabhakar maximal matching theorem by removing the constraint of admissibility.

**Theorem 8.** *A maximal matching is sufficient for asymptotic 100% throughput, in an input-queued switch, to flows $(i, j)$ belonging to non-oversubscribed outputs, as long as the speedup $s \geq 2$. i.e.,*

$$\text{(CIOQ, \{Maximal Matching\})} \overset{T, f_3}{\simeq} \text{(OQ, \{WC\})},\ s \geq 2, T\text{: SLLN}$$

*Proof.* Fix $s = 2$. For each $(i, j)$ define $T_{i,j}$ as:

$$\forall i, j, n \quad T_{i,j}(n) = \sum_k Q_{i,k}(n) + \sum_k Q_{k,j}(n)$$

From (5.9) and the property of maximal matching, we can state the following, where $D_{T_{i,j}}(n)$ denotes the sum of the departures from row $i$ and column $j$, until slot $n$:

$$\begin{aligned} \forall i, j, n \qquad T_{i,j}(n) &= \sum_k A_{i,k}(n) + \sum_k A_{k,j}(n) - D_{T_{i,j}}(n) && (5.21) \\ D_{T_{i,j}}(n+1) &\geq D_{T_{i,j}}(n) + 2, \quad \text{whenever } Q_{i,j}(n) \geq 2 && (5.22) \end{aligned}$$

Let $(\bar{A}, \bar{Q}, \bar{T}, \bar{D}_T)$ be the respective fluid limits (3.6) of $(A, Q, T, D_T)$ after necessary linear interpolations. By definition, $\bar{A}_{i,j}(t) = \lambda_{i,j} t$. Combined with (5.19) (assuming a strict inequality in its second row), the above equations translate to the following in the fluid limit.

$$\begin{aligned} \forall i, t\ \forall j \notin A \quad \bar{T}_{i,j}(t) &< 2t - \bar{D}_{T_{i,j}}(t) && (5.23) \\ \bar{D}'_{T_{i,j}}(t) &\geq 2, \quad \text{whenever } \bar{Q}_{i,j}(t) > 0 && (5.24) \end{aligned}$$

The second row is on account of the linear interpolation of $D_{T_{i,j}}(n)$, due to which $D_{T_{i,j}}(t)$ decreases at a rate of 2 the instant $Q_{i,j}(t)$ becomes greater than 0. This immediately gives $\bar{T}'_{i,j}(t) < 0, \forall t \geq 0$, whenever $\bar{Q}_{i,j}(t) > 0$, for all pairs $(i, j)$ such that $j \notin A$. This is sufficient for rate stability, using the Dai-Prabhakar fluid-limit theorem, as elaborated below[9].

[9]We remind the reader that the proof so far mimics the proof for the Dai-Prabhakar maximal matching theorem [29].

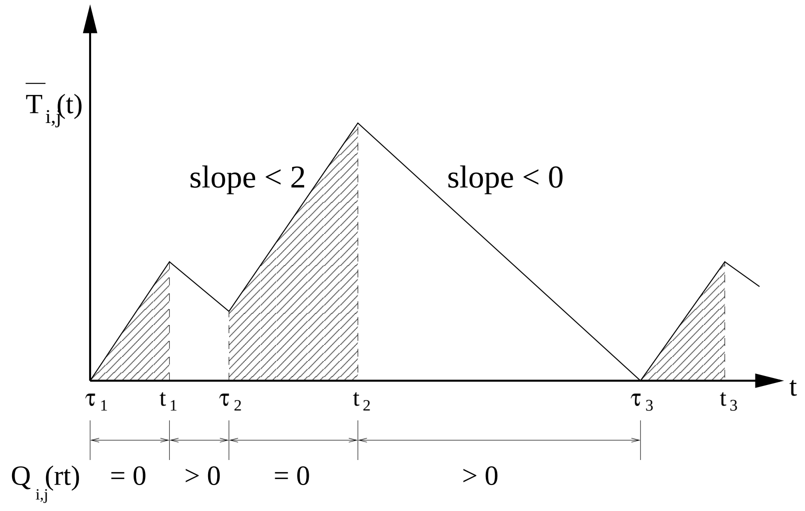


Figure 5.15: How $\bar{T}_{i,j}(t)$ varies with $t$ based on $Q_{i,j}(rt)$

Recognize that $\bar{T}_{i,j}(t)$ is upper bounded by the curve shown in Fig. 5.15. The shaded portions indicate intervals for which $Q_{i,j}(rt) = 0$, where $r$ is the scaling parameter of the fluid limit. In such intervals, $\bar{T}_{i,j}(t)$ increases at a rate no greater than 2. The instant $t$ at which $Q_{i,j}(rt)$ becomes positive, $\bar{T}_{i,j}(t)$ decreases at some finite rate. Define two time series:

$$\{\tau_k | Q_{i,j}(r\tau_k-) > 0 \text{ and } Q_{i,j}(r\tau_k) = 0\} \quad \text{and}$$
$$\{t_k | Q_{i,j}(rt_k-) = 0 \text{ and } Q_{i,j}(rt_k) > 0\}$$

If $t_k$ is an infinite series, then VOQ $(i,j)$ is stable since the queue returns to zero infinitely often. More precisely, we may re-define (pg. 49) the fluid limits as

$$\bar{Q}_{i,j}(t) = \lim_{k\to\infty} \frac{1}{r_{n_k}} Q_{i,j}(r_{n_k}t) \quad \text{where} \quad r_{n_k}t = rt_k$$

In such a limit, $\bar{Q}_{i,j}(t) = 0, \forall t \geq 0$, leading to rate stability.

The latter elegantly proceeds to establish $\bar{Q}_{i,j}(t) = \bar{T}_{i,j}(t) = 0$ for all pairs $(i,j)$ $\forall t$ by proving that the inner product $f(t) = \langle \bar{Q}(t), \bar{T}(t) \rangle = 0$. This is based on the fact that $f(t) \geq 0$ and

$$f'(t) = \langle \bar{Q}(t), \bar{T}'(t) \rangle < 0.$$

We need to depart from this approach because a partial inner product that only considers pairs $(i,j)$ s.t. $j \notin A$ does not yield $f'(t) = \langle \bar{Q}(t), \bar{T}'(t) \rangle$. Instead, we resort to first principles of fluid limits. Notice in our proof that, for inadmissible traffic, $\bar{T}_{i,j}(t)$ need not be 0, even for pairs in non-oversubscribed outputs.

We prove that $t_k$ is an infinite series by contradiction. Let $|\{t_k\}| = X$, where $X$ is finite. By definition, there are no arrivals for $(i, j)$ in $[r\tau_k, rt_k)$ $\forall k$. Recall that

$$\begin{aligned} \lim_{r\to\infty} A_{i,j}[r\tau_k, rt_k) &= \lambda_{i,j}(t_k - \tau_k) \\ &> 0 \quad \text{whenever } t_k > \tau_k. \end{aligned}$$

Therefore, as $r \to \infty$, $t_k \to \tau_k$. Since $\bar{T}_{i,j}(t)$ is upper bounded by the curve in Fig. 5.15, we obtain

$$\bar{T}_{i,j}(t_X) \leq 2X \sum_{k=1}^{X} (t_k - \tau_k).$$

For finite $X$, $\bar{T}_{i,j}(t_X) \to 0$, and the entire series repeats itself, i.e., $|\{t_k\}| > X$, leading to a contradiction. Hence $t_k$ is an infinite[10] series. □

Theorems 7 and 8 provide an instance of the (strong-form) batch-continuous hypothesis. Since batch-mode critical matchings require the addition of request-pruning for inadmissible traffic, its analog is meaningless in the continuous mode. The above, then, is quite a remarkable result for a general maximal matching, since it provides superior performance compared to more complex algorithms, like MWM or critical matching, in the presence of inadmissible traffic. The speedup of 2 is not prohibitive, as we will demonstrate later that such speedup is required anyway for simultaneous satisfaction of QoS and optimal throughput. In light of this, and the fact that it remains one of the simplest algorithms to implement, one would need a very strong reason to prefer some other method for arbitration in an input-queued switch.

## 5.5 Strict Relative Stability

Our next focus is on providing strict relative stability to input-output flows in an input-queued switch, with respect to a work-conserving OQ switch that employs some flow-based scheduler $S$ at its output links. More precisely, we wish to find matching algorithms that can ensure that every flow $(i, j)$ that is stable in the reference switch is also stable in the input-queued switch under

[10]For further intuition, notice that if $|\{t_k\}|$ is infinite, the last inequality need not tend to 0, neither is it guaranteed to be finite. This is as expected since there are components in $\bar{T}_{i,j}(t)$ that belong to oversubscribed outputs. Notice also that any finite value of $\bar{T}_{i,j}(t_X)$, for finite $X$, would lead to the same conclusion, since the non-zero negative slope when $Q_{i,j}(rt) > 0$ would ensure that $\bar{T}_{i,j}(t)$ returns to 0, thereby causing the series to repeat.

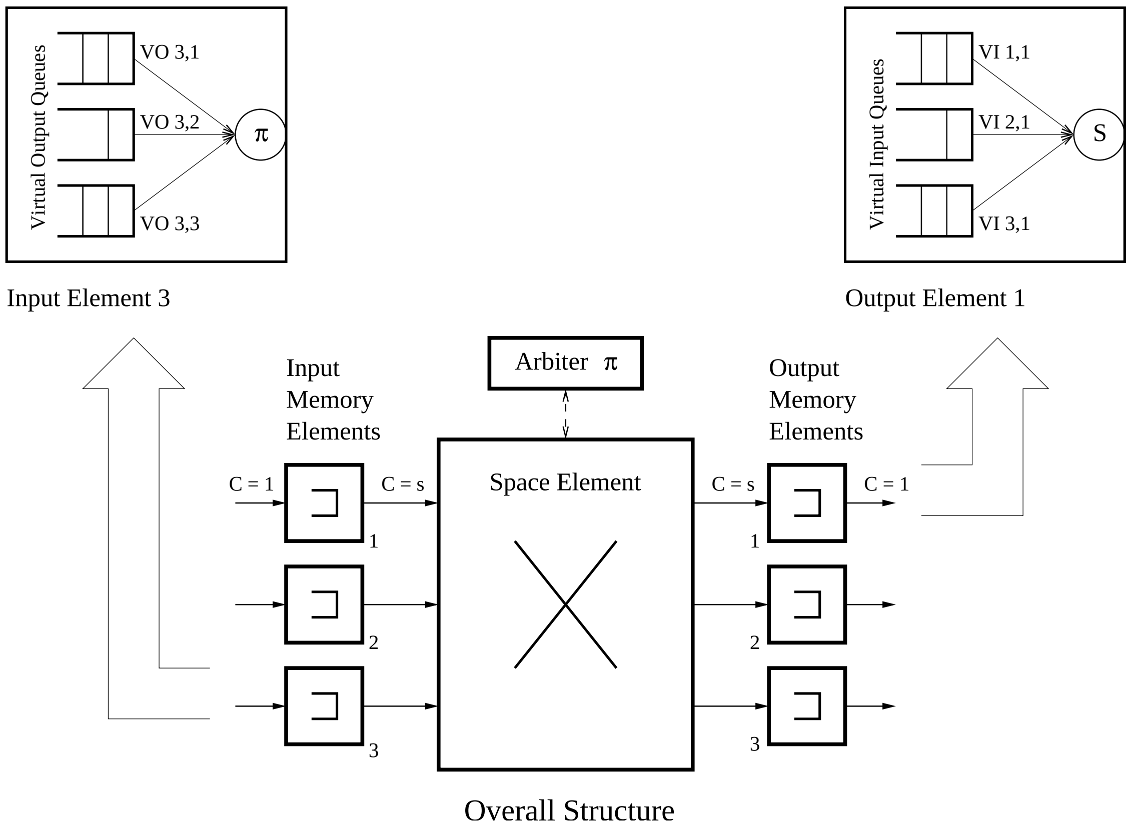


Figure 5.16: A CIOQ switch with virtual input queueing in the output elements

consideration, for the same arrival processes. In other words, the goal is to isolate instability on a flow basis. We may say then that the input-queued switch provides the same long-term fairness as an ideal switch in the distribution of output-link bandwidth. From the point of view of stable flows, the switch would asymptotically emulate the ideal.

Consider a $N \times N$ CIOQ switch, under a discrete-time arrival process specified by matrix $A(n)$, with (unknown) normalized offered rates $\lambda$ (2.1). Let the link scheduler of the third-stage memory element be set to $S$, the output link scheduler of the reference OQ switch. We assume *virtual input queueing* (VIQ) in the third-stage, which would enable $S$ to choose any desired flow $(i, j)$ at each output $j$, without head-of-line blocking. This is illustrated in Fig. 5.16. We additionally define the following $N \times N$ matrices:

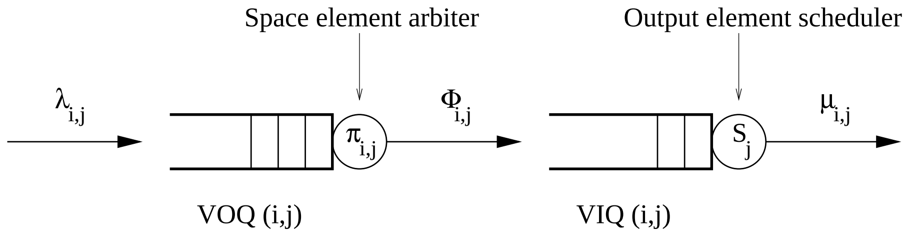


Figure 5.17: A CIOQ switch as a set of tandem queues

| | | |
|---|---|---|
| $Q_{i,j}(n)$ | : | Length of VOQ $(i,j)$ at timeslot $n$ |
| $D_{i,j}(n)$ | : | Cumulative departures from VOQ $(i,j)$ until timeslot $n$ |
| $X_{i,j}(n)$ | : | Length of VIQ $(i,j)$ at timeslot $n$ |
| $\Phi_{i,j}$ | : | Long-term departure rate from VOQ $(i,j)$ |
| $\mu_{i,j}$ | : | Long-term departure rate from the switch |
| $\mu^R_{i,j}$ | : | Long-term departure rate from the reference switch under $A(n)$ |

Then, each flow is served by a tandem of queues as shown in Fig. 5.17. The VOQ is served by the arbiter of the space element, and gets a service whenever $\pi_{i,j} = 1$, and the VIQ is served by the output link scheduler $S$.

If $\lambda$ is admissible, a maximal matching ($s \geq 2$) or a critical matching ($s \geq 1$) is sufficient to ensure stability for each VOQ. Since $S$ is work-conserving, each VIQ is also stable. Therefore, additional means are required only when $\lambda$ is inadmissible. Furthermore, if output $j$ is not oversubscribed ($\sum_i \lambda_{i,j} < 1$), a maximal matching with $s = 2$ is sufficient for the stability of flows $(i,j)$, from the previous section. Hence, the issue of strict relative stability is meaningful only for flows destined to oversubscribed outputs. Let $B$ denote the set of *conforming flows*, defined as follows:

$$B = \{(i,j)|\lambda_{i,j} = \mu^R_{i,j}\}$$

In other words, for the same arrival processes, flow $(i,j)$ is stable in the reference switch. Short of emulating the reference, $\mu^R_{i,j}$ is not known. As shown in Sec. 4.2.2, even if we know the static weights used by $S$, $\mu^R_{i,j}$ cannot be deduced. It merely happens as a result of the operation of $S$

**Algorithm 9.** *Shortest output-queue first matching*

```
Initialize:          1. For all $(i, j)$, $Q_{i,j} \leftarrow 0$
Cell Enqueue:        1. If cell belongs to $(i, j)$, increment $Q_{i,j}$

Every Timeslot $n$:  1. $Y \leftarrow$ List of $(i, j)$ sorted by $X_{i,j}$
                     2. For all $(i, j)$, $\pi_{i,j} \leftarrow 0$
                     3. For $k = 1$ to $N^2$ do steps 3a, 3b, 3c
                       3a. $(i, j) \leftarrow Y[k]$
                       3b. If $\sum_i \pi_{i,j} = 0$ and $\sum_j \pi_{i,j} = 0$, $\pi_{i,j} \leftarrow 1$
                       3c. If $Q_{i,j} = 0$, $\pi_{i,j} \leftarrow 0$
                     4. $Q \leftarrow Q - \pi$
```

Figure 5.18: Shortest output-queue first (SOQF) maximal matching

in the reference switch, under the specific arrival pattern given by $A$. However, we know that $\lambda$ satisfies the following:

$$\begin{aligned} \forall i, \quad \sum_j \lambda_{i,j} &\leq 1 \\ \forall (i,j) \in B, \quad \sum_i \lambda_{i,j} &\leq 1 \end{aligned} \tag{5.25}$$

The first inequality is due to the physical limitation of the input links, and the second due to the fact that $\lambda_{i,j}$ is equal to $\mu^R_{i,j}$ for $(i, j) \in B$, which cannot sum up to a value greater than 1, for each output. We establish strict relative stability if the VOQ and VIQ of each conforming flow are both stable in the CIOQ switch.

A comparative stability analysis requires a characterization of how $S$ behaves in the two systems. We define $S$ to be *asymptotically convergent* if, given $S$ provides a long-term departure rate of $\mu^{(1)}$ under arrivals with rate $\lambda^{(1)}$, it will dispatch at a rate of at least $\mu^{(1)}_{i,j}$ for each flow $(i, j)$ if offered at least $\mu^{(1)}_{i,j}$ by that flow. Specifically, let $\mu^{(2)}$ be the departure rate corresponding to some arrival process with rate matrix $\lambda^{(2)}$. Then,

$$\begin{aligned} \forall (i,j), \quad \mu^{(2)}_{i,j} &= \lambda^{(2)}_{i,j} \quad \text{if } \lambda^{(2)}_{i,j} \leq \mu^{(1)}_{i,j} \\ &\geq \mu^{(1)}_{i,j} \quad \text{otherwise} \end{aligned}$$

Note that this property is defined in terms of the *long-term* departure rates, and furthermore, not on the basis of any static service parameters used by the scheduler. This definition is satisfied by a broad class of link schedulers, for example, any work-conserving scheduler based on weights or WFQ.

Notice that, in our tandem-queue system (Fig. 5.17), $S$ operates on the virtual input queues, whose long-term arrival rates are given by $\Phi$. Let $S$ be asymptotically convergent. Then, the following is true for the departure rates from the respective VIQ:

$$\begin{aligned} \forall(i,j), \quad \Phi_{i,j} \leq \mu^R_{i,j} &\quad\Rightarrow\quad \mu_{i,j} = \Phi_{i,j} \\ \text{and} \quad \Phi_{i,j} > \mu^R_{i,j} &\quad\Rightarrow\quad \mu_{i,j} \geq \mu^R_{i,j} \end{aligned} \tag{5.26}$$

Since $(i,j) \in B$ is covered by the first row, VIQ $(i,j)$ is stable for conforming flows, though the converse is not necessarily true. A non-conforming flow can end up receiving more than its respective $\mu^R$ and remain stable, at the expense of some other flows. Irrespective of this fact, a conforming flow will receive 100% throughput if its VOQ is stable.

The only algorithm in the literature that can ensure VOQ stability for $(i,j) \in B$ under inadmissible traffic is the stable matching [24, 107] policy that exactly emulates the reference OQ switch (see pg. 58). For our less stringent goal, we show that a greedy maximal matching that proceeds in a sorted order of the VIQ lengths is sufficient for strict relative stability, as long as the speedup $s \geq 3$. The intuition is that the length of the VIQ represents the congestion experienced by each input-output pair under $S$, which may be used by the space element to serve less congested flows, thereby allowing conforming flows to receive the same throughput as in an ideal switch.

### 5.5.1 Shortest Output-Queue First

We introduce the *shortest output-queue first* (SOQF) maximal matching algorithm, shown in Fig. 5.18. Here, an output queue refers to the virtual input queue of a flow, in the output memory element. In each timeslot $n$, flows are sorted in increasing order of $X_{i,j}(n)$. A maximal matching steps through this list and connects idle pairs, without backtracking. When $Q_{i,j}(n)$ is decremented, the corresponding $X_{i,j}(n)$ is incremented, as is expected in a tandem-queue sys-

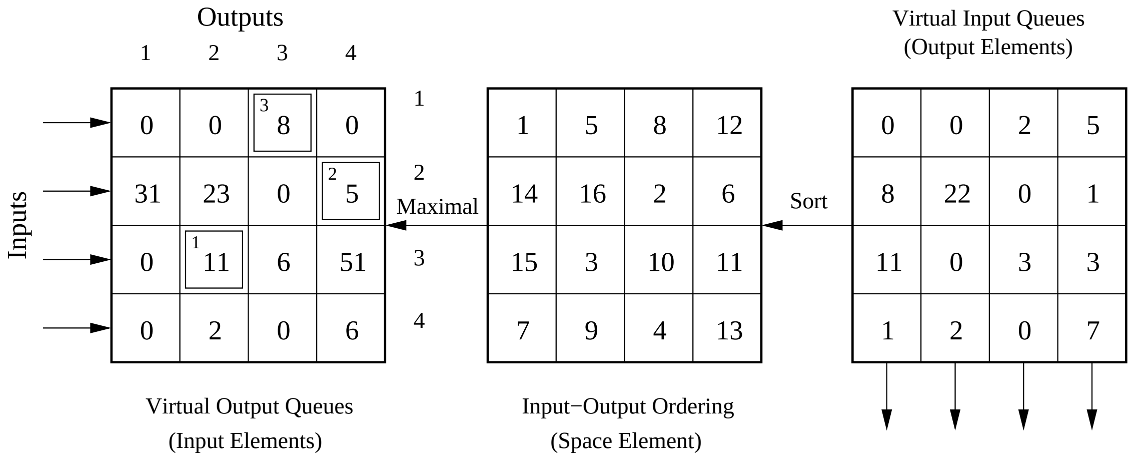


Figure 5.19: Example: SOQF maximal matching algorithm

tem. An example is shown in Fig. 5.19, in which the boxed pairs denote the ones that become connected. The order in which they are chosen is indicated in the left corner. The complexity of the algorithm is dominated by the sorting operation, which runs in $O(N^2 \log N)$ time. The rich literature in parallel sorting may be used to decrease this run-time, e.g., using Cole's merge sort [26], which runs in $O(\log N)$ time using $N^2$ EREW comparators, or using a parallel Batcher sorting network. The complexity of SOQF then becomes $O(N^2)$ like any other maximal matching algorithm. We could not find an EREW implementation that simultaneously sorts and chooses elements satisfying the conflict-free property of the matching (2.5).

Since SOQF is maximal, it provides $f_3$ equivalence with a work-conserving OQ switch with $s \geq 2$, in accordance with Theorem 8. In this section, we prove using the fluid-limit approach that all conforming flows, even those destined to oversubscribed outputs (not addressed by $f_3$), remain stable as long as $s \geq 3$. Note that SOQF does not rely on knowing $\mu^R$. Instead, it lets $S$ operate independently, and deduces which flows are preferred by $S$ by monitoring the state of the virtual input queues. Also, unlike stable matching algorithms, SOQF does not simulate the OQ link scheduler to drive the matching. Rather, it feeds sufficient traffic to less congested flows, and the asymptotically convergent scheduler does the rest. Any flow that is served by the arbiter in excess of what $S$ can handle experiences larger queues in the output element, and the arbiter corrects itself. Since a conforming flow does not *offer* traffic in excess of what $S$ can handle, the VOQ ends up being stable.

The defining property of SOQF is that if a flow $(i, j)$ has a cell in its VOQ, then either $\pi_{i,j} = 1$, $\pi_{i,k} = 1$ for some flow $(i, k)$ with a shorter or equal VIQ, or $\pi_{k,j} = 1$ for a flow $(k, j)$ with a shorter or equal VIQ. The matching is not sensitive to the specific VOQ lengths. In other words, for $T_{i,j}$ defined below, a cell is dispatched from at least one of the queues on the right hand side, in each internal timeslot, if $Q_{i,j}(n) > 0$.

$$T_{i,j}(n) = \sum_{\{(i,k):X_{i,k}(n)\leq X_{i,j}(n)\}} Q_{i,k}(n) + \sum_{\{(k,j):X_{k,j}(n)\leq X_{i,j}(n)\}} Q_{k,j}(n) \tag{5.27}$$

**Asymptotic 100% Throughput**

From (5.26), we know that VIQ $(i, j)$ is stable for $(i, j) \in B$. However, given VIQ $(i, j)$ is stable, it does not imply $(i, j) \in B$. To aid in analysis, we further partition the flow-space and define set $C$ as follows:

$$C = \{(i, j) \notin B | \Phi_{i,j} = \mu_{i,j}\}$$

In other words, $C$ refers to the set of non-conforming flows, which end up with stable virtual input queues. The arbiter neither knows the identity of such flows during its online operation, nor does it use this knowledge in the matching. Note that the following is true for the set $D$ of non-conforming flows without stable VIQ:

$$D = \{(i, j) \notin C | \Phi_{i,j} > \mu^R_{i,j}\}$$

We replace each virtual output queue of the tandem system in Fig. 5.17 with a pair of *imaginary* queues $Z$ and $Y$, defined as follows. For $(i, j) \in \{B \cup D\}$, $Z_{i,j}$ is the same as VOQ $(i, j)$, with offered rate $\lambda_{i,j}$ and departure rate $\Phi_{i,j}$, and $Y_{i,j}$ is always empty. For $(i, j) \in C$, $Z_{i,j}$ is a queue that receives a portion of the offered traffic, equal to $\mu_{i,j}$, and $Y_{i,j}$ retains the rest, i.e., $\lambda_{i,j} - \mu_{i,j}$. This is illustrated in Fig. 5.20, in which $Z_{i,j}$ contains no more than $s$ cells (in fact, any finite $|Z_{i,j}|$ would work), and whenever it is served, a cell is transfered from $Y_{i,j}$ to $Z_{i,j}$. The arbiter only serves $Z$, and $Y$ is ignored. We just converted a two-queue tandem system into a three-queue system for non-conforming flows, for which it is given that the respective VIQ is stable.

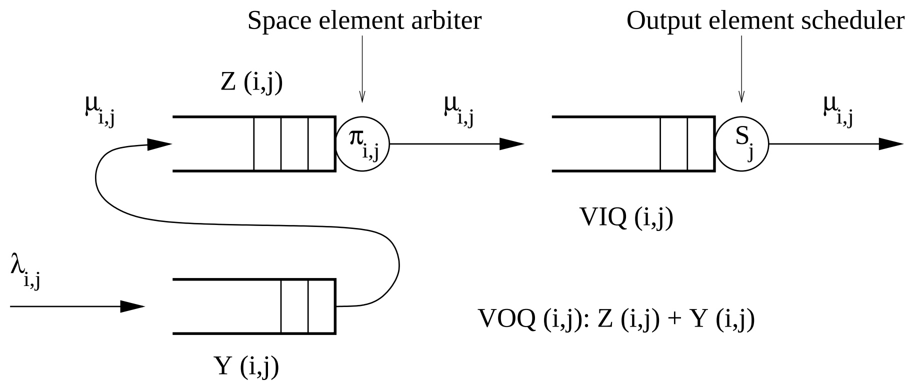


Figure 5.20: Imaginary queue system for non-conforming flows with stable output queues

We make the following observations about the imaginary queue system. The sequence of configurations $\pi$ generated by SOQF in the imaginary system is identical to the one in the real system. This is because SOQF depends only on whether a VOQ is non-empty, and $Z_{i,j}$ is always non-empty whenever $Q_{i,j} > 0$. Furthermore, for $(i,j) \in B$, stability of $Z_{i,j}$ immediately implies that VOQ $(i,j)$ is stable, thus establishing $f_4$ equivalence with the reference switch. We redefine $T_{i,j}(n)$ from (5.27) as $C_{i,j}(n)$ for the imaginary queue system, and let $D_{C_{i,j}}(n)$ be the cumulative departures from the queues represented in $C_{i,j}(n)$, until timeslot $n$. Then, the SOQF property may be re-written as:

$$\begin{aligned} \forall(i,j) \quad C_{i,j}(n) &= \sum_{\{(i,k):X_{i,k}(n)\le X_{i,j}(n)\}} Z_{i,k}(n) + \sum_{\{(k,j):X_{k,j}(n)\le X_{i,j}(n)\}} Z_{k,j}(n) \\ D_{C_{i,j}}(n+1) &\ge D_{C_{i,j}}(n) + s, \qquad \text{whenever } Z_{i,j}(n) \ge s \end{aligned} \tag{5.28}$$

**Theorem 9.** *SOQF maximal matching is sufficient for strict relative stability of an input-queued switch, with respect to an OQ switch employing an asymptotically convergent link scheduler $S$, as long as the speedup $s \ge 3$, i.e.,*

$$\text{(CIOQ, \{SOQF, } S\text{\})} \overset{T,f_4}{\simeq} \text{(OQ, \{WC, } S\text{\})},\ s \ge 3,\ T\text{: SLLN}$$

*Proof.* Let $(\bar{Z}, \bar{X}, \bar{C}, \bar{D}_C)$ be the respective fluid limits (3.6) of $(Z, X, C, D_C)$ after necessary linear interpolations to make the latter continuous functions of time. Then, by definition[11],

$$\forall(i,j) \quad \bar{C}_{i,j}(t) = \sum_{\{(i,k):\bar{X}_{i,k}(t)\le\bar{X}_{i,j}(t)\}} \bar{Z}_{i,k}(t) + \sum_{\{(k,j):\bar{X}_{k,j}(t)\le\bar{X}_{i,j}(t)\}} \bar{Z}_{k,j}(t)$$

For $(i,j) \in B$, we know that $\bar{X}_{i,j}(t) = 0$, $t \ge 0$, since the VIQ is stable. Also, for $(i,j) \in C$, $\bar{Z}_{i,j}(t) = \bar{X}_{i,j}(t) = 0$, $t \ge 0$. Finally, since the VIQ is unstable for $(i,j) \in D$, $\bar{X}_{i,j}(t) > 0$, $t \ge 0$. Therefore, for conforming flows $(i,j)$, $\bar{X}_{k,l}(n) \le \bar{X}_{i,j}(n)$ implies $(k,l) \in B \cup C$. For such flows, then, the SOQF property (5.28) translates to the following:

$$\begin{aligned}
&\forall(i,j) \in B : \\
\bar{C}_{i,j}(t) &= \sum_{(i,k)\in B\cup C} \bar{Z}_{i,k}(t) + \sum_{(k,j)\in B\cup C} \bar{Z}_{k,j}(t) && (5.29)\\
&= \left( \sum_{(i,k)\in B\cup C} \lambda_{i,k}t + \sum_{(k,j)\in B} \mu^R_{k,j}t + \sum_{(k,j)\in C} \mu_{k,j}t \right) - \bar{D}_{C_{i,j}}(t) && (5.30)\\
&\le 3t - \bar{D}_{C_{i,j}}(t) && (5.31)\\
\bar{D}'_{C_{i,j}}(t) &\ge s, \qquad \text{whenever } \bar{Z}_{i,j}(t) > 0 && (5.32)
\end{aligned}$$

Note that (5.30) results from the fact that, for $(i,j) \in B$, $\lambda_{i,j} = \mu^R_{i,j}$, and for $(i,j) \in C$, the arrival rate into $Z_{i,j}$ is equal to $\mu_{i,j}$. The first summation is no greater than $t$ because of the physical limitation of input link $i$, and the second[12] and third are no greater than $t$ because the departure rates cannot exceed 1 for the entire column $j$ in both, the switch under consideration, and the reference switch. The above set of equations imply, for $(i,j) \in B$, $\bar{C}'_{i,j}(t) \le 0$ whenever $\bar{Z}_{i,j}(t) > 0$, $t \ge 0$, as long as $s \ge 3$.

Let $f_X(t)$ be the partial inner product of $\bar{Z}(t)$ and $\bar{C}(t)$, defined for flows with a stable virtual input queue. That is,

$$f_X(t) = \sum_{(i,j)\in B\cup C} \bar{Z}_{i,j}(t)\bar{C}_{i,j}(t)$$

[11]The fluid limits in the ranges of the summations follow from the limit definition. Specifically, start with all pairs $(i,k)$ s.t. $X_{i,k}(rt) \le X_{i,j}(rt)$, where $r$ is the scaling parameter. If $\bar{X}_{i,k}(t) > \bar{X}_{i,j}(t)$, it would violate the previous inequality.

[12]We may not assume the second sum to be $\sum \mu_{k,j}t$, as it is indeed $\lambda_{k,j} = \mu_{k,j}$ that we are trying to prove. However, we may for the third sum, by construction. Moreover, we are given that the VIQ for $(i,j) \in C$ is stable. We did not use the same condition to similarly split the traffic for $(i,j) \in B$ since the proof needs $Z_{i,j} = Q_{i,j}$ for such flows.

Clearly, $f_X(t) \geq 0$, for all $t$. Furthermore, $f_X(t) = 0$ implies $\bar{Z}_{i,j}(t) = 0$ for $(i,j) \in B$, and hence $\bar{Q}_{i,j}(t) = 0$. For $(i,j) \in C$, we already know that $\bar{Z}_{i,j}(t) = 0$, and no new information can be gathered about $\bar{Q}_{i,j}(t)$. Consider the case: $f_X(t) > 0$. Then, using similar algebra as in [29], we get

$$\begin{aligned} f'_X(t) &= 2 \sum_{\{(i,j)\in B\cup C:\bar{Z}_{i,j}(t)>0\}} \bar{Z}_{i,j}(t)\bar{C}'_{i,j}(t) \\ &= 2 \sum_{\{(i,j)\in B:\bar{Z}_{i,j}(t)>0\}} \bar{Z}_{i,j}(t)\bar{C}'_{i,j}(t) \\ &\leq 0, \qquad \text{whenever } f_X(t) > 0, s \geq 3. \end{aligned}$$

From lemma 7, $f_X(t) = 0, t \geq 0$. This implies that $\bar{Z}_{i,j}(t) = 0, t \geq 0$, for all conforming pairs. Hence, from the Dai-Prabhakar fluid-limit theorem, the virtual output queues of such flows are rate-stable. □

In fact, it is possible to devise an alternate proof by relating $\bar{C}_{i,j}(t)$ and $\bar{Z}_{i,j}(t)$ using time series $\{\tau_k, t_k\}$, in a similar fashion as shown in Fig. 5.15. We chose to use the inner product to relate the two quantities, as in [29], due to its elegance.

The role of the imaginary-queue system as an analytical tool should now be apparent. It allowed us to characterize the behavior of non-conforming flows $C \cup D$, without requiring an explicit determination of their long-term departure rates. For such flows, either the VIQ becomes unstable ($D$), which is monitored by the arbiter allowing it to give preference to less congested flows, or we know that the arbiter has already (magically!) isolated the instability caused by such flows ($C$) to their respective VOQ. Note that our result establishes $\mu_{i,j} = \lambda_{i,j} = \mu^R_{i,j}$ for $(i,j) \in B$, as long as $s \geq 3$. However, we have not established the same for non-conforming flows. We know, by the definition of $D$ and (5.26), that $\mu_{i,j} \geq \mu^R_{i,j}$ for $(i,j) \in D$. Furthermore, the departure rates must satisfy $\forall j, \sum_i \mu_{i,j} \leq \sum_i \mu^R_{i,j}$. Otherwise, the asymptotically convergent scheduler in the reference OQ switch would have ensured that the departure rates in that switch end up higher than the given values. Therefore, we can state the following, for $s \geq 3$:

$$\begin{aligned} \forall(i,j) \in B, \quad \mu_{i,j} &= \mu^R_{i,j} \\ \forall(i,j) \in C, \quad \mu_{i,j} &\leq \mu^R_{i,j} \\ \forall(i,j) \in D, \quad \mu_{i,j} &\geq \mu^R_{i,j} \end{aligned}$$

Ideally, we would have liked equalities for each of the above sets. Nevertheless, for equivalence purposes, queue stability for conforming flows suffices. It is worth noting here that a practical implementation of SOQF might be encumbered by its high required speedup, and the complexity of the sorting operation. In reality, while we proved the sufficiency of $s \geq 3$, we suspect that the same goal can be achieved with lesser speedup. More precisely, we now know that $\mu_{i,j} \leq \mu^R_{i,j}$ for all flows in $C$, if the VOQ of conforming flows are stable. If this fact could have been established independently, without presuming stability, then the summations in (5.30) would lead to a value no greater than $2t$, by collapsing the last two sums into a single sum, no greater than $t$. Consequently, we could have established stability for $(i, j) \in B$ for any $s \geq 2$. Furthermore, given $s \geq 3$, we observe that $C_{i,j}(t)$ drops faster than is required, when $Z_{i,j}(t) > 0$. We suspect that the higher speedup in our result is an outcome of our proof methodology[13], and likely, $s \geq 3$ is not strictly necessary. All the same, since we do not possess an analytical proof of the latter, we will settle for the sufficiency established in Theorem 9.

## 5.5.2 Alternative Approaches

### LOOFA: Comparison

Notice that SOQF matching looks similar but is markedly different from LOOFA [65], previewed in Sec. 3.2.5. In LOOFA, outputs are sorted (Fig. 3.8) using the *total* occupancy at each output link, i.e., $\sum_i X_{i,j}$ in our terminology. On the other hand, SOQF requires virtual input queueing at the outputs, and input-output *pairs* are sorted using the individual VIQ occupancy at each output link. The order in the sorted list in SOQF immediately determines the pairs to be connected. In contrast, LOOFA proceeds in sorted order of outputs, picking any input-output pair within those outputs. LOOFA might prefer pairs with a high $X_{i,j}$ within an output $j$ with low $\sum_i X_{i,j}$, while SOQF might serve pairs with a low $X_{i,j}$ within outputs with a high cumulative queue-length.

Consequently, the performance of these two algorithms end up being significantly different. LOOFA ensures that the CIOQ switch is work-conserving, for $s \geq 2$. In terms of stability, this implies $f_3$ equivalence with an OQ switch (for which any maximal matching suffices). For

[13]Could we envision an iterative game-theoretic proof with a Nash equilibrium at $s = 2$? This is an intriguing thought, yet beyond the current expertise of the author.

emulation of an OQ switch with FIFO schedulers at the output links, nothing additional is necessary. However, LOOFA does not ensure stability for specific input-output pairs, under flow-based output schedulers. SOQF matching, in comparison, does not ensure work-conservation, yet meets the stability goal, for $s \geq 3$. Specifically, SOQF does not guarantee that an output occupancy ($\sum_i X_{i,j}$) exceeds the input thread of each cell destined to that output, a property required for work-conservation, but instead guarantees property (5.28), which leads to strict relative stability.

The authors of LOOFA suggest that the order in which inputs are selected within the sorted output list would determine the fairness in the distribution of link bandwidth to input-output pairs. Accordingly, we explored the possibility of combining LOOFA with SOQF as follows. The arbiter first sorts the outputs, ensuring work-conservation for $s \geq 2$, then chooses the pair with the smallest $X_{i,j}$, within output $j$. Then $T_{i,j}$ in (5.27) may be redefined as follows:

$$T_{i,j}(n) = \sum_{\{(i,k):\forall l, X_{i,k}(n) \leq X_{l,k}(n) \text{ and } \sum_l X_{l,k} \leq \sum_l X_{l,j}\}} Q_{i,k}(n) + \sum_{\{(k,j):X_{k,j}(n) \leq X_{i,j}(n)\}} Q_{k,j}(n)$$

Atleast one of the queues in the summations is guaranteed a service turn whenever $Q_{i,j} > 0$. For $(i,j) \in B$, the stability of $X_{i,j}$ would immediately imply $\bar{X}_{k,j} = 0$ in the second sum. However, notice that the first sum does not depend on $X_{i,j}$. If both outputs $k$ and $j$ are oversubscribed, a pair $(i,k)$ in the first sum may refer to a flow with an unstable VIQ. Unfortunately, therefore, we were unsuccessful in applying the above property towards any stability result. We conjecture that the combined provision of work-conservation and strict relative stability is beyond the scope of a maximal matching for non-trivial values of $s$, i.e., for $s \neq f(N)$. The fairness property referred to by the authors of LOOFA likely stands for some form of max-min fairness[14] in the allocation of bandwidth, considering the input and output elements as separate switching nodes. Clearly, the latter allocation need not be the same as the one in the reference OQ switch.

### Heuristic Approximations

A coarse approximation of the sorting operation in SOQF is to partition the input-output pairs into two sets, a less congested set for which $X_{i,j}(n) \leq L$ for some finite $L$, and a congested set

[14]Max-min fairness has been a useful measure for studying the achievable bandwidth for a flow as it traverses a network of nodes. An end-to-end network may be considered max-min fair if the minimum achievable service rate for a flow, on its path, is maximized. We refrain from using such a measure *inside* a switch as it exposes the internal switching structure to network engineering.

**Algorithm 10.** *Maximal matching with backpressure*

```
Initialize:          1. For all $(i, j)$, $Q_{i,j} \leftarrow 0$
Cell Enqueue:        1. If cell belongs to $(i, j)$, increment $Q_{i,j}$

Every Timeslot $n$:  1. For all $(i, j)$, $G_{i,j} \leftarrow 1_{\{X_{i,j}<L\}} \cap 1_{\{Q_{i,j}>0\}}$
                     2. $\pi \leftarrow$ Maximal matching on $G$
                     3. $Q \leftarrow Q - \pi$
```

Figure 5.21: A backpressure-based approximation of SOQF

for which $X_{i,j}(n) > L$. The arbiter may then isolate congestion by finding a maximal matching, restricted to pairs in the first set, as shown in Fig. 5.21. In other words, the VIQ lengths are being used to apply *backpressure*, or feedback control, to the arbiter, for congested pairs. Note that the overall matching might not even be maximal, and hence the stability analysis of the previous section will cease to apply. Nevertheless, such backpressure-based schemes are already found in practice [19, 20, 16], especially suited for finite-sized memory elements. While an analytical characterization of backpressure is currently a popular topic, in the guise of buffered crossbar switching, the above older works have long since demonstrated the merit, primarily using simulation studies on restricted arrivals, of using such an approach for isolating congestion.

## 5.6 Putting it Together: Switched Fair-Airport Policies

In the previous sections, we showed how several online matchings, combined with rate-shaping, may be used to provide bandwidth and delay guarantees, specified by a reservation matrix $R$, independent of the offered arrivals. We also established how various matchings may be used to optimize throughput, at different levels of functional equivalence with a reference OQ switch. In general, the task is to satisfy both goals simultaneously. We introduced the concept of integrated matchings in Sec. 3.3.1, where two matching algorithms, $M_1$ and $M_2$, are used in conjunction with each other to satisfy multiple goals. We now apply that concept to the results shown in this chapter.

Let $M_1$ be the policy chosen to provide QoS guarantees, in order to satisfy $R$, with speedup $s_1$. $M_1$ may be based on offline templates using a BVN decomposition of $R$ ($s_1 = 1$), or a batch-mode matching operating on guaranteed-queue (GQ) counters, $\Pi_{i,j}$ for each pair $(i, j)$, assembled by the shaper at each epoch $kM$ as:

$$\Pi_{i,j}(kM) = \min(R_{i,j}M, Q_{i,j}(kM)),$$

where $1/M$ is the GCD of the rates, and $Q_{i,j}$ is the respective VOQ length. The matching may be maximal ($s_1 = 2$), critical ($s_1 = 1$), or SPS ($s_1 = 1$), the latter only applicable when $R_{i,j} \leq 1/N$, $\forall(i, j)$. Each of these online policies ensures a bandwidth trunk of $R_{i,j}$ and a worst-case latency of $2M$ to each pair, for the specified speedup values. If $R$ is inadmissible, request pruning is performed on $\Pi$, prior to matching, as described in Sec. 5.4.2.

Let $M_2$ be the policy chosen to optimize throughput, with speedup $s_2$. This matching operates directly on the VOQ lengths, $Q_{i,j}$ for each pair $(i, j)$, adding to the service turns given by $M_1$. $M_2$ may be maximal ($s = 2$ for $f_3$), critical, MWM, LPF ($s = 1$ for $f_2$), SOQF ($s = 3$ for $f_4$), or SPS ($s = 1$ for $f_2$), the latter only applicable for uniform arrivals. We refer to the integrated policy $(M_1, M_2)$ as a *switched fair-airport* (SFA) algorithm, as this may be viewed as a two-dimensional variant of the fair-airport link scheduler (Sec. 3.3.3). The precise method of integration depends on the relative properties of the two matchings. If hierarchical scheduling is implemented, as described in Sec. 3.3.2, in order to distribute the service turns to individual flows, then the connections made by $M_1$ are consumed by the respective GBS in the input elements, and those made by $M_2$ are consumed by EBS.

### 5.6.1 Multi-phase Combination

Recall that a matching $M_i$ is additive with respect to $M_j$, if $M_i$ can start with the configuration $\pi$ generated by $M_j$ in each timeslot, and generate a new configuration $\pi'$, which satisfies the properties of $M_j$, by merely adding connections to $\pi$, i.e., $\pi_{i,j} = 1$ implies $\pi'_{i,j} = 1$. For example, let $M_1$ be a batch-mode critical matching operating on $\Pi$. Let $M_2$ be a maximal matching that adds connections, operating on $Q$. Clearly, since $\forall(i, j)$, $\Pi_{i,j} > 0$ also implies $Q_{i,j} > 0$, the overall matching is guaranteed to be maximal. On the other hand, if $M_2$ is a critical matching,

**Algorithm 11.** *SFA-multiphase matching*

```
Initialize:            1. For all $(i,j)$, $Q_{i,j} \leftarrow 0$, $\Pi_{i,j} \leftarrow 0$
                       2. $M \leftarrow 1/\text{GCD}(R)$

Cell Enqueue:          1. If cell belongs to $(i,j)$, increment $Q_{i,j}$

Every Epoch, $n = kM$: 1. For all $(i,j)$, $\Pi_{i,j} = \min(R_{i,j}M, Q_{i,j})$
                       2. Prune $\Pi$, if necessary (Fig. 5.14, step 1)

Every Timeslot $n$:    1. For all $(i,j)$, $G_{i,j} \leftarrow 1_{\{Q_{i,j}>0\}}$
                       2. For all $(i,j)$, $G'_{i,j} \leftarrow 1_{\{\Pi_{i,j}>0\}}$
                       3. $\pi \leftarrow$ Apply $M_1$ on $G'$
                       4. $\pi \leftarrow$ Augment $\pi$ using $M_2$ on $G$
                       5. For all $(i,j)$, if $(\Pi_{i,j} \cap \pi_{i,j})$, decrement $\Pi_{i,j}$
                       6. $Q \leftarrow Q - \pi$ (dispatch cell)
```

Figure 5.22: SFA policy using a multi-phase combination of matchings

the connections already generated by $M_1$ may, in general, conflict with the goal of covering all critical lines of $Q$. This is because a critical line of $\Pi$ need not be a critical line of $Q$, and vice-versa. The combined matching, therefore, may not remain critical with respect to the VOQ lengths.

The advantage of additive matchings is that the configuration generated in a timeslot satisfies the properties of both $M_1$ and $M_2$, thereby minimizing the total required speedup to $s = \max(s_1, s_2)$, for simultaneously satisfying the performance goals of both. This integrated matching is referred to as a multi-phase combination, illustrated in Fig. 5.22. For our purpose, notice that there is no $M_1$ that is additive to any given $M_2$, primarily because $Q_{i,j} > 0$ need not imply $\Pi_{i,j} > 0$. Therefore, $M_1$ is always used in the first phase, and an $M_2$, if additive, augments the configuration in the second phase. In the policies seen so far, a general maximal matching is additive to any $M_1$. An SPS matching may be additive with another SPS, if the two sequences are aligned, in which case it reduces to a single matching. Notice that, in multi-phase combinations,

**Algorithm 12.** *SFA-exclusive matching*

```
Initialize:                       1. For all $(i, j)$, $Q_{i,j} \leftarrow 0$, $\Pi_{i,j} \leftarrow 0$
                                  2. $M \leftarrow 1/\text{GCD}(R)$, $T \leftarrow s_1 + s_2$

Cell Enqueue:                     1. If cell belongs to $(i, j)$, increment $Q_{i,j}$

Every Epoch, $n = kM$:            1. For all $(i, j)$, $\Pi_{i,j} = \min(R_{i,j}M, Q_{i,j})$
                                  2. Prune $\Pi$, if necessary (Fig. 5.14, step 1)

Every Timeslot $n$                1. For all $(i, j)$, $G_{i,j} \leftarrow 1_{\{\Pi_{i,j}>0\}}$
s.t. $kT \leq n < kT + s_1$:      2. $\pi \leftarrow$ Apply $M_1$ on $G$
                                  3. $\Pi \leftarrow \Pi - \pi$
                                  4. $Q \leftarrow Q - \pi$ (dispatch cell)

Every Timeslot $n$                1. For all $(i, j)$, $G_{i,j} \leftarrow 1_{\{Q_{i,j}>0\}}$
s.t. $kT + s_1 \leq n < kT + s_2$: 2. $\pi \leftarrow$ Apply $M_2$ on $G$
                                  3. $Q \leftarrow Q - \pi$ (dispatch cell)
```

Figure 5.23: SFA policy using an exclusive combination of matchings

the guaranteed queues (or the GBS) are given strict priority in the consumption of each service turn.

### 5.6.2 Exclusive Combination

If $M_2$, chosen to optimize throughput, is not additive to $M_1$, chosen to satisfy the reservation matrix $R$, then each policy must necessarily ignore the presence of the other, and operate independently for the simultaneous satisfaction of the goals. Consequently, the required speedup becomes $s = s_1 + s_2$. This integrated matching is an exclusive combination, illustrated in Fig. 5.23 (for integral values of $s_1$ and $s_2$), in which either $M_1$ or $M_2$ operates in each timeslot. Let $T$ be an integer such that both $s_1T$ and $s_2T$ are integers. Since, in $(s_1 + s_2)T$ consecutive timeslots, $\Pi$ receives at least as many service turns as it would under $M_1$ in $s_1T$ timeslots, and $Q$ receives

| $M_2$ \ $M_1$ | **BVN** $s \geq 1$ | **SPS** $s \geq 1$, $R_{i,j} \leq 1/N$ | **Maximal** $s \geq 2$ | **Critical** $s \geq 1$ |
|---|---|---|---|---|
| **MWM/LPF** $s \geq 1, f_2$ | 2 | 2 | 3 | 2 |
| **Critical** $s \geq 1, f_2$ | 2 | 2 | 3 | 2 |
| **SPS** $s \geq 1, f_2$ (Uniform $\lambda$) | 2 | 1 | 3 | 2 |
| **Maximal** $s \geq 2, f_3$ | 2 | 2 | 2 | 2 |
| **SOQF** $s \geq 3, f_4$ | 4 | 4 | 5 | 4 |

Table 5.1: SFA: Speedup required to simultaneously provide QoS and optimal throughput

at least as many turns as under $M_2$ in $s_2T$ timeslots, both the goals are satisfied.

In the policies seen so far, critical and SOQF matchings are not additive, in general, to any $M_1$. Neither are matchings such as MWM and LPF in the literature. Consequently, the benefit of the lower speedup required by some of these policies, for throughput and QoS, individually, is annulled if they need to be integrated with other matchings. This is true even if both the matchings belong to the same policy, e.g., a critical matching on $\Pi$ for QoS ($s = 1$), integrated with a critical matching on $Q$ for throughput ($s = 1$), will need an overall speedup of 2. This is no better than a multi-phase combination of any QoS-enabling matching, seen so far, with $s \leq 2$, and a maximal matching on $Q$ for throughput ($s = 2$).

Table 5.1 shows the speedup required for various SFA combinations, for the simultaneous satisfaction of QoS and optimal throughput. The individual speedup values are based on the results in this chapter, and those reviewed from the literature. For known matching algorithms, other than the restrictive case of (SPS, SPS), such combinations need a speedup of at least 2. Consequently, for the above goal, a practical maximal matching for $M_2$ is as good as any other policy, due to its additive nature. We conclude by noting that the alternative to such integrated matchings is the stable matching policy that exactly emulates an OQ switch.

## 5.7 Extensions to Multicast Traffic

While the primary focus of this work is on unicast traffic, several of the results seen so far, for admissible traffic, find direct application to a class of multicast scheduling schemes. There are two main types of multicasting techniques in the input-queueing literature. The first, and the more common, technique is to replicate cells in the space element. More precisely, each cell is associated with a set of destinations, referred to as its *fanout*. To serve a cell at the head of a queue in input element $i$, the input is connected to several outputs in the same timeslot, corresponding to a subset of the fanout $f(i)$. In other words, for $k \subseteq f(i)$, $\pi_{i,k} = 1$. The conflict-free property, for space-element multicasting, is merely $\sum_i \pi_{i,j} \leq 1$, for all outputs $j$. Conflicts are allowed at the inputs, i.e., $\sum_j \pi_{i,j}$ may be greater than 1, as long as the same cell consumes all the connections. The WBA algorithm [90] is one of the first such policies, which employs a single queue of multicast cells in each input element.

It has long been recognized [41, 75] that such a scheme suffers from significant head-of-line blocking. To alleviate this condition, a commonly used technique (e.g., [69, 42]) is to maintain unicast cells in their respective VOQ, and employ a small number of dedicated queues for multicast cells. Cells are hashed to specific queues on arrival. To completely eliminate blocking, however, we need a prohibitive $2^N - 1$ multicast queues [76], one for each possible fanout set. Determining the tradeoff between the number of queues and the observed throughput for various traffic and fanout patterns has been a fertile playground in switching research. Nevertheless, most practical schemes that utilize space-element replication suffer from less than 100% throughput, and multicast is essentially treated as second-class traffic for throughput and QoS purposes.

An alternative approach, originally due to Turner [112] utilizes the concept of *recycling*, in which each input element is used as a relay point for copying cells. This technique does not employ space-element replication, and suffers from high delays, yet most of the results for unicast throughput and QoS may now be applied to multicast traffic.

### Recycling

Recycling uses a modified pipeline for the forwarding path, as shown in Fig. 5.24. The input elements do not employ special multicast queues, and contain the same $N$ virtual output queues,

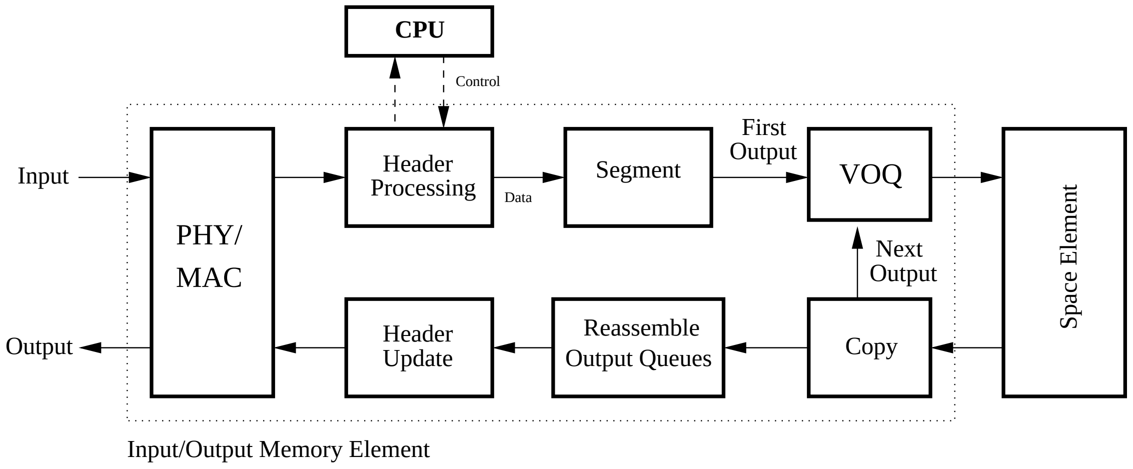


Figure 5.24: Forwarding-path pipeline for multicast recycling support

originally meant for unicast traffic. A destination list is created for each cell (or for each fine-grain flow) using its fanout set. When a cell first arrives, it is enqueued in precisely a single VOQ, corresponding to the first output in its destination list. On arriving at the output element, if there are additional destinations for the cell, a single copy of the cell is made, destined to the next output in the list, and is injected into the ingress pipeline. If the fanout size is $F$, then all the destinations receive a copy in $F$ traversals of the space element. The multiple number of traversals result in a high delay. In addition, it is evident that the memory bandwidth required in the VOQ system is $(1 + 2s)C$, where $s$ is the internal speedup, and $C$ the external link capacity.

The main issue with multicast traffic is that admissibility does not imply both the conditions in (2.2). While the output constraint remains, i.e, $\forall j$, $\sum_i \lambda_{i,j} \leq C$, there is effectively no input constraint. Even though the arrival rate at each input cannot exceed $C$, each cell represents several copies, yielding a higher effective arrival rate. Recycling reins in the arrival process at the virtual output queues. Notice that the VOQ system observes a unicast stream from the input interface, with a total rate of no greater than $C$, and an additional stream from the egress pipeline, which also cannot be greater than $C$ due to the output constraint. Therefore the effective arrivals $\lambda$ into the VOQ system satisfies:

$$\forall i \sum_j \lambda_{i,j} \leq 2C \quad \text{and} \quad \forall j \sum_i \lambda_{i,j} \leq C. \tag{5.33}$$

**Throughput and QoS**

The multicast admissibility condition implies that most of the matching results for unicast traffic will yield similar results for multicast traffic, with an additional speedup of 1. Specifically, the proof of Theorem 4 can be modified to account for (5.33) by simply changing (5.16) to

$$\bar{\mathcal{Z}}(t) \leq 2t - \bar{D}_{\mathcal{Z}}(t).$$

Consequently, for $s \geq 2$, a critical matching provides 100% throughput to admissible multicast arrivals. Similarly, the Dai-Prabhakar maximal-matching result can be modified by recognizing that $\bar{C}'_{i,j}(t) \leq 0$ in (3.10) for $s \geq 3$. This gives us two corollaries (below) for multicast traffic. Unfortunately, the results for inadmissible traffic (Theorems 8 and 9) do not apply because the input constraint in (5.33) ceases to hold. In fact, if a cell is recycled from an oversubscribed output, it spreads the instability to the corresponding input VOQ system.

**Corollary 6.** *A critical matching, with recycling, is sufficient for asymptotic 100% throughput in an input-queued switch, for admissible multicast arrivals, as long as the speedup $s \geq 2$.*

**Corollary 7.** *Any maximal matching, with recycling, is sufficient for asymptotic 100% throughput in an input-queued switch, for admissible multicast arrivals, as long as the speedup $s \geq 3$.*

Similarly, maximal and critical matchings may be used in batch-mode to either provide exact 100% throughput to bounded multicast traffic, or ensure bandwidth guarantees, combined with rate shaping. A multicast cell with a fanout $F$ finds itself in a sequence of $F$ batches, leading to a worst-case latency of $F + 1$. The delay bound for leaky-bucket constrained traffic (5.7) becomes

$$D_{i,j} \leq \frac{\sigma_{i,j}}{R_{i,j}} + \frac{F+1}{r}.$$

Without further elaboration, we state the following corollaries of Theorems 1 and 3.

**Corollary 8.** *Any maximal matching, combined with rate shaping and recycling, is sufficient to provide isolated bandwidth guarantees to input-output pairs of an input-queued switch, for multicast traffic, as long as the internal speedup $s \geq 3 - r$.*

**Corollary 9.** *A critical matching, combined with rate shaping and recycling, is sufficient to provide isolated bandwidth guarantees in an input-queued switch, for multicast traffic, as long as the speedup $s \geq 2$.*

## 5.8 Summary

We addressed the performance of matching algorithms for input-queued switches, the starting point of the BCS taxonomy, with the dual goals of providing bandwidth and delay guarantees, and optimizing throughput, based on the framework of functional equivalence with an OQ switch. While input-queued switches are already well represented in the literature, we established several new results, some fundamental, on stronger performance guarantees for well-known matching policies.

We demonstrated that *rate-shaping*, combined with an online maximal or critical matching (Theorems 1, 2, 3) may be used to provide bandwidth guarantees in an input-queued switch, independent of the offered traffic. These are extensions of the Weller-Hajek theorems on C-MSM and maximal matching for bounded traffic, and may be preferred to template-based schemes in the literature that are associated with a high storage overhead. Our proof methodology analytically established the analogy between Clos fitting in space and maximal matching in time. The latter may be viewed as a strategy for Clos fitting, to eliminate backtracking in time. We derived delay bounds for leaky-bucket constrained traffic for such batch-mode matchings, and showed how shaping may be implemented using simple counters. A method called *request pruning* was introduced for applying such matchings to an oversubscribed reservation, so as to guarantee rates (Theorem 7), while allowing for statistical multiplexing in overloaded outputs.

For admissible traffic, we proved (Theorem 4), using the fluid-limit theorem, that a *critical* matching suffices for 100% throughput, without speedup. For the same capability, the best results in the literature are for more complex policies, including LPF and maximum weight matching. We answered the long-standing issue of whether 100% throughput can be provided by a general class of maximum-size matchings by reporting that less is needed. The cardinality of a matching, while beneficial for instantaneous throughput, turns out to be an irrelevant property for long-term throughput. For the latter, a matching merely needs to cover all the critical lines in each timeslot. Next, we showed that uniform arrivals are so benign that a pre-determined *perfect sequence* suffices for 100% throughput, without speedup (Theorem 5). The matching need not even be maximal, and most popular algorithms such as iSLIP and dual round-robin turn out to be overkill for such restricted arrivals. An $O(1)$ parallel EREW matching, with staggered pointers, generates (Theorem 6) such a perfect sequence and is sufficient for optimal throughput to uniform, including bursty, arrival traffic.

We next made an unprecedented foray into throughput optimization for inadmissible traffic. Our first result, for such traffic, extended the Dai-Prabhakar maximal-matching theorem, so far restricted to admissible traffic. We showed (Theorem 8) that a maximal matching, with a speedup no less than 2, suffices to isolate instability to oversubscribed outputs and provide 100% throughput to the rest of the flows. This obviates the add-on heuristics we see in the literature for preventing starvation in the presence of hot-spot traffic. While any work-conserving scheme, such as LOOFA, also provides the same wide-sense relative stability with an OQ switch, this result holds for much simpler matchings. We then introduced the *shortest output-queue first* maximal matching to provide 100% throughput (Theorem 9) to the same set of flows as in a reference OQ switch, i.e., strict relative stability, for a broad class of link scheduling schemes. The only algorithms in the literature that exceed such a capability are the impractical stable matching policies that exactly emulate the reference switch.

Finally, we showed (Sec. 5.6) how two policies, one for QoS and the other for optimal throughput, may be integrated into a *switched fair-airport* combination. We noted that maximal matching ends up as the best choice for implementation, due to its simplicity and its suitability for integrated matchings without increasing the speedup requirement. Such combinations fit well in a hierarchical scheduling framework, typically used (Sec. 3.3.2) to serve fine-grain flows, if required. We concluded by showing (Sec. 5.7) that all the reported results for admissible unicast traffic may be directly applied to multicast traffic as well, with an additional speedup of 1, for a class of multicasting techniques that utilize cell *recycling* in the memory elements.

# Chapter 6

# CIOQ: Aggregation and Pipelining

Our next focus is on the remaining single-path buffered Clos switches, specifically CIOQ with aggregation (CIOQ-A), CIOQ with pipelining (CIOQ-P), and general memory-space-memory (G-MSM) switches. Such designs, especially G-MSM, are already found in commercial[1] use [19, 21, 22], and in the literature [85, 93, 92, 101]. While less-studied than the CIOQ switch, the general approach so far has been to propose and analyze matching algorithms for such switches independently of CIOQ switching. We demonstrate in this chapter that such stand-alone treatment is unnecessary, and the rich results of CIOQ may be directly applied to all single-path BCS. These turn out to be straight-forward extensions of CIOQ, and a careful emulation of the latter suffices to translate its results to single-path buffered Clos switches.

We show that *shadowing* a CIOQ followed by decomposition in CIOQ-A, and by a sequential dispatch or striping in CIOQ-P, is sufficient to exactly emulate a high-performance CIOQ switch and thereby inherit its QoS and throughput performance. For lower complexity, the same matching algorithms as in CIOQ may be used with a different queueing structure in CIOQ-A, and with an equal dispatch method in CIOQ-P, to inherit most of the performance results. A G-MSM employs a natural combination of the methods used individually by CIOQ-A and CIOQ-P. We believe that these findings eliminate the need to individually analyze each item in single-path BCS.

[1]The author was one of the architects of the Atlanta/$\pi$ switching chipsets at Bell Labs in the 1995-2000 timeframe. Most of the analytical results here are inspired by that work and the copious amounts of simulation studies done as part of their performance evaluation.

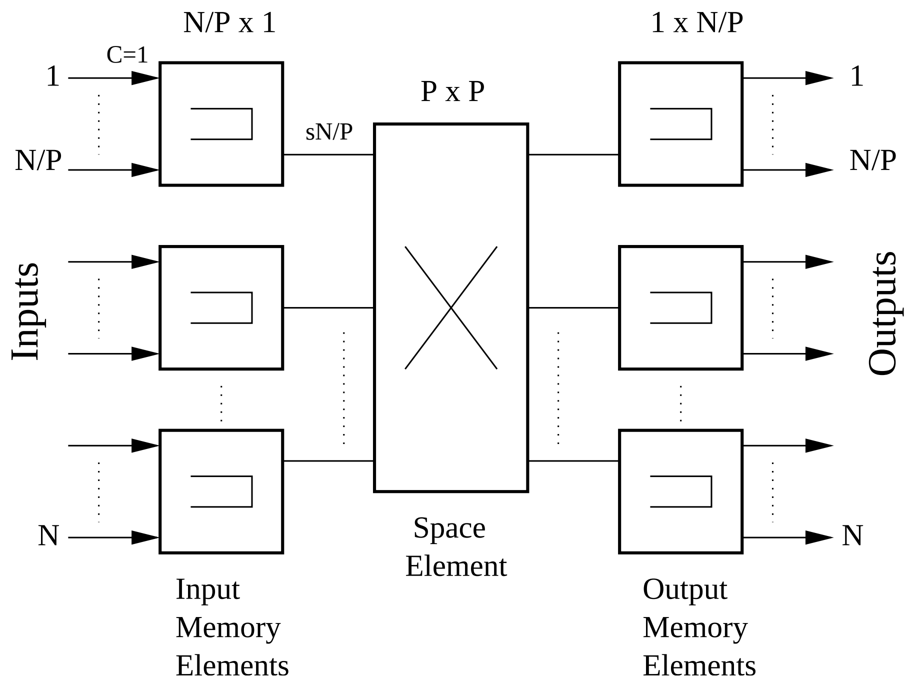


Figure 6.1: CIOQ-A: CIOQ switch with aggregation

## 6.1 Aggregation

Fig. 6.1 shows a $N \times N$ CIOQ-A switch $(N, [MSM], P, 1, s)$. We obtain such a switch by aggregating $N/P$ consecutive interfaces of a CIOQ switch into the same memory element. Consequently, there are $P$ first-stage memory elements of dimension $N/P \times 1$, a single $P \times P$ space element in the center, and $P$ instances of $1 \times N/P$ third-stage memory elements. We assume that $N/P$ is an integer and $1 < P < N$.

We refer to the time required to transfer a cell on the external links as an *external timeslot*. As in a CIOQ switch, this equals $L/C$, where $L$ is the length of a cell, and $C$ is the link capacity. Normalized to an external capacity of one cell per timeslot, the internal links operate at $sN/P$, where $s$ is the internal speedup. In other words, there are $sN/P$ *internal timeslots* for the space element, for every external timeslot. Clearly, $s \geq 1$ is required to sustain the offered traffic. Recall (Sec. 4.1.2) that the memory bandwidth of $(1+s)\frac{N}{P}$ and a space-element arbitration frequency of $s\frac{N}{P}$ matchings per external timeslot constitute the drawbacks of such a design with respect to a

$N \times N$ CIOQ switch. The benefits include a smaller albeit faster space element, especially suited to currently available optical components, support for subports, and possibly a lower matching complexity, depending upon the queueing structure.

### 6.1.1 Queueing Strategies

We introduce three distinct queueing strategies for the first-stage elements, as shown in Fig. 6.2. In *virtual element queueing* (VEQ), cells are organized per output element. Specifically, there are $P$ queues in every input element, one for each output element. A cell belonging to input-output flow $(i, j)$ arrives at element $\lceil \frac{i}{N/P} \rceil$, and is enqueued to VEQ $\lceil \frac{j}{N/P} \rceil$, irrespective of the specific input and output ports. Let the $N \times N$ discrete-time matrix $A^*(n)$ denote the cumulative arrivals for each pair until external timeslot $n$, with average rates $\lambda^*$. The evolution of the VEQ system may then be described by $P \times P$ matrices $(A(n), D(n), Q(n))$, denoting the cumulative arrivals, cumulative departures and current queue-length, respectively. Let $\lambda$ be the average rates, associated with $A(n)$, for aggregate flows belonging to each input-output *element* pair. Then, $(A^*, \lambda^*)$ is related to $(A, \lambda)$ as:

$$\begin{aligned} \forall n, \forall i \leq P, \forall j \leq P, \quad A_{i,j}(n) &= \sum_{k=\frac{N}{P}(i-1)+1}^{k \leq \frac{N}{P} i} \sum_{l=\frac{N}{P}(j-1)+1}^{l \leq \frac{N}{P} j} A^*_{k,l}(n) \\ \forall i \leq P, \forall j \leq P, \quad \lambda_{i,j} &= \sum_{k=\frac{N}{P}(i-1)+1}^{k \leq \frac{N}{P} i} \sum_{l=\frac{N}{P}(j-1)+1}^{l \leq \frac{N}{P} j} \lambda^*_{k,l} \end{aligned} \tag{6.1}$$

Clearly, if the average rates are admissible on the external links (2.2), i.e., if $\forall i \leq N$, $\sum_j \lambda^*_{i,j} \leq 1$, and $\forall j \leq N$, $\sum_i \lambda^*_{i,j} \leq 1$, then

$$\forall i \leq P, \ \sum_{j=1}^{P} \lambda_{i,j} \leq \frac{N}{P} \quad \text{and} \quad \forall j \leq P, \ \sum_{i=1}^{P} \lambda_{i,j} \leq \frac{N}{P}. \tag{6.2}$$

The arbiter operates on $Q(n)$, and finds a matching $\pi(k)$ in each internal timeslot $k$. Since $Q$ is a $P \times P$ matrix, in general, the matching complexity on VEQ is lower than the corresponding one in a CIOQ switch. As there are $sN/P$ internal slots for each external one, the VEQ evolution is described by the following:

$$\forall n, \quad Q(n) = A(n) - D(n), \tag{6.3}$$

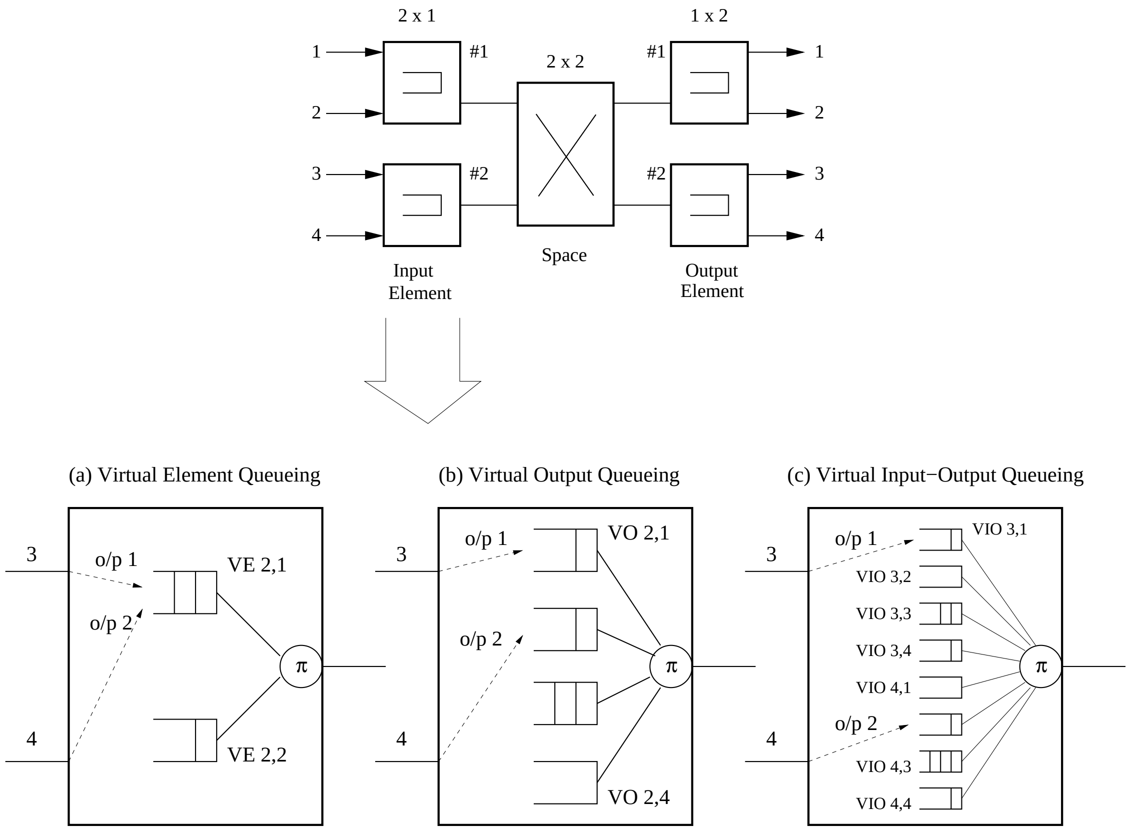


Figure 6.2: Different queueing strategies for CIOQ-A switches

$$D(n) \quad = \quad \sum_{k=1}^{s\frac{N}{P}n} \pi(k). \tag{6.4}$$

In *virtual output queueing* (VOQ), shown in Fig. 6.2(b), cells are organized per output port. There are $N$ queues in every input element, one for each output port. A cell belonging to flow $(i, j)$ is enqueued to VOQ $j$ in input-element $\lceil \frac{i}{N/P} \rceil$, irrespective of the specific input port. The evolution of the VOQ system is described by $P \times N$ matrices $(A(n), D(n), Q(n))$, with $\lambda$ denoting the average rates associated with $A(n)$. Then $(A^*, \lambda^*)$ is related to $(A, \lambda)$ as:

$$\forall n, \forall i \le P, \forall j \le N, \quad A_{i,j}(n) \quad = \quad \sum_{k=\frac{N}{P}(i-1)+1}^{k \le \frac{N}{P} i} A^*_{k,j}(n)$$

$$\forall i \leq P, \forall j \leq N, \quad \lambda_{i,j} \quad = \quad \sum_{k=\frac{N}{P}(i-1)+1}^{k \leq \frac{N}{P} i} \lambda^*_{k,j}$$

If the average rates $\lambda^*$ are admissible on the external links, then

$$\forall i \leq P, \ \sum_{j=1}^{N} \lambda_{i,j} \leq \frac{N}{P} \quad \text{and} \quad \forall j \leq N, \ \sum_{i=1}^{P} \lambda_{i,j} \leq 1. \tag{6.5}$$

The arbiter operates on the VOQ matrix $Q(n)$ to compute a $P \times P$ matching $\pi(k)$ in each internal timeslot $k$. Depending on how the $P \times N$ queue-state is reduced to a matching of smaller dimensions, the complexity may or may not be lower than in a CIOQ switch. For the algorithms considered in this chapter, matching on such a queueing structure yielded no benefits in performance with respect to the less complex VEQ. Nevertheless, for posterity, we note here that the VOQ evolution is described by (6.3) and a modification of (6.4), shown below, the latter because a service turn given to element pair $(i, j)$ is distributed to some output $k$, where $\frac{N}{P}(j-1) < k \leq \frac{N}{P} j$, within input element $i$.

$$\forall n, i, j, \quad \sum_{l=\frac{N}{P}(j-1)+1}^{\frac{N}{P} j} D_{i,l}(n) \quad = \quad \sum_{k=1}^{s \frac{N}{P} n} \pi_{i,j}(k) \tag{6.6}$$

Finally, in *virtual input-output queueing* (VIOQ), shown in Fig. 6.2(c), the structure exactly resembles that of a CIOQ. Cells are organized separately per input-output port pair. Each input element contains $\frac{N^2}{P}$ queues, for the $\frac{N}{P}$ inputs to the element and $N$ output ports, for a total of $N^2$ queues in the first-stage. The evolution of the VIOQ system is described by $N \times N$ matrices $(A(n), D(n), Q(n))$, with $\lambda$ denoting the average rates associated with $A(n)$, all of which exactly correspond to the respective matrices in a $N \times N$ CIOQ switch. In other words, $(A, \lambda)$ is the same as $(A^*, \lambda^*)$ for VIOQ. The arbiter operates on the $N \times N$ matrix $Q(n)$ to compute a $P \times P$ matching $\pi(k)$ in each internal timeslot. Due to the dimensions of $Q$, we do not expect the matching complexity to be any lower than in a CIOQ switch. Each service turn given to element pair $(i, j)$ is distributed to port pair $(k, l)$, where $k$ is an input to first-stage element $i$, and $l$ is an output of third-stage element $j$. Therefore, the VIOQ evolution may be described by (6.3) and the following modification of (6.4):

$$\forall n, i, j, \quad \sum_{k=\frac{N}{P}(i-1)+1}^{\frac{N}{P} i} \ \sum_{l=\frac{N}{P}(j-1)+1}^{\frac{N}{P} j} D_{k,l}(n) \quad = \quad \sum_{k=1}^{s \frac{N}{P} n} \pi_{i,j}(k) \tag{6.7}$$

Notice that cells do not experience head-of-line blocking in any of the queueing structures. The choice then would be determined by a tradeoff between matching complexity and performance. The output elements may independently employ any queueing strategy, unless a specific organization is mandated by the matching algorithm (such as, e.g., SOQF). At a minimum, cells are organized per output port, within each output element. As in the previous chapter, we note, without elaboration, that the service turns given to each coarse-grain VEQ, VOQ, or VIOQ may be translated to per-flow service using virtual schedulers in a hierarchical fashion, as described in Sec. 3.3.2.

### 6.1.2 Shadowing a CIOQ

We first introduce a method called *shadowing* in order to exactly emulate a CIOQ switch and inherit its performance. Consider a $N \times N$ CIOQ-A switch $(N, [MSM], P, 1, s)$. Let a $N \times N$ CIOQ switch, employing a set of matching algorithms $\mathcal{A}$, be the reference switch. Let $s^*$ be the internal speedup required to meet the desired QoS and/or throughput performance with $\mathcal{A}$, in the reference. Here, $\mathcal{A}$ may contain an algorithm such as critical ($s^* \geq 1$), SPS ($s^* \geq 1$), maximal ($s^* \geq 2$), or SOQF ($s^* \geq 3$) matching for throughput optimization ($f_2, f_2$ for uniform traffic, $f_3$ and $f_4$ equivalence with an OQ switch, respectively), or rate-shaping (Sec. 5.1.3) combined with critical ($s^* \geq 1$), SPS ($s^* \geq 1$), or maximal ($s^* \geq 2$) matching for bandwidth and delay guarantees, or an SFA combination of both, with associated internal speedup as shown in Table. 5.1.

For exact emulation, the CIOQ-A switch employs virtual input-output queueing. The arbiter, under shadow-$\mathcal{A}$, first computes a matching $\pi^*$ by running $\mathcal{A}$ on the VIOQ state matrix $Q$, at a frequency of $s^*$ per external slot. If rate-shaping is required, $Q$ is used to derive a batch matrix $\Pi$ (5.6) using the requested rates $R$, and the computation is instead (or additionally, for SFA) applied on $\Pi$. Clearly, if a cell is dispatched, from the first to the third stage, for every connected pair in $\pi^*$ before the next computation, then the space element exactly emulates the one in the reference switch. The link schedulers at the outputs are chosen to exactly match the one in the reference. Consequently, the queue states in the two systems become indistinguishable, except in the interval between two consecutive matchings.

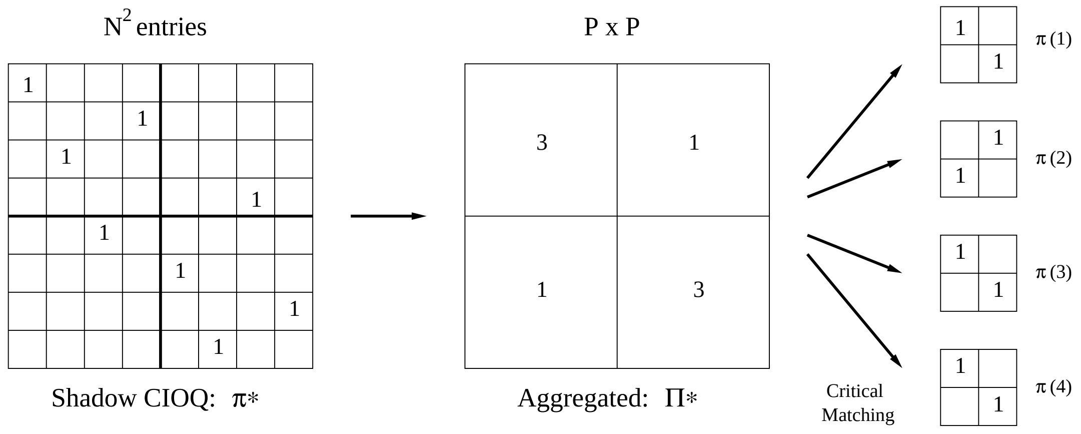


Figure 6.3: CIOQ-A: Emulating a CIOQ using shadowing and decomposition

To ensure that a cell is dispatched through the $P \times P$ space element for every connected pair in the $N \times N$ matching $\pi^*$, an aggregate matrix $\Pi^*$ is assembled as follows:

$$\forall i \leq P, \forall j \leq P, \quad \Pi^*_{i,j} = \sum_{k=\frac{N}{P}(i-1)+1}^{\frac{N}{P}i} \sum_{l=\frac{N}{P}(j-1)+1}^{\frac{N}{P}j} \pi^*_{k,l}. \tag{6.8}$$

In other words, $\Pi^*$ represents the aggregate number of connections for each input-output element pair. From (6.8), each line sum of $\Pi^*$ is no greater than $N/P$. We may then decompose the matrix into a sequence of $P \times P$ matchings $\pi$ using well-known matrix decomposition techniques. Specifically, a maximal matching decomposes $\Pi^*$ using a sequence of no greater than $2\frac{N}{P} - 1$ configurations (lemma 2), and a critical matching does the same using no more than $\frac{N}{P}$ configurations (lemma 5). Such a combination of shadowing and decomposition is illustrated in Fig. 6.3, for an $8 \times 8$ switch with $P = 2$.

When $\pi_{i,j}(k) = 1$ for some internal slot $k$, input element $i$ is connected to output element $j$, providing a path for exactly one connection in the respective quadrant of $\pi^*$. In practice, a cell may first be transferred from $Q_{i,j}$ to a temporary FIFO $Q'_{\lceil \frac{i}{N/P} \rceil, \lceil \frac{j}{N/P} \rceil}$ immediately when $\pi^*_{i,j} = 1$, and the matchings $\pi$ may subsequently be used to drain $Q'$. Due to its small size, upper bounded by $N/P$ cells, $Q'$ may be implemented using on-chip queues. An instance of the shadow-and-decompose method for the exact emulation of a CIOQ switch is illustrated in Fig. 6.4.

**Algorithm 13.** *CIOQ-A: Shadow and Decompose*

```
Initialize:                     1. For all $(i,j), i \le N, j \le N,\ Q_{i,j} \leftarrow 0$
                                2. For all $(i,j), i \le N, j \le N,\ \Pi_{i,j} \leftarrow 0$ (Batches for QoS)
                                3. For all $(i,j), i \le P, j \le P,\ Q'_{i,j} \leftarrow 0$ (Temporary FIFOs)
                                4. $M \leftarrow$ 1/GCD($R$), $T \leftarrow \frac{sN}{s^*P}$

Cell Enqueue:                   1. If cell belongs to $(i,j)$, increment $Q_{i,j}$
Every external epoch $n = kM$:  1. For all $(i,j), i \le N, j \le N,\ \Pi_{i,j} = \min(R_{i,j}M, Q_{i,j})$

Every internal epoch $n = kT$:  1. $\pi^* \leftarrow$ Apply $\mathcal{A}$ on $Q, \Pi$
                                2. For all $(i,j)$, $i \le N$, $j \le N$, with $\pi^*_{i,j} = 1$ do steps 2a, 2b
                                2a. Decrement $Q_{i,j}$
                                2b. Increment $Q'_{\lceil \frac{i}{N/P} \rceil, \lceil \frac{j}{N/P} \rceil}$

Every internal slot $n$:        1. $\pi \leftarrow$ Maximal/critical matching on $Q'$
                                2. $Q' \leftarrow Q' - \pi$ (Dispatch cells)
```

Figure 6.4: CIOQ-A: Emulating a CIOQ using shadowing and decomposition

**Theorem 10.** *A CIOQ-A switch, with virtual input-output queueing, exactly emulates a CIOQ switch with algorithm set $\mathcal{A}$ and speedup $s^*$, by shadowing $\mathcal{A}$ to compute matchings $\pi^*$, followed by a maximal or critical decomposition to exactly cover $\pi^*$, as long as the internal speedup $s \ge (2 - \frac{P}{N})s^*$ for maximal decomposition and $s \ge s^*$ for critical decomposition, i.e.,*

$$\text{(CIOQ-A, \{Shadow-}\mathcal{A}\text{, Maximal Decomposition\})} \overset{T,f_5}{\simeq} \text{(CIOQ, }\mathcal{A}\text{)},\quad s \ge (2-\tfrac{P}{N})s^*$$
$$\text{(CIOQ-A, \{Shadow-}\mathcal{A}\text{, Critical Decomposition\})} \overset{T,f_5}{\simeq} \text{(CIOQ, }\mathcal{A}\text{)},\quad s \ge s^*$$

*Proof.* The shadowing method computes an $N \times N$ matching $\pi^*$ at a rate of $s^*$ per external timeslot. Since there are $s\frac{N}{P}$ internal timeslots for every external one, the shadowing occurs at internal epochs $kT$, where $T = \frac{sN}{s^*P}$, $k \ge 1$. A maximal matching covers $\Pi^*(kT)$ in no greater than $2\frac{N}{P} - 1$ configurations. A cell is dispatched for every connection in $\Pi^*(kT)$ as long as the configurations are implemented before epoch $(k+1)T$, i.e., if

$$\frac{sN}{s^*P} \geq 2\frac{P}{N} - 1$$

This immediately yields $s \geq (2 - \frac{P}{N})s^*$ for maximal decompositions. In the same way, since $\Pi^*(kT)$ is covered in no more than $\frac{N}{P}$ configurations by a critical decomposition, $s \geq s^*$. $\square$

In other words, a CIOQ-A switch can exactly emulate a reference CIOQ switch, with the same speedup as the latter using critical matching to decompose $\Pi^*$, or with approximately twice the speedup[2] of the latter using maximal matching. The VIOQ departure matrix $D(n)$ is then governed by the following modification of (6.7):

$$\forall n, i \leq N, j \leq N, \qquad D_{i,j}(n) = \sum_{k=1}^{ns^*} \pi^*_{i,j}(k)$$
$$\forall n, i \leq P, j \leq P, \qquad \sum_{k=\frac{N}{P}(i-1)+1}^{\frac{N}{P}i} \; \sum_{l=\frac{N}{P}(j-1)+1}^{\frac{N}{P}j} \pi^*_{k,l}(n) = \sum_{k=(n-1)\frac{sN}{s^*P}+1}^{\frac{sN}{s^*P}n} \pi_{i,j}(k)$$

Nevertheless, an explicit characterization of $D(n)$ is not required to deduce the QoS and throughput performance of the switch. If $\mathcal{A}$ suffices in the reference CIOQ switch for functional equivalence with an OQ switch, at level $f_i$, with speedup $s^*$, then the CIOQ-A switch using shadowing and decomposition achieves the same level of equivalence with the OQ switch, with an internal speedup specified by Theorem 10. The containment (4.5) and transitivity (4.6) properties of equivalence lead us to the following straight-forward extension:

**Corollary 10.** *(CIOQ, $\mathcal{A}$) $\overset{T,f_i}{\simeq}$ (OQ, $\mathcal{A}'$), $s \geq s^*$ $\Rightarrow$*
*(CIOQ-A, {Shadow-$\mathcal{A}$, Maximal Decomposition}) $\overset{T,f_i}{\simeq}$ (OQ, $\mathcal{A}'$), $s \geq (2 - \frac{P}{N})s^*$, and*
*(CIOQ-A, {Shadow-$\mathcal{A}$, Critical Decomposition}) $\overset{T,f_i}{\simeq}$ (OQ, $\mathcal{A}'$), $s \geq s^*$*

This allows us to translate the performance results for a CIOQ switch, from the literature and Chapter 5, to a CIOQ-A switch. Table 6.1 shows the speedup $s$ required for various combinations of $\mathcal{A}$, and maximal or critical decomposition, for different levels of functional equivalence with a work-conserving OQ switch. The values of $s^*$ are derived from the result by McKeown et

[2]The speedup requirement becomes much less than twice that of the CIOQ switch as $P$ approaches $N$, i.e., as the CIOQ-A switch starts to look more like a CIOQ switch itself.

| Decomposition / $\mathcal{A}$ | **Critical** | **Maximal** | | |
|---|---|---|---|---|
| | | $\frac{N}{P}=2$ | $\frac{N}{P}=4$ | $\frac{N}{P}=8$ |
| **MWM/Critical** $s^* \geq 1, f_2$ | 1 | 1.5 | 1.75 | 1.88 |
| **SPS** $s^* \geq 1, f_2$ (Uniform $\lambda$) | 1 | 1.5 | 1.75 | 1.88 |
| **Maximal** $s^* \geq 2, f_3$ | 2 | 3 | 3.5 | 3.75 |
| **SOQF** $s^* \geq 3, f_4$ | 3 | 4.5 | 5.25 | 5.63 |

Table 6.1: Shadowing in CIOQ-A: Speedup required for throughput optimization

al. for MWM [80], and Theorems 4, 5, 8 and 9 for critical, SPS, maximal and SOQF matchings, respectively. Note that the value of $s$ for maximal decomposition depends on $N/P$.

The same values of internal speedup apply for critical, SPS and maximal matching on the batch matrix $\Pi$, for bandwidth guarantees. Since the queue states at internal slot $kT$, $\forall k$, in the CIOQ-A switch, and at slot $k$ in the reference CIOQ switch are indistinguishable for the same arrivals, the delay bounds in (5.7) and (5.8) continue to hold. For the combined provision of optimal throughput and QoS, $\mathcal{A}$ may correspond to an SFA combination. For critical decomposition, the speedup values remain the same as in the reference CIOQ switch, shown in Table 5.1. For maximal decomposition, the latter needs to be multiplied by the appropriate entry in the first row of Table 6.1 for the respective values of $N/P$. Clearly, the less complex maximal matching becomes impractical for decomposing more complex $\mathcal{A}$. For example, for $\mathcal{A} = \{\text{Maximal}, \text{SOQF}\}$, the required speedup for shadowing approaches a ridiculously prohibitive value of 10.

## Matching Complexity

The run-time complexity of each matching in a maximal decomposition is $O(P^2)$, and that in a critical decomposition is $O(P^{2.5})$. Since there are $s\frac{N}{P}$ internal slots for every external one, the decomposition runs in $O(NP)$ and $O(NP^{1.5})$ time, respectively, with respect to the external clock. If the algorithm set $\mathcal{A}$ runs in $O(\xi)$ time, the total complexity of the shadow-and-decompose method becomes $O(\xi + NP)$ and $O(\xi + NP^{1.5})$, respectively. For the parallel EREW implementation of a maximal matching, each decomposition step runs in $O(P)$ time, or $O(N)$ on the

| Decomposition / $\mathcal{A}$ | **Critical** | **Maximal** | **EREW Maximal** |
|---|---|---|---|
| **MWM** $\xi = N^3$ | $O(N^3)$ | $O(N^3)$ | $O(N^3)$ |
| **Critical** $\xi = N^{2.5}$ | $O(N^{2.5})$ | $O(N^{2.5})$ | $O(N^{2.5})$ |
| **SPS** $\xi = 1$ | $O(NP^{1.5})$ | $O(NP)$ | $O(N)$ |
| **Maximal** $\xi = N^2$ | $O(N^2 + NP^{1.5})$ | $O(N^2)$ | $O(N^2)$ |
| **EREW Maximal** $\xi = N$ | $O(NP^{1.5})$ | $O(NP)$ | $O(N)$ |
| **SOQF** $\xi = N^2 \log N$ | $O(N^2 \log N + NP^{1.5})$ | $O(N^2 \log N)$ | $O(N^2 \log N)$ |

Table 6.2: CIOQ-A: Arbitration complexity of the shadow-and-decompose method

external clock. As is evident, there is no benefit in arbitration complexity using this approach, inspite of the smaller space element. Table 6.2 shows the run-time complexity for various combinations.

Recall (Sec. 3.1) that Clos fitting or Slepian-Duguid (SD) fitting may also be used to partition $\Pi^*$ into $2\frac{N}{P} - 1$ or $\frac{N}{P}$ configurations, respectively. This is as expected due to the analogy of the two algorithms respectively to maximal and critical matching. Since there are $N$ entries to be fitted into $P \times P$ matrices, the former runs in $O(\frac{N^2}{P})$ time, while the latter runs in $O(NP^2 + \frac{N^2}{P})$. However, since these algorithms do not operate in timeslot sequence, all the matrices need to be pre-computed before they can be used to drain the connections represented in $\Pi^*$. Consequently, the decomposition is performed in two steps. First, $\Pi^*(kT)$ is decomposed into a set of matrices $\pi$ in $[kT, (k+1)T)$. The connections in $\Pi^*(kT)$ are subsequently drained in $[(k+1)T, (k+2)T)$. Hence, the departures emulate those in the reference CIOQ switch, with a lag of exactly one internal slot of the reference.

### Batch-mode Decomposition

We say that a switch $\mathcal{S}_2$ emulates a reference switch $\mathcal{S}_1$ with a constant lag of $\delta$ if all the cells that depart in $\mathcal{S}_1$ by external timeslot $n$ also depart in $\mathcal{S}_2$ by slot $(n + \delta)$, for the same arrivals. Clearly,

**Algorithm 14.** *CIOQ-A: Batch-mode Shadow and Decompose*

```
Initialize:            1. For all $(i, j), i \le N, j \le N,\ Q_{i,j} \leftarrow 0$
                       2. For all $(i, j), i \le P, j \le P,\ Q'_{i,j} \leftarrow 0$ (Temporary FIFOs)
                       3. $M \leftarrow$ Batch-size, $T \leftarrow \frac{sN}{s^*P}$

Cell Enqueue:          1. If cell belongs to $(i, j)$, increment $Q_{i,j}$

Every epoch $n = kT$:  1. $\pi^* \leftarrow$ Apply $\mathcal{A}$ on $Q$
                       2. For all $(i, j)$, $i \le N$, $j \le N$, with $\pi^*_{i,j} = 1$ do steps 2a, 2b
                       2a. Decrement $Q_{i,j}$
                       2b. Increment $Q'_{\lceil \frac{i}{P} \rceil, \lceil \frac{j}{P} \rceil}$

Every epoch $n = kMT$: 1. $\Pi^* \leftarrow Q'$

Every internal slot $n$: 1. $\pi \leftarrow$ Maximal/critical matching on $\Pi^*$
                       2. $\Pi^* \leftarrow \Pi^* - \pi$
                       3. $Q' \leftarrow Q' - \pi$ (Dispatch cells)
```

Figure 6.5: CIOQ-A: Batch-mode shadowing and decomposition to emulate a CIOQ with lag

emulation with lag does not imply $f_5$ equivalence. However, since $\delta$ is a constant, assuming the queueing structures in the two switches are the same, stability of a queue in $\mathcal{S}_1$ implies the same in $\mathcal{S}_2$. In other words,

$$(\mathcal{S}_2, \mathcal{A}_2) \overset{T,f_4}{\simeq} (\mathcal{S}_1, \mathcal{A}_1) \tag{6.9}$$

This opens up the possibility of batch-mode decomposition of the shadow configuration $\pi^*$, and yet inherit most of the performance of the reference switch, specifically, everything except $f_5$ equivalence with another (e.g., ideal) switch.

Fig. 6.5 shows the batch-mode shadow-and-decompose method for CIOQ-A, to emulate a CIOQ switch with lag. The shadowing operation remains the same[3] as in Fig. 6.4. In other

[3]The batch-mode decomposition here is not to be confused with the batch-mode matching on the rate-shaper counters $\Pi$ for bandwidth guarantees. In fact, the latter may co-exist and contribute to $\pi^*$ much as in Fig. 6.4. We

words, $\pi^*$ is computed by running $\mathcal{A}$ on the VIOQ state matrix at a frequency of $s^*$ per timeslot. However, instead of immediately assembling the aggregate matrix $\Pi^*$ (6.8), followed by its subsequent decomposition before the next computation of $\pi^*$, the connections in $M$ consecutive shadow matchings are assembled at internal epochs $kMT$, $k \geq 1$, where $M$ is the suitably chosen size of a batch, and $T = \frac{sN}{s*P}$, the number of internal slots between consecutive computations of $\pi^*$, i.e.,

$$\forall k, \forall i \leq P, \forall j \leq P, \quad \Pi^*_{i,j}(kMT) = \sum_{n=(k-1)MT}^{n<kMT} \sum_{k=\frac{N}{P}(i-1)+1}^{\frac{N}{P}i} \sum_{l=\frac{N}{P}(j-1)+1}^{\frac{N}{P}j} \pi^*_{k,l}(n). \qquad (6.10)$$

If all the connections represented in $\Pi^*(kMT)$ are realized before epoch $(k+1)MT$, in a total of $MT$ internal slots, then the CIOQ-A emulates the reference switch with a constant lag. For online decompositions of $\Pi^*$, as shown in Fig. 6.5, the lag is exactly $MT$, or $\delta = M/s^*$ external slots. The decomposition maybe the offline (not shown), i.e., all the configurations $\pi$ may be pre-computed in $[kMT, (k+1)MT)$, and used in $[(k+1)MT, (k+2)MT)$ to drain $\Pi^*(kMT)$, in which case $\delta = 2M/s^*$.

Note that the line sums of $\Pi^*$ never exceed $NM/P$. Since the shadow operation is performed at the same frequency as before, the batch-mode method offers no complexity advantage with respect to the reference switch. However, there may be significant advantages in the decomposition step itself. For example, referring to Table 6.2 for uniform traffic, while the SPS matching runs in $O(1)$ time, the overall complexity is dominated by the decomposition. Consider a batch-mode method instead, with $M = N$. Due to the property of SPS, $\pi^*_{i,j} = 1$ for at most one configuration in such a batch, for all $(i, j)$. Consequently,

$$\forall n, \forall i \leq P, j \leq P, \quad \Pi^*_{i,j}(n) \leq \frac{N^2}{P^2}$$

A sub-maximal perfect sequence of $P$ configurations provides a single service turn to each pair in $\Pi^*$. Therefore, a repeating sequence will drain $\Pi^*$ in $N^2/P$ internal slots, which exactly equals the batch interval with $s = s^* = 1$. In other words, a batch-mode SPS decomposition is sufficient for uniform traffic, bringing the overall complexity down to $O(1)$ for each internal slot, or $O(\frac{N}{P})$ with respect to the external clock.

have not explicitly shown that batching in Fig. 6.5 for the sake of brevity.

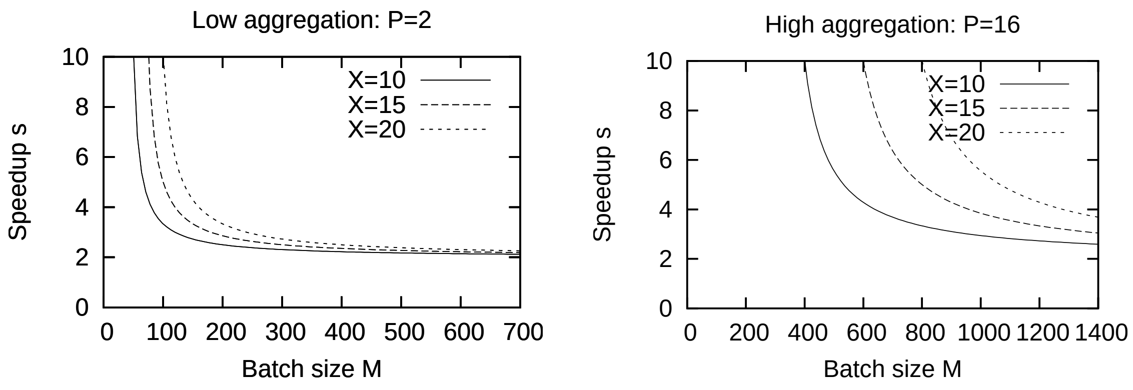


Figure 6.6: CIOQ-A shadowing: Relationship between speedup and batch-size

Recall that one of the advantages of aggregation is the usage of smaller albeit faster space elements, making it especially suited to all-optical cross-connects. The latter, however, suffer from huge reconfiguration delays. Batch-mode operation may be used to resolve this issue in CIOQ-A switches, much like the usage of batch-mode matchings in CIOQ switches proposed by Towles and Dally in [111]. The efficient matrix decomposition technique DOUBLE, from that work, can decompose $\Pi^*$ as a convex combination of $2P$ configurations using a total of $2\frac{NM}{P}$ internal slots. Let $X$ be the reconfiguration overhead in external timeslots. Then, the amount of internal slots wasted in context-switching in each batch equals $(X \times s\frac{N}{P} \times 2P)$, or $2sXN$. Since the total number of slots to cover $\Pi^*$ cannot exceed $MT$, we obtain

$$\begin{aligned} M\frac{sN}{s^*P} &\geq 2\frac{NM}{P} + 2sXN, \\ \text{or} \quad s &\geq \frac{2s^*}{1 - \frac{2s^*XP}{M}} \end{aligned} \tag{6.11}$$

This gives us the following revision of Corollary 10, for batch-mode decompositions:

**Corollary 11.** $\forall f_i, i \neq 5$, *(CIOQ, $\mathcal{A}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$)*, $s \geq s^* \quad \Rightarrow$

$$\textit{(CIOQ-A, \{Shadow-}\mathcal{A}\textit{, DOUBLE\})} \overset{T,f_i}{\simeq} \textit{(OQ, }\mathcal{A}'\textit{)}, \quad s \geq \frac{2s^*}{1 - \frac{2s^*XP}{M}}$$

Note that $M > 2s^*XP$ is required to at least cover the context-switching overhead. When the reconfiguration delay is negligible, i.e., as $X \rightarrow 0$, the speedup approaches 2, much as in maximal decomposition. For large $X$, a suitably large value of $M$ may be chosen to get the speedup close to 2, at the expense of a higher lag, as shown in Fig. 6.6 for a $32 \times 32$ switch. The lag $2M/s^*$ adds to the upper bounds in (5.7) and (5.8), hence worsening the delay guarantees. The complexity of DOUBLE, which equals $O(P^2 \log P)$, amortized over a batch interval of $M/s^* > 2XP$ external slots, results in an overall run-time complexity of $O(\xi + P \log P)$. If the temporary FIFOs $Q'$ can be implemented using on-chip queues, the bandwidth of the external VIOQ memory remains unaffected by the high speedup.

### 6.1.3 Low-complexity Matchings

While shadowing a CIOQ (Fig. 6.4) provides a convenient way to inherit its throughput and QoS performance, we lose one of the benefits of the CIOQ-A design, namely, there is no advantage in terms of arbitration complexity inspite of using a smaller space element. This is primarily because the matching was based on a virtual input-output queue matrix, which exactly resembles the VOQ matrix in a CIOQ switch of the same dimensions. We show here that matching on virtual *element* queues retains the performance of a CIOQ switch for admissible traffic, and realizes the complexity advantage.

Consider a CIOQ-A switch $(N, [MSM], P, 1, s)$, with virtual element queueing. Let $A^*(n)$ be the cumulative arrivals until external slot $n$, for the input-output port pairs, with associated average rates $\lambda^*$. Let $Q(n)$ be the $P \times P$ VEQ length matrix, with $A(n)$ denoting the arrivals into those queues, and $\lambda$ the average rates. As seen before, $(A_{i,j}(n), \lambda_{i,j})$, for element pair $(i, j)$, is simply the summation of $(A^*_{k,l}(n), \lambda^*_{k,l})$ over all the port pairs $(k, l)$ belonging to element pair $(i, j)$. A matching algorithm $\mathcal{A}$ is applied directly on the VEQ matrix, without regard to the specific input and output port, thereby configuring the space element in each internal timeslot, as shown in Fig. 6.7. For QoS guarantees, if desired, rate-shaping is performed on the virtual element queues, using an aggregate of the given $N \times N$ reservation matrix $R^*$, to derive a $P \times P$ batch matrix $\Pi$. In that case, $\mathcal{A}$ operates on $\Pi$ instead of or in addition (SFA combination) to $Q$.

Let the average rates $\lambda^*$ be admissible (2.2). We assume an arbitrary work-conserving policy in the output memory elements, which ensures stable queues for admissible arrivals. Then,

**Algorithm 15.** *CIOQ-A: VEQ Matching $\mathcal{A}$*

```
Initialize:                  1. For all $(i,j), i \leq P, j \leq P$, do steps 1a, 1b, 1c
                                1a. $Q_{i,j} \leftarrow 0$
                                1b. $\Pi_{i,j} \leftarrow 0$ (Batches for QoS)
                                1c. $R_{i,j} \leftarrow \sum_{k=\frac{N}{P}(i-1)+1}^{\frac{N}{P}i} \sum_{l=\frac{N}{P}(j-1)+1}^{\frac{N}{P}j} R^*_{k,l}$
                             2. $M \leftarrow$ 1/GCD($R$)

Cell Enqueue:                1. If cell belongs to $(i,j)$, increment $Q_{\lceil \frac{i}{N/P} \rceil, \lceil \frac{j}{N/P} \rceil}$
Every external epoch $n = kM$: 1. For all $(i,j), i \leq P, j \leq P,\ \Pi_{i,j} = \min(R_{i,j}M, Q_{i,j})$

Every internal slot $n$:     1. $\pi \leftarrow$ Apply $\mathcal{A}$ on $Q, \Pi$
                             2. $Q \leftarrow Q - \pi$ (Dispatch cells)
```

Figure 6.7: CIOQ-A: Direct matching on virtual element queues

the switch provides asymptotic 100% throughput as long as the VEQ system is stable. We first introduce the following general result on the throughput of the system, based solely on the properties of $\mathcal{A}$.

**Theorem 11.** *Given a matching algorithm $\mathcal{A}$, if $\mathcal{A}$ ensures asymptotic 100% throughput to a CIOQ switch for admissible traffic, with a speedup lower-bounded by $s^*$, then $\mathcal{A}$ suffices for 100% throughput to a CIOQ-A switch with virtual element queueing for similarly restricted traffic, provided $s \geq s^*$, i.e.,*

$$\text{(CIOQ, } \mathcal{A}\text{)} \overset{T,f_2}{\simeq} \text{(OQ, } \{\text{WC}\}\text{)}, \quad s \geq s^* \quad \Rightarrow$$
$$\text{(CIOQ-A, } \mathcal{A}\text{)} \overset{T,f_2}{\simeq} \text{(OQ, } \{\text{WC}\}\text{)}, \quad s \geq s^*$$

*Proof.* For admissible $\lambda^*$, the average arrival rates $\lambda$ into the virtual element queues of a CIOQ-A switch satisfy the following, from (6.2):

$$\forall i \leq P,\ \sum_{j=1}^{P} \lambda_{i,j} \leq \frac{N}{P} \qquad \text{and} \qquad \forall j \leq P,\ \sum_{i=1}^{P} \lambda_{i,j} \leq \frac{N}{P}.$$

Consider a shadow $P \times P$ CIOQ switch with virtual output queues, served by a space element operating under $\mathcal{A}$. Let the interface rates be equal to $\frac{N}{P}$, normalized to the interface rates of the CIOQ-A switch under consideration. Further, let the arrivals into the VOQ system exactly mimic those into the VEQ system of the latter.

Clearly, $\lambda$ is admissible for the shadow switch. Therefore, $\mathcal{A}$ ensures VOQ stability with a speedup no less than $s^*$, or an internal link rate of at least $s^*\frac{N}{P}$ and a matching frequency of at least $s^*\frac{N}{P}$ per external slot of the CIOQ-A. The departures from the VEQ system of the latter, under $\mathcal{A}$, would exactly mimic the ones from the VOQ of the shadow switch, for the same matching frequency. Since the VEQ is matched once every $s\frac{N}{P}$ slots, the VEQ remains stable for $s \geq s^*$. □

The above result allows us to apply all the well-known stability results for CIOQ matchings, for admissible traffic, without the need for analyzing them independently on a CIOQ-A switch. Specifically, we obtain the following results, for virtual element queueing:

(CIOQ-A, {MWM}) $\overset{T,f_2}{\simeq}$ (OQ, {WC}), $T$: SLLN
(CIOQ-A, {LPF}) $\overset{T,f_2}{\simeq}$ (OQ, {WC}), $T$: i.i.d.
(CIOQ-A, {Maximal Matching}) $\overset{T,f_2}{\simeq}$ (OQ, {WC}), $s \geq 2$, $T$: SLLN
(CIOQ-A, {Critical Matching}) $\overset{T,f_2}{\simeq}$ (OQ, {WC}), $T$: SLLN
(CIOQ-A, {SPS Matching}) $\overset{T,f_2}{\simeq}$ (OQ, {WC}), $T$: SLLN, Uniform

The first three are consequences of the CIOQ results on MWM, LPF, and maximal matching from [29, 79]. The next two are derived from Theorems 4 and 5, respectively. Note that a direct proof for each of the above results may be easily obtained as an extension of the respective CIOQ proof. For example, the stability proof for maximal matching in [29] may be extended to the VEQ system by modifying (3.10) (pg. 51) as

$$\forall i,j \leq P, \quad \bar{C'}_{i,j}(t) = \sum_k \lambda_{i,k} + \sum_k \lambda_{k,j} - s\frac{N}{P} \qquad \text{if } \bar{Q}_{i,j}(t) > 0$$

From (6.2), $\bar{C'}_{i,j}(t) \leq 0$ for $s \geq 2$, leading to VEQ stability. Similarly, referring to critical matchings in a CIOQ, the phantom bucket evolution in our proof (pg. 120) yields the following

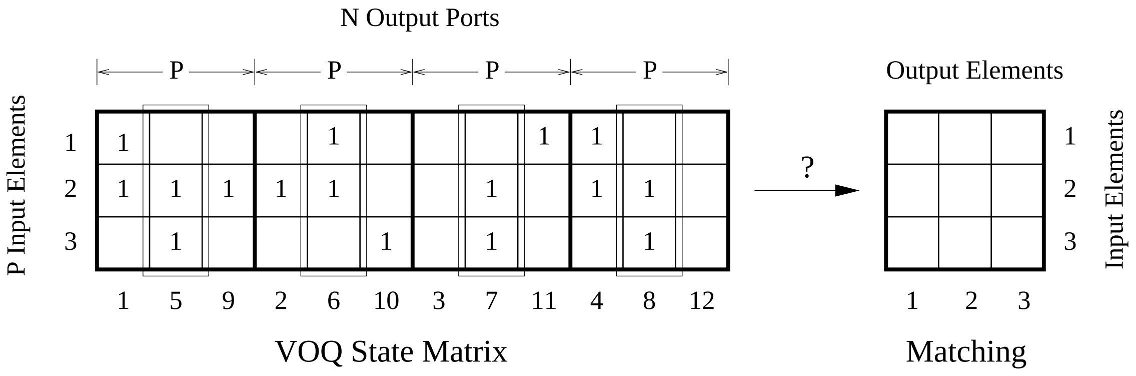


Figure 6.8: CIOQ-A: How do we generate a $P \times P$ matching from a $P \times N$ queue state?

extensions of (5.14) and (5.15):

$$\begin{aligned}\bar{\mathcal{Z}}(t) &\leq \frac{N}{P}t - \bar{D}_{\mathcal{Z}}(t)\\ \bar{D'}_{\mathcal{Z}}(t) &= \frac{N}{P}, \quad \text{if } \bar{\mathcal{Z}}(t) > 0\end{aligned}$$

This immediately leads to a negative drift on the phantom bucket, and the stability of each VEQ. Nevertheless, in light of Theorem 11, there is no need for such independent analyses.

Unfortunately, the stronger equivalence results of CIOQ do not lend themselves to matching on per-element queues in a CIOQ-A. For intuition, notice that if some output $j$ is oversubscribed, we may have $\sum_i \lambda_{i,\lceil \frac{j}{N/P} \rceil} > \frac{N}{P}$. In other words, the over-subscription of $j$ will affect the stability of all the outputs in the same element as $j$, thereby ruling out any isolation of instability per output. We explored virtual output queueing in a CIOQ-A to resolve this situation, since at the very least, the traffic belonging to different output ports are kept separate in such a queueing structure. However, a VOQ structure opens up the challenge of generating a $P \times P$ matching based on a $P \times N$ queue state, as shown in Fig. 6.8. Note that a matching on $Q$ has a constraint, in addition to (2.5), that if an output $j$ is connected in element $i$, all the other outputs (emphasized in the figure) in the same element as $j$ need to remain unconnected in that slot, thereby resulting in one output affecting the others. Matching concurrently on separate quadrants of $Q$ presents the same issue. Therefore, the problem of providing stronger equivalence, specifically, for inadmissible traffic, without shadowing a CIOQ switch remains open.

| $\mathcal{A}$ | **VEQ Matching** |
|---|---|
| **MWM** $\xi = P^3$ | $O(NP^2)$ |
| **Critical** $\xi = P^{2.5}$ | $O(NP^{1.5})$ |
| **SPS** $\xi = 1$ | $O(\frac{N}{P})$ |
| **Maximal** $\xi = P^2$ | $O(NP)$ |
| **EREW Maximal** $\xi = P$ | $O(N)$ |

Table 6.3: CIOQ-A: Arbitration complexity for VEQ matching

### Memory Bandwidth and Matching Complexity

The memory bandwidth required for VEQ matching might be slightly higher than that for shadowing a CIOQ switch. Recognize that, if the queues belonging to each input in the latter are maintained in separate physical memories as in a CIOQ switch, and if the cells served by the shadowing operation are immediately placed in a temporary on-chip FIFO (Fig. 6.4), then the memory bandwidth required equals $(1 + s^*)C$ per output, where $s^* \leq s$ is the speedup of the shadow switch and $C$ is the external link rate. On the contrary, VEQ matching, in general, requires a bandwidth of $(1 + s)\frac{N}{P}C$. This is because each input-output pair in the former is served no more than once every $\frac{sN}{s^*P}$ internal timeslots, while cells belonging to such pairs may be served in consecutive slots with VEQ matching. Consequently, we may also expect the average delay in VEQ to be lower.

If each invocation of $\mathcal{A}$ runs in $O(\xi)$ time on the $P \times P$ VEQ matrix, then the overall matching complexity becomes $O(\xi\frac{N}{P})$ based on the common external clock. Therefore, the complexity results in Table 6.2 for shadowing may be contrasted with those in Table 6.3 for direct VEQ matching. As is evident, the run-time advantage due to the smaller queue-state matrix comes at the expense of unsatisfactory performance in the presence of inadmissible traffic. Note that, much as in a CIOQ switch (pg. 62), the effective complexity can be further reduced using envelope matching, without additional speedup.

Similar to the shadowing case, we may reduce the reconfiguration frequency using batch-mode matchings, to make the design suitable for the usage of optical elements. Specifically, let $s^*$ be the minimum speedup required by $\mathcal{A}$ to ensure VEQ stability, and $s$ be the actual speedup employed to account for batching. Then $\mathcal{A}$ is invoked on $Q$ once every $s/s^*$ internal slots. Let $M$ resulting $P \times P$ configurations be accumulated into matrix $\Pi^*$, which is subsequently decomposed. Since the line sums of $\Pi^*$ do not exceed $M$, we know that DOUBLE decomposes $\Pi^*$ into $2P$ configurations using a total of $2M$ internal slots. The amount of slots wasted in context-switching remain equal to $2sXN$, where $X$ is the reconfiguration overhead in external timeslots. Since the total number of slots to cover $\Pi^*$ cannot exceed $M\frac{s}{s^*}$, we obtain the following modification of (6.11):

$$\begin{aligned} M\frac{s}{s^*} &\geq 2M + 2sXN, \\ \text{or} \quad s &\geq \frac{2s^*}{1 - \frac{2s^*XN}{M}} \end{aligned} \tag{6.12}$$

This gives us the following corollary of Theorem 11 for batch-mode VEQ matching.

**Corollary 12.** *(CIOQ, $\mathcal{A}$) $\overset{T,f_2}{\simeq}$ (OQ, $\{WC\}$), $s \geq s^* \quad \Rightarrow$*
*(CIOQ-A, $\{\mathcal{A}$, DOUBLE$\}$) $\overset{T,f_2}{\simeq}$ (OQ, $\{WC\}$), $s \geq \dfrac{2s^*}{1 - \frac{2s^*XN}{M}}$*

**Hierarchical Output Scheduling**

For queue stability under admissible traffic, there is no need to explicitly distribute the service, given by the VEQ matching to per-element queues, to specific input-output pairs within the elements. However, a hierarchical policy might be beneficial for short-term properties like delay. Fig. 6.9 shows a modified structure suitable for this purpose, in which a VEQ is replaced by a virtual element scheduler (VES). A service turn given by the matching is consumed by the VES and is distributed to an input-output pair, or alternatively, just an output (not shown). Note that this is not the same as VIOQ or VOQ matching, since the arbiter continues to operate on the total VEQ occupancy, without regard to VIOQ/VOQ state. As in CIOQ switches, the hierarchy may be extended to per-flow queues.

Consider the special case of uniform traffic, i.e., $\lambda^*_{i,j} \leq 1/N$, for all input-output port pairs $(i, j)$. Then, $\lambda_{k,l} \leq N/P^2$ for each element pair $(k, l)$, and an SPS matching that serves each VEQ once every $P$ internal slots, or $P^2/N$ external slots for $s = 1$, suffices to keep the

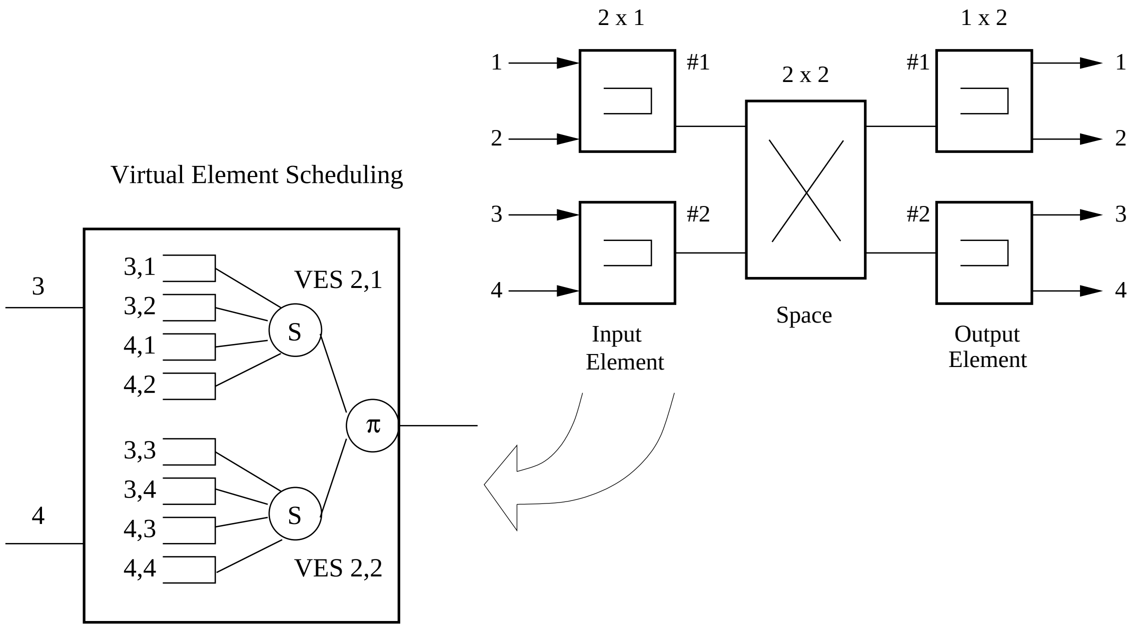


Figure 6.9: CIOQ-A: Replacing a VEQ with a virtual element scheduler

VEQ stable. Let the service turns be explicitly distributed to individual VIOQ in a round-robin fashion. Since there are $N^2/P^2$ queues for each element pair, each VIOQ is guaranteed one turn per $\frac{P^2}{N} \times \frac{N^2}{P^2}$ slots, i.e., $N$ external slots, thereby getting a virtual bandwidth trunk of $1/N$. Such a two-dimensional SPS matching (based on EREW maximal) is illustrated in Fig. 6.10, a variant of which can be found in the Atlanta/$\pi$ [19, 21] chipsets. While there is no advantage with respect to a plain SPS matching on VEQ, it provides a guaranteed trunk to each input-output pair, with a short latency. Furthermore, the required memory bandwidth reduces to $(\frac{N}{P} + 1)C$ per output port, since each output port can get no more than a single service turn in each external slot.

Hierarchical scheduling is also beneficial for bandwidth and delay guarantees to input-output pairs. Given a $N \times N$ rate reservation matrix $R^*$, referring to Fig. 6.7, note that an aggregate $P \times P$ matrix $R$ is first derived for VEQ matching purposes. The rate-shaper assembles a batch $\Pi$ at each epoch $kM$ of the external clock as follows:

$$\forall i, j \leq P, \quad \Pi_{i,j}(kM) = \min(R_{i,j}M, Q_{i,j}(kM)),$$

where $M$ is the reciprocal of the GCD $r$ of the aggregate rates in $R$. Since the row and column

**Algorithm 16.** *Two-dimensional EREW sub-maximal*

```
Initialize:          1. For all $i, j \leq P$, $Q_{i,j} \leftarrow 0$
                     2. For all $i \leq P$, do steps 2a, 2b
                        2a. $T_i = i$    (Output element pointer)
                        2b. For all $j \leq P$, $S_{i,j} = 1$    (I/O pair pointer)

Cell Enqueue:        1. If cell belongs to $(i, j)$, increment $Q_{i,j}$

Every Timeslot $n$:  For all $i \leq P$ in parallel    (Input elements)
                     1. $j \leftarrow T_i$    (Output element)
                     2. $k \leftarrow \lceil \frac{S_{i,j}}{N/P} \rceil + (i-1)\frac{N}{P}$    (Input port)
                     3. $l \leftarrow S_{i,j} - (\lceil \frac{S_{i,j}}{N/P} \rceil - 1)\frac{N}{P} + (j-1)\frac{N}{P}$    (Output port)
                     4. If $Q_{k,l} > 0$, do steps 4a, 4b
                        4a. $\pi_{i,j} \leftarrow 1$
                        4b. Decrement $Q_{k,l}$    (Dispatch cell)
                     5. Increment $T_i$ mod $P$
                     6. Increment $S_{i,j}$ mod $\frac{N^2}{P^2}$
```

Figure 6.10: Example of a two-dimensional $O(1)$ matching in CIOQ-A for uniform traffic

sums of $\Pi$ are no greater than $NM/P$, a maximal matching would drain $\Pi(kM)$ in no more than $2NM/P - 1$ internal slots, and a critical matching would do so in no more than $NM/P$ slots. As there are $s\frac{N}{P}M$ internal slots in a batch interval, each element pair $(i, j)$ is guaranteed a trunk of $R_{i,j}$, provided $s \geq 2 - \frac{r}{N/P}$ for maximal matching and $s \geq 1$ for critical matching. Notice that the former speedup is slightly less than the corresponding one for a CIOQ switch. Also, the latency of scheduling $2M$ experienced by a VEQ is typically less than the latency in a CIOQ switch, since GCD$(R) \geq$ GCD$(R^*)$.

Unlike the stability goal, for which an explicit distribution of service turns to input-output pairs was optional, notice that providing isolated trunks to element pairs is an artificial result– an outcome of the internal switch architecture, which provides no meaningful isolation to input-output pairs. Therefore, hierarchical scheduling becomes mandatory, by which the service given to a VEQ is distributed in proportion to the required rates to fine-grain flows within an element pair, or to the contained input-output pairs. The delay bound for leaky-bucket traffic, in both cases, is given by (5.8). This leads us to the following corollaries of Theorem 11 for bandwidth guarantees, which mirror Theorems 1 and 3 for a CIOQ switch.

**Corollary 13.** *Any maximal matching, combined with rate shaping and hierarchical rate scheduling, is sufficient to provide isolated bandwidth guarantees to input-output pairs of a CIOQ-A switch with virtual element queueing, as long as the internal speedup $s \geq 2 - \frac{r}{N/P}$.*

**Corollary 14.** *A critical matching, combined with rate shaping and hierarchical rate scheduling, is sufficient to provide isolated bandwidth guarantees to input-output pairs of a CIOQ-A switch with virtual element queueing, without internal speedup.*

For simultaneous provision of optimal throughput and QoS, an SFA combination (Sec. 5.6) of the above method on the batch matrix $\Pi$, and a direct matching on $Q$, is required. Since the speedup requirement is not materially different, for both, from a CIOQ switch, the values in Table 5.1 continue to hold for CIOQ-A switches. In fact, the algorithm $\mathcal{A}$ in Fig. 6.7 was an SFA combination.

### Extensions for Multicast Traffic

Theorem 10 for shadowing a CIOQ switch subsumes recycled (Sec. 5.7) multicast traffic to a CIOQ-A switch. If a cell is copied and recycled at the output elements, once for every output port in its fanout set, then the arrivals into a VIOQ at epochs $kT$, where $T$ is the shadowing interval, would exactly mimic those into the respective VOQ of the reference switch, for identical arrivals at the inputs. Accordingly, the proof of Theorem 10 continues to hold for recycling-based multicast traffic. From Corollaries 10 and 6, for $\mathcal{A} = \{\text{Critical Matching}\}$, $s^* = 2$ for 100% throughput to admissible traffic, leading to $s \geq 2$ with critical decomposition and $s \geq 4 - 2\frac{N}{P}$ for maximal decomposition in CIOQ-A. Similarly, from Corollary 7, for $\mathcal{A} = \{\text{Maximal Matching}\}$, $s^* = 3$ yielding $s \geq 3$ for critical decomposition and $s \geq 6 - 3\frac{P}{N}$ for maximal. Since no more than $s^*$ cells are dispatched to each output port per external slot, it is easy to verify that the memory bandwidth required per input port becomes $(1 + 2s^*)C$, assuming the employment of temporary FIFOs for decomposition, where $C$ is the external link capacity.

Notice that shadowing a CIOQ switch for multicast traffic might lead to wasteful copying, since we do not leverage the fact that a cell may be enqueued to several output ports within the same output element at the same time. If $F(c) \subseteq \{i \leq N\}$ is the fanout set for cell $c$, then it takes $|F(c)|$ traversals through the space element for all the copies to be dispatched. On the other hand, direct VEQ matching allows the usage of a more efficient recycling strategy. Specifically, $F(c)$ may be partitioned as follows:

$$\begin{aligned} F_e(c) &= \{i \leq P \mid \left\lceil \frac{i}{N/P} \right\rceil \in F(c)\} \\ \forall i \in F_e(c), \quad F_o(c,i) &= \{j \in F(c) \mid \left\lceil \frac{j}{N/P} \right\rceil = i\} \end{aligned}$$

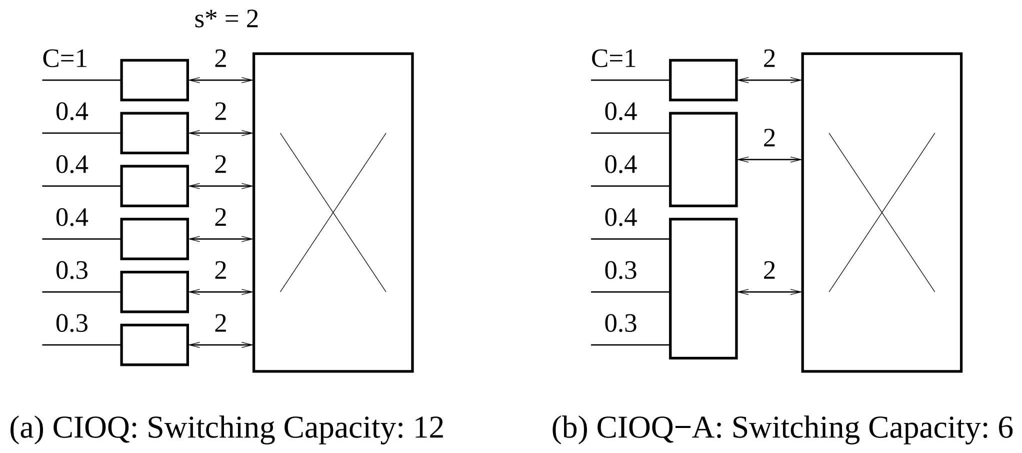
In other words, $F_e(c)$ is the set of destination elements for $c$, and $F_o(c,i)$ is the set of ports within output element $i$ to which $c$ is to be dispatched. Clearly, $F(c) = \bigcup_{i \in F_e(c)} F_o(c,i)$. When a multicast cell $c$ reaches output element $i$, it is immediately copied to all the output ports $F_o(c,i)$, and is recycled to the next element $j$ in $F_e(c)$, by enqueueing it to VEQ $(i,j)$. As a result, it takes $|F_e(c)| \leq |F(c)|$ traversals through the space element, possibly resulting in lower delays. However, the speedup requirement remains the same as in a CIOQ switch, even with efficient recycling. For admissible traffic, the arrivals into the VEQ satisfy:

$$\forall i \leq P, \ \sum_{j=1}^{P} \lambda_{i,j} \leq 2\frac{N}{P} \quad \text{and} \quad \forall j \leq P, \ \sum_{i=1}^{P} \lambda_{i,j} \leq \frac{N}{P}.$$

Therefore, following the same reasoning as in the proof of Theorem 11, the VEQ system remains stable with $s \geq 2$ for critical matching, and with $s \geq 3$ for maximal matching. The memory bandwidth required becomes $(1+2s)\frac{N}{P}C$.

### Support for Subports

The CIOQ-A architecture, with virtual element queueing, provides an efficient way to build switches with heterogeneous link capacities. Consider a $N \times N$ switch, with capacity $C_i$ for external link $i \leq N$. Let $C_{\min}$ and $C_{\max}$ be the minimum and maximum values of $C_i$, respectively. A CIOQ design would require each element to be dimensioned for a $N \times N$ switch with homogeneous link capacities of $C_{\max}$. If $s^*$ is the lower bound on the speedup, assuming



s* = 2
C=1
0.4
0.4
0.4
0.3
0.3
2
(a) CIOQ: Switching Capacity: 12
C=1
0.4
0.4
0.4
0.3
0.3
2
(b) CIOQ−A: Switching Capacity: 6

Figure 6.11: The CIOQ-A architecture applied to a switch with heterogeneous links

homogeneous external links, then the effective speedup of the CIOQ design equals

$$s = \frac{NC_{\max}}{\sum_{i \le N} C_i} s^*$$

On the other hand, several external links may be aggregated in a CIOQ-A switch to homogenize the total element capacity, as shown in Fig. 6.11. In general, the task of selecting the specific links to aggregate is a linear programming problem. Select integer $P < N$ and a $P$-partition $N_i, i \le P$ of the external link set, i.e., $|\bigcup_{i \le P} N_i| = N$. Furthermore, select a rate value $C$, $C_{\min} \le C < \sum_{i \le N} C_i$, and minimize $PC$ under the constraint:

$$\forall i \le P, \quad \sum_{k \in N_i} C_k \le C$$

Then, the effective speedup for a CIOQ-A design would equal the following expression for $s$, which, depending on the relative values of $C_i$, may be significantly lower than the comparable value for a CIOQ design. Clearly, links $N_i$ have been relegated to the role of subports of port $i$, of the space element, and since the arbitration on the VEQ system is unaware of the specific input-output pairs, the value of $s^*$ in Theorem 11 continues to hold.

$$s = \frac{PC}{\sum_{i \le N} C_i} s^*$$

To summarize, one of the main benefits of the CIOQ-A design, namely, lower matching complexity with respect to CIOQ, is realized by virtual element queueing at the expense of unsatisfactory throughput performance in the presence of inadmissible traffic. VEQ may also be expected to yield a lower delay-bound for the guaranteed component of the offered traffic. Shadowing a CIOQ switch, on the other hand, retains all the performance results of CIOQ, but without the complexity advantage. The memory bandwidth at $(1+s)C$ per input port (assuming on-chip decomposition FIFO queues and separate per-input physical memories) is less than the bandwidth required by VEQ, namely, $(1+s)\frac{N}{P}C$. We may address the issue of higher arbitration frequency, which is at least $\frac{N}{P}$ times that in CIOQ, by using envelope-based matching. Finally, both shadowing and VEQ matching might require batch-mode operation, with additional speedup, to reduce the reconfiguration frequency and make it suitable for optical components, in order to realize the smaller space-element advantage.

## 6.2 Spatial Pipelining

Fig. 6.12 shows a $N \times N$ CIOQ-P switch $(N, [MSM], N, K, s)$. We obtain such a switch by replacing the single space element of CIOQ, running at interface rate $s$, by $K$ instances of elements running at $s/K$, normalized to the external rate. Consequently, there are $N$ first-stage memory elements, much as in a CIOQ switch, of dimensions $1 \times K$, $K$ $N \times N$ space elements in the second stage, and $N$ third-stage memory elements of dimensions $K \times 1$. Note that this design refers to a spatial (or super-scalar) pipeline, and not to a temporal one. To prevent head-of-line blocking, virtual output queueing (VOQ) is employed in the input elements. The VOQ system is simultaneously served by the $K$ space elements, resulting in a memory bandwidth requirement of $(1+s)$ times the external capacity, comparable to a CIOQ switch. Clearly, the primary benefit of this design is the employment of slower space elements. Furthermore, if their configurations can be computed in parallel, the arbitration frequency may go down by a factor of $K$.

The time required to transfer a cell on the external links is again referred to as an *external timeslot*. As $s$ cells may be transferred per external timeslot through the second stage, from/to each memory element, we refer to the time unit of $1/s$ as an *internal timeslot*. While one cell may be placed by an input element in each internal slot on some space element $k$, the latter takes a total of $K$ internal slots, or $K/s$ external ones, to transfer the cell to the respective output element. We refer to this transfer time as the *space element (SE) timeslot*. For convenience, we assume that

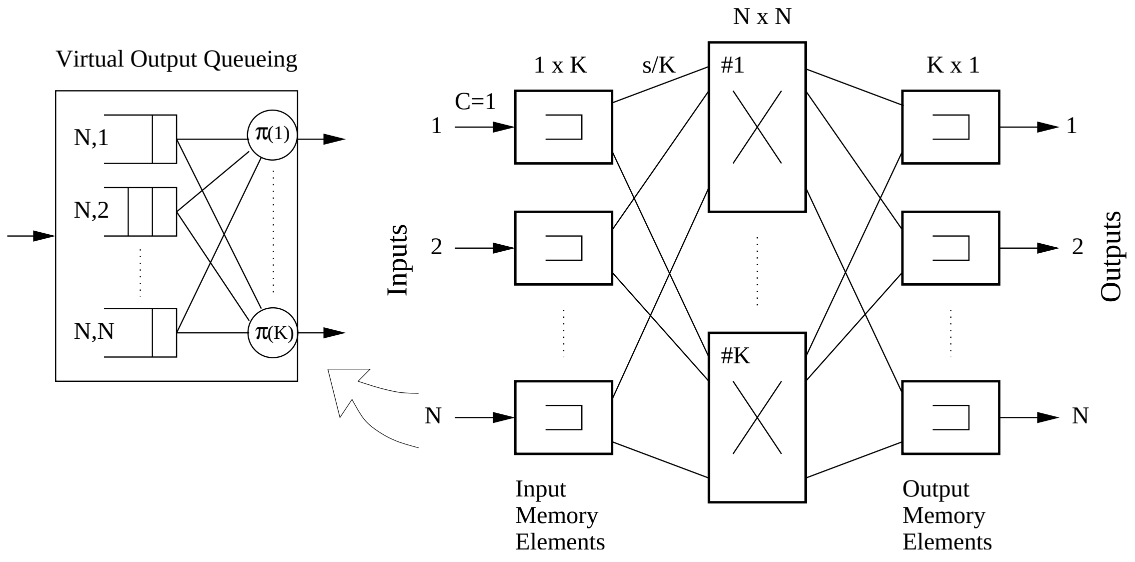


Figure 6.12: CIOQ-P: CIOQ switch with spatial pipelining

the SE slots are staggered as shown in Fig. 6.13. In other words, an SE slot $n$ for space element $k$ occupies the interval $[(n-1)K+k, nK+k)$ of internal timeslots. Then, the VOQ system is served by space element $k$, $k \leq K$ at internal clock epochs $KT+k$, $T>0$. Let $\pi^{(k)}(n)$ be the configuration, which satisfies (2.5), of space element $k$ at its SE slot $n$. Let $\pi(n)$ be the configuration that serves the VOQ system in internal timeslot $n$. Then, we have

$$\forall n, \quad \pi(n) = \pi^{(k)}(\left\lceil \frac{n}{K} \right\rceil), \qquad \text{where } \ k = n \bmod K + 1. \tag{6.13}$$

A sequential matching would compute $\pi(n)$ in each internal slot, i.e., at the same frequency as in a CIOQ switch, and assign it to the appropriate $\pi^{(k)}$. On the other hand, a parallel matching would compute each $\pi^{(k)}$ concurrently, and set $\pi(n)$ to the correct $\pi^{(k)}(n/K)$ in each internal slot $n$.

Let the $N \times N$ discrete-time matrix $A(n)$ denote the cumulative arrivals for each input-output pair until external timeslot $n$, with associated average rates given by $\lambda$. The evolution of the VOQ system may then be described by $N \times N$ matrices $(A(n), D(n), Q(n))$, denoting the cumulative arrivals, cumulative departures and current queue-length, respectively. The relation between them continue to be governed by (3.2) and (3.3) as in a CIOQ switch. Therefore,

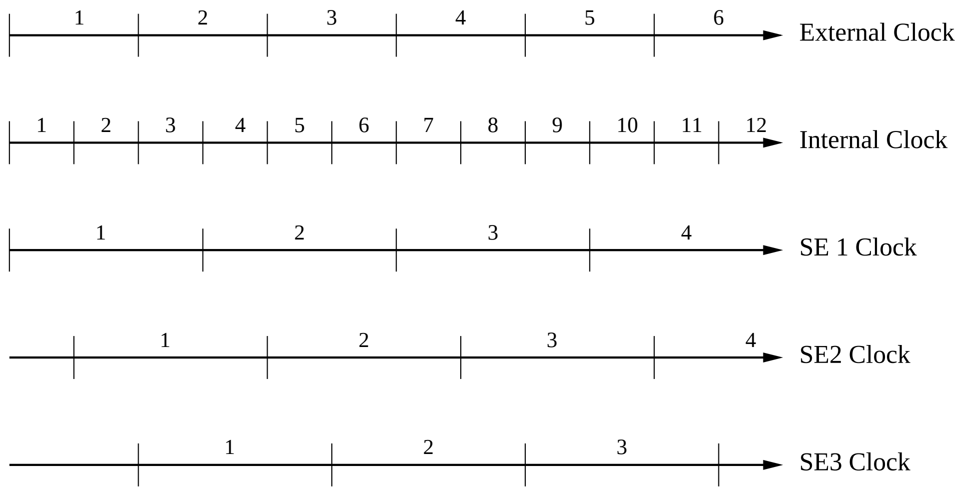


Figure 6.13: Staggered timeslots in a CIOQ-P switch: $s = 2$, $K = 3$

from (6.13), the cumulative departures are given by

$$\forall n, \quad D(n) \geq \sum_{k=1}^{K} \sum_{m=1}^{\lfloor s\frac{n}{K} \rfloor} \pi^{(k)}(m) \tag{6.14}$$

Note that (6.14) only considers SE slots that are completed in all space elements, mainly for notational convenience. A more precise formulation is given below, though not necessary for most throughput analysis.

$$\forall n, \quad D(n) = \sum_{k=1}^{K} \sum_{m=1}^{\lfloor s\frac{n}{K} \rfloor} \pi^{(k)}(m) + \sum_{k=1}^{sn \bmod K} \pi^{(k)}(\left\lceil s\frac{n}{K} \right\rceil)$$

### 6.2.1 Shadowing a CIOQ

We introduce a *shadowing* method, similar to the one seen in the context of a CIOQ-A switch, in order to exactly emulate a CIOQ switch and inherit its performance. Let a $N \times N$ CIOQ switch, employing a set of matching algorithms $\mathcal{A}$, with speedup $s^*$, be the reference switch. Again, $\mathcal{A}$ may contain a throughput-optimizing algorithm such as critical ($s^* \geq 1$, for $f_2$ equivalence with a work-conserving OQ switch), SPS ($s^* \geq 1$, $f_2$ for uniform traffic), maximal ($s^* \geq 2$ for $f_3$) or SOQF ($s^* \geq 3$ for $f_4$) matching; or rate-shaping, given reservation matrix $R$ for input-output pairs, combined with a batch-mode version of critical ($s^* \geq 1$) or maximal ($s^* \geq 2$) matching; or an SFA combination of both, with associated speedup as shown in Table 5.1.

For exact emulation, the CIOQ-P switch computes a matching $\pi^*$ by running $\mathcal{A}$ on the VOQ state matrix $Q$, at a frequency of $s^*$ per external slot. If rate-shaping is desired, $Q$ is used to derive a batch matrix $\Pi$ (5.6) using the required rates $R$, and the computation is instead (or additionally, for SFA) applied on $\Pi$. Clearly, if a cell is dispatched, from the first to the third stage, through some space element, for every connected pair in $\pi^*$, then the departures from the VOQ exactly emulate those in the reference switch. The link schedulers at the outputs are chosen to exactly mimic the one in the reference switch. Since a space element takes $K/s$ external slots to transfer a cell, as opposed to $1/s^*$, the arrivals into the output element lag those in the reference CIOQ switch by $\delta = (\frac{K}{s} - \frac{1}{s^*})$ external slots. As the output schedulers are identical, each cell departs from the system with a constant lag of $\delta$ with respect to the reference. For equivalence purposes, focusing only on the VOQ system, we will accept such behavior as exact emulation, or $f_5$ equivalence.

The task then is to ensure that there is a free space element to realize the connections in each computation of $\pi^*$. We introduce a simple method called *sequential dispatch*, shown in Fig. 6.14, which merely assigns $\pi^*(m)$ to the space element $k(m) \leq K$, whose SE slot $t(m)$ begins exactly at, or immediately after, the end of the $m$th computation of $\pi^*$. That is,

$$\forall m, \quad \pi^{(k(m))}(t(m)) = \pi^*(m) \tag{6.15}$$

**Theorem 12.** *A CIOQ-P switch, with virtual output queueing, exactly emulates a CIOQ switch with algorithm set $\mathcal{A}$ and speedup $s^*$, by shadowing $\mathcal{A}$ to compute matchings $\pi^*$, followed by a sequential dispatch method, as long as $s = s^*$, i.e.,*

$$\textit{(CIOQ-P, }\{\textit{Shadow-}\mathcal{A}\textit{, Sequential Dispatch}\}\textit{)} \overset{T,f_5}{\simeq} \textit{(CIOQ, }\mathcal{A}\textit{)}, \quad s = s^*$$

*Proof.* Fix $s = s^*$. Then, $\pi^*$ is computed once every internal timeslot. Therefore, $\pi^*(m)$ coincides with slot $\lceil \frac{m}{K} \rceil$ of space element $m \bmod K + 1$, from (6.13). As a result, (6.15) may be re-written as

$$\forall n, k, \quad \pi^{(k)}(n) = \pi^*(K(n-1) + k) \tag{6.16}$$

**Algorithm 17.** *CIOQ-P: Shadow and Sequential Dispatch*

```
Initialize:                  1. For all $(i,j), i \leq N, j \leq N,\ Q_{i,j} \leftarrow 0$
                             2. For all $(i,j), i \leq N, j \leq N,\ \Pi_{i,j} \leftarrow 0$ (Batches for QoS)
                             3. $M \leftarrow$ 1/GCD($R$)

Cell Enqueue:                1. If cell belongs to $(i,j)$, increment $Q_{i,j}$
Every external epoch $n = kM$: 1. For all $(i,j), i \leq N, j \leq N,\ \Pi_{i,j} = \min(R_{i,j}M, Q_{i,j})$

Every internal slot $n$:     1. $k \leftarrow n$ mod $K + 1$ (Select space element)
                             2. $\pi^* \leftarrow$ Apply $\mathcal{A}$ on $Q, \Pi$
                             3. $\pi^{(k)} \leftarrow \pi^*$
                             4. $Q \leftarrow Q - \pi^{(k)}$ (Dispatch through SE $k$)
```

Figure 6.14: CIOQ-P: Emulating a CIOQ using shadowing and sequential dispatch

We make two observations. First, notice that each computation of $\pi^*$ is assigned to some space element. Second, there are $K$ internal slots between consecutive selections of space element $k$, $\forall k$, therefore ensuring that the element is free when selected. □

Fig. 6.14 is actually a special case of the sequential dispatch method, for $s = s^*$. It is instructive to recognize that, for general values of $s$, the selected space element $k(m)$, and its SE slot $t(m)$, for the $m$th computation of $\pi^*$ is given by

$$\begin{aligned} k(m) &= \left\lceil \frac{ms}{s^*} \right\rceil \bmod K + 1, \quad \text{and} \\ t(m) &= \left\lceil \frac{ms}{Ks^*} \right\rceil \end{aligned}$$

Since $\pi^*(m)$ is computed at instant $\frac{ms}{s^*}$ of the internal clock, the method in Fig. 6.14 may be generalized by running the lower block at epochs $\lceil \frac{ms}{s^*} \rceil$, $m > 0$. Clearly, if $s < s^*$, several computations of $\pi^*$ may be assigned to the same slot of a space element, thus establishing the necessity of the speedup in Theorem 12. If $s > s^*$, several internal slots are skipped resulting in an under-utilization of the space elements. Nevertheless, $f_5$ equivalence with the reference switch continues to hold.

Much as with shadowing $\mathcal{A}$ in a CIOQ-A switch, if $\mathcal{A}$ suffices in the reference CIOQ switch for functional equivalence $f_i$ with an OQ switch, with speedup $s^*$, then the CIOQ-P switch using shadowing and sequential dispatch achieves the same level of equivalence for the same speedup. The properties of containment (4.5) and transitivity (4.6) lead us to the following corollary of Theorem 12.

**Corollary 15.** *(CIOQ, $\mathcal{A}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$),* $s \geq s^* \quad \Rightarrow$
*(CIOQ-P, $\{$Shadow-$\mathcal{A}$, Sequential Dispatch$\}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$),* $s \geq s^*$

This allows us to translate the performance results for a CIOQ switch, from the literature and Chapter 5, to a CIOQ-P switch, without requiring an independent analysis of the latter. Specifically, sequential dispatch may be combined with MWM or critical matching, $s \geq 1$, for asymptotic 100% throughput for admissible arrivals, with SPS matching, $s \geq 1$, for uniform arrivals, with maximal matching, $s \geq 2$ for non-oversubscribed outputs, or with SOQF matching, $s \geq 3$ for conforming input-output pairs. Similarly, sequential dispatch may be combined with BVN templates or batch-mode critical matching on $\Pi$, $s \geq 1$, or with batch-mode maximal matching, $s \geq 2$, for bandwidth and delay guarantees. The delay bounds in (5.7) and (5.8) continue to hold, with an additional latency of $\delta$. For simultaneous provision of optimal throughput and QoS, the required speedup values remain the same as in the reference CIOQ switch, shown in Table 5.1.

Recognize that the sequential dispatch method relies on the same space-time duality of Clos networks exposed in Sec. 5.1.1. We merely transformed $K$ consecutive slots, of a CIOQ switch, to $K$ spatially separated slots. If the algorithm set $\mathcal{A}$ runs in $O(\xi)$ time in the reference switch, the matching complexity remains $O(\xi)$ in the CIOQ-P switch, since $\pi^*$ is computed by a centralized arbiter at the same frequency as in the reference CIOQ design. In other words, there is no complexity advantage for sequential dispatch, despite using a super-scaler pipeline.

### 6.2.2 Concurrent Dispatch

We now address the issue of parallelizing the computation of matchings, over the $K$ space elements, with the goal of reducing the overall complexity. A parallel variant of the sequential

**Algorithm 18.** *CIOQ-P: Concurrent Dispatch*

```
                      Initialize:  Same as in Fig. 6.14
                   Cell Enqueue:  Same as in Fig. 6.14
  Every external epoch $n = kM$:  Same as in Fig. 6.14

                                  For every space element $k$ in parallel:
               Every SE slot $n$:  1. $Q \leftarrow Q - \pi^{(k)}$ (Dispatch through SE $k$)
                                  2. $\pi^{(k)} \leftarrow$ Apply $\mathcal{A}$ on $Q, \Pi$
```

Figure 6.15: CIOQ-P: Emulating a CIOQ using a concurrent matching

dispatch method of Fig. 6.14, called *concurrent dispatch*, is shown in Fig. 6.15. Here, each computation of $\pi^{(k)}$ takes an entire SE slot, and is carried out by a separate arbiter for each space element, using, in general, the queue state at the beginning of the SE slot. The matching computed in internal slots $[n-K, n)$, by space element ($n$ mod $K+1$), is subsequently set to $\pi(n)$, and is used to dispatch cells from the VOQ system at slot $n$. Clearly, if $\mathcal{A}$ runs in $O(\xi)$ time, the matching complexity becomes $O(1 + \frac{\xi}{K})$.

Observe that $\pi(n)$ does not shadow the configurations in a reference CIOQ switch, under identical arrivals, since it is based on $Q(n-K)$. More importantly, a pair $(i, j)$ may be connected by several, say $k \le K$, space elements concurrently, even if $Q_{i,j} < k$, leading to several wasted connections, and possibly less than 100% throughput. This situation arises because $\pi^{(k)}(\lceil \frac{n}{K} \rceil)$ is based on $Q(n)$, and before the configuration may be used to dispatch any cell, at the end of SE slot $\lceil \frac{n}{K} \rceil$, several cells are dispatched from $Q$ based on configurations that completed in $(n, n+K)$. To provide further intuition, let $\mathcal{A} = \{\text{Maximal Matching}\}$. Let $C_{i,j}(n)$ denote the sum of row $i$ plus the sum of column $j$, of the VOQ matrix, at the beginning of the SE slot that ends at internal slot $n$. Let $D^{(k)}_{C_{i,j}}(n)$ be the cumulative connections made for those queues, by space element $k$, until slot $n$. Then, for a maximal matching, we have

$$\begin{aligned}
\forall i, j, n, \quad C_{i,j,k}(n) &= \sum_l Q_{i,l}(n-K) + \sum_l Q_{l,j}(n-K) \\
D^{(k)}_{C_{i,j}}(n) &\ge D^{(k)}_{C_{i,j}}(n-K) + 1, \quad \text{if } Q_{i,j}(n-K) > 1, \ k = n \bmod K + 1
\end{aligned}$$

In the fluid model, therefore, $\bar{D}^{(k)\prime}_{C_{i,j}}(t) \geq \frac{s}{K}$, in terms of the external clock. Unfortunately, however, the total departures $D_{C_{i,j}}(n)$, from the queues represented in $C_{i,j}$, do not increase at a rate of $s$, because

$$D_{C_{i,j}}(n) \neq \sum_k D^{(k)}_{C_{i,j}}(n)$$

Therefore, the conditions in the maximal-matching proof in [29] that lead to $\bar{C}'_{i,j}(t) \leq 0$, whenever $\bar{Q}_{i,j}(t) > 0$, cease to hold for any constant value of $s$. In other words, a maximal matching on stale queue states fails to ensure asymptotic 100% throughput, for admissible traffic, with $s = 2$.

Nevertheless, a restricted class of matching algorithms does lend itself to concurrent dispatch, without losing performance. As is evident, algorithms that are insensitive to the queue state, such as BVN templates and SPS matching, may readily be applied. More interestingly, by slightly modifying the queue-state update procedure of Fig. 6.15, so that an update is made immediately when a connection is granted by a space element, some greedy matching algorithms may also be applied to concurrent dispatch. The intuition is that, for greedy methods, a connection once made remains unaffected by the actions in the remainder of the SE slot, and hence may be immediately reflected in the queue state, even though the corresponding cell departs in the next SE slot. This ensures that each space element uses the latest VOQ information. We show a specific instance of a parallel maximal matching that illustrates such an application, later in this section.

**BVN Decomposition**

For bandwidth and delay guarantees, the same repeating sequence of templates, generated using a BVN decomposition (pg. 46) of $R$, may be applied on each space element. Recall that, given an admissible $R$, it can be expressed as a convex combination of permutations, such that the sum of the weights does not exceed one. Therefore, $\exists T, T \leq 1/\text{GCD}(R)$, such that a sequence of size $T$, of the above permutations, guarantees $R_{i,j}T$ service turns to each pair $(i, j)$. Consequently, with $s = 1$, each space element, in $KT$ external slots, ensures $R_{i,j}T$ turns to the pair, for a total of $R_{i,j}KT$ over all the space elements. We may therefore state the following corollary of the BVN decomposition theorem.

**Algorithm 19.** *CIOQ-P: Best-fit BVN assignment*

```
Initialize:    1. $\{P_i\} \leftarrow$ Set of permutations from a BVN decomposition of $R$
               2. $M \leftarrow |\{P_i\}|$
               3. $\phi_i \leftarrow$ Weight of permutation $P_i$,  $\phi \leftarrow$ GCD$(\phi_i)$
               4. For all $i \le M, T_i \leftarrow \phi_i/\phi$   (Size of sub-sequences)
               5. $T \leftarrow \sum_i T_i$

Assign:        1. For all $k \le K$, $P^{(k)} \leftarrow$ Empty set
               2. $k \leftarrow 1, T^{(k)} \leftarrow 0$
               3. For all $i \le M$, do steps 3a–d
                  3a. $T^{(k)} \leftarrow T^{(k)} + T_i$
                  3b. $P^{(k)} \leftarrow P^{(k)} \bigcup \{P_i\}$, $\phi_i^{(k)} \leftarrow \phi_i$  (Assign $P_i$ to element $k$)
                  3c. If $T^{(k)} = \lceil \frac{T}{K} \rceil$, increment $k$, $T^{(k)} \leftarrow 0$
                  3d. While $T^{(k)} \ge \lceil \frac{T}{K} \rceil$, split $(i, k{+}, T^{(k)} - \lceil \frac{T}{K} \rceil)$

Split $(i,k,\delta)$:  1. $\phi_i^{(k)} \leftarrow \phi_i^{(k)} - \delta\phi$
               2. $T^{(k+1)} \leftarrow \delta$
               3. $P^{(k+1)} \leftarrow \{P_i\}$, $\phi_i^{(k+1)} \leftarrow \delta\phi$  (Also assign $P_i$ to element $k+1$)
```

Figure 6.16: Best-fit assignment of BVN templates to $K$ space elements

**Corollary 16.** *The BVN decomposition procedure on an admissible $R$ is sufficient to ensure bandwidth guarantees, specified by $R$, to input-output flows in a CIOQ-P switch, by concurrently applying the same sequence of permutations on each space element.*

Notice that the scheduling latency becomes $KT$ slots as opposed to $T$ in a CIOQ switch under BVN decomposition. Each space element ends up providing a trunk of bandwidth $R_{i,j}/K$ to pair $(i, j)$, in a balanced fashion. Similar to the above result, template sequences based on other matrix decomposition schemes, such as MIN and DOUBLE, with the associated speedup values, may also be applied to concurrent dispatch, yielding a similar latency.

In reality, a lower latency of $T$ external slots, comparable to a CIOQ switch, may be achieved by selectively assigning the BVN permutations, and associated weights, to the $K$ space elements, at the expense of a modest internal speedup. An unbalanced *best-fit* BVN-template assignment method is shown in Fig. 6.16, in which the set of permutations $\{P_i\}$, with respective weights $\phi_i$, are partitioned into $K$ sets $P^{(k)}$, $k \leq K$, one for each space element. Note that some of the permutations may be represented in more than one element. A sequence of size $\lceil T/K \rceil$, over all the space elements in a concurrent fashion, end up providing $R_{i,j}T$ turns to each pair. The minimum speedup $s_{\min}$, therefore, is given by

$$\begin{aligned} s_{\min} &= \frac{\lceil T/K \rceil}{T/K} \\ &\leq \frac{(T+K)/K}{T/K} \leq 1 + \frac{K}{T} \\ &\leq 1 + Kr, \quad \text{where } r = \text{GCD}(R) \end{aligned}$$

Notice also that $s_{\min} \leq 2$ if $T \geq K$. Also, recognize that the speedup can be made close to one, at the expense of additional latency, by performing best-fit on $K_s \leq K$ repeating sequences of templates, in which case $s_{\min} \leq (1 + \frac{K_s}{K} r)$. As is evident, when $K_s = K$, the speedup requirement drops to one (since $\lceil K_s T/K \rceil = KsT/K$), and the latency becomes exactly $KT$ slots, no better than with a balanced assignment of all the permutations to each space element, as in Corollary 16. Then, the delay bound for leaky-bucket constrained input-output flows (pg. 106) may be easily verified as

$$D_{i,j} \leq \frac{\sigma_{i,j}}{R_{i,j}} + \frac{K}{r} \tag{6.17}$$

**Concurrent SPS**

Recall (pg. 123) that a repeating sequence of $N$ perfect configurations that cover the unit matrix, or SPS matching, is sufficient for 100% throughput to uniform arrivals in a CIOQ switch. Since the configurations are insensitive to queue state, the matching lends itself to concurrent dispatch. Fig. 6.17 shows a concurrent version of the $O(1)$ EREW sub-maximal algorithm in Fig. 5.11, which generates $K$ perfect sequences, one on each space element. It is evident that each element guarantees a single service turn to each input-output pair in $N$ consecutive SE slots. For $s = 1$, therefore, each pair receives a total of $K$ turns in $KN$ external slots, or a trunk of bandwidth $1/N$, which suffices for VOQ stability under uniform arrivals. Hence, we state the following corollary of Theorem 5.

**Algorithm 20.** *CIOQ-P: Concurrent SPS*

```
Initialize:           1. For all $(i,j)$, $Q_{i,j} \leftarrow 0$
                      2. For all $k \leq K$ do          (For all space elements)
                         2a. For all $i \leq N$, $P_i^{(k)} = i$ (Initialize pointers)

Cell Enqueue:         1. If cell belongs to $(i,j)$, increment $Q_{i,j}$

                      For every space element $k$ in parallel:
Every SE slot $n$:    For all $i \leq N$ in parallel (all inputs)
                      1. $\pi_{i,j}^{(k)} = 1$ iff $j = P_i^{(k)}$
                      2. Increment $P_i^{(k)}$ mod $N$

Every internal slot $n$:  1. $k \leftarrow n$ mod $K+1$, $\pi \leftarrow \pi^{(k)}$
                      2. For all $(i,j)$, if $Q_{i,j} > 0$ and $\pi_{i,j} = 1$, dequeue cell
```

Figure 6.17: Concurrent SPS matching: EREW sub-maximal implementation

**Corollary 17.** *A concurrent SPS matching is sufficient, without speedup, for asymptotic 100% throughput in a CIOQ-P switch, for uniform admissible arrivals, i.e.,*

$$\text{(CIOQ-P, \{Concurrent SPS\})} \overset{T,f_2}{\simeq} \text{(OQ, \{WC\})}, T\text{: SLLN, Uniform}$$

Notice that each space element uses $N$ desynchronized pointers, each of which, in parallel, is updated with a single $O(1)$ operation in each SE slot. Therefore, the concurrent matching itself runs in $O(\frac{1}{K})$ time on the external clock. Recognize also that a slightly different pointer initialization, specifically, $P_i^{(k)} = (i+k)$ mod $N+1$, reduces the timescale over which the trunk of $1/N$ is provided to each pair, from $KN$ to $N$ external slots. Most of the concurrent dispatch schemes in the literature [19, 85], for such switches, are iterative request-grant-accept (RGA) (see pg. 58) implementations of the EREW sub-maximal matching above. We have merely shown here, based on the SPS analysis on CIOQ switches, that a single iteration of such algorithms would suffice for uniform traffic.

**Algorithm 21.** *CIOQ-P: 3D Maximal Matching*

```
Initialize:            1. For all $(i,j)$, $Q_{i,j} \leftarrow 0$
                       2. For all $k \le K$ do        (For all space elements)
                          2a. For all $i \le N$, $P_i^{(k)} = i$ (Initialize pointers)

Cell Enqueue:          1. If cell belongs to $(i,j)$, increment $Q_{i,j}$

                       For every space element $k$ in parallel:
Every SE slot $n$:     1. For all $(i,j)$, $\pi_{i,j}^{(k)} \leftarrow 0$
                       2. For each output $j$, $D_j^{(k)} = 0$   (Column state)
                       3. For each input $i$ in parallel, until connected, do $N$ steps of 3a–d
                          3a. $j \leftarrow P_i^{(k)}$
                          3b. If $D_j^{(k)} = 1$, go to 3d   (Column already connected)
                          3c. If $Q_{i,j} > 0$, do 3c1 and 3c2, else do 3d
                            3c1. $\pi_{i,j}^{(k)} \leftarrow 1$, decrement $Q_{i,j}$
                            3c2. $D_j^{(k)} \leftarrow 1$,  $P_i^{(k)} \leftarrow i$    (Reset for next SE slot)
                          3d. Increment $P_i^{(k)}$ mod $N$

Every internal slot $n$:  1. $k \leftarrow n$ mod $K+1$, $\pi \leftarrow \pi^{(k)}$
                       2. For all $(i,j)$, if $\pi_{i,j} = 1$, dequeue cell
```

Figure 6.18: A three-dimensional EREW concurrent matching for CIOQ-P

## 3D Maximal Matching

We now present a concurrent dispatch algorithm, called *3D maximal matching* (3DMM), shown in Fig. 6.18, which provides 100% throughput to non-uniform arrivals, under some restrictions. The basic idea is to run a $N$-step EREW maximal matching (Fig. 3.9) in each SE slot of all the space elements, in parallel. Each element uses an independent EREW column-state vector $D_N^{(k)}$, and $N$ desynchronized pointers, one for each input. A single matching $\pi^{(k)}$ is computed in each SE slot by element $k$, which is then used to dispatch cells at the end of the slot.

The space elements share the VOQ length matrix $Q$. If $K \le N$, due to the staggering

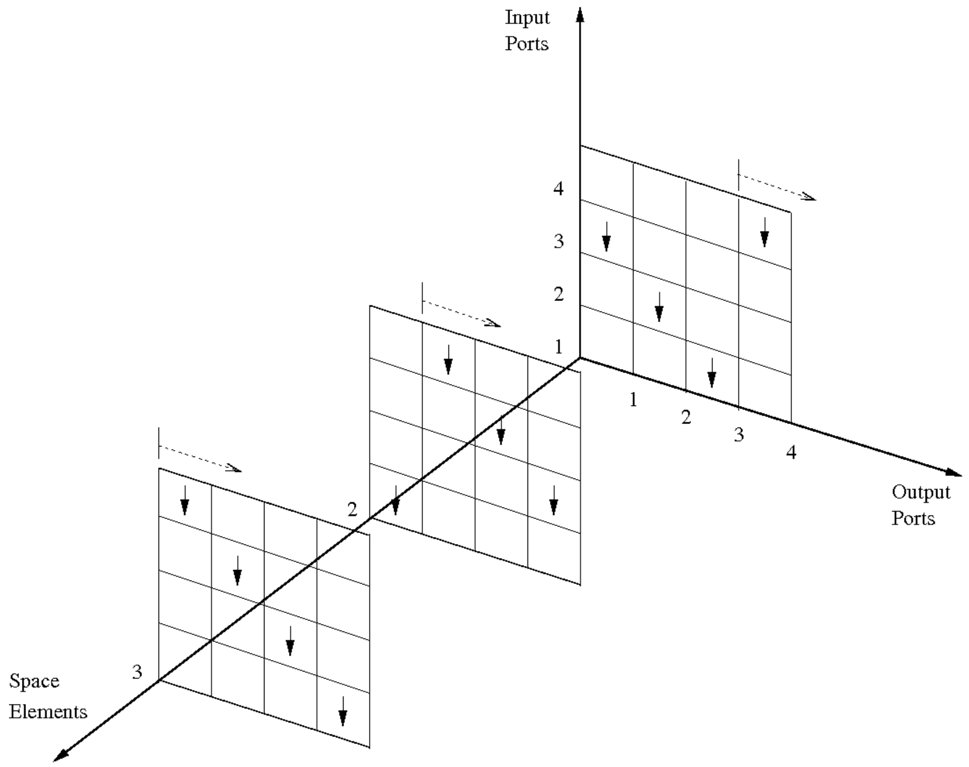


Figure 6.19: 3D maximal matching: A cube of desynchronized pointers

of SE slots, $\forall i, P_i^{(k)} \neq P_i^{(l)}$ for $k \neq l$, at every instant. Therefore, $\forall i, j$, $Q_{i,j}$ is never accessed simultaneously by more than one space element, hence ensuring EREW operation. In $m$ internal timeslots, within an SE slot, each pointer is advanced $\lfloor \frac{mN}{K} \rfloor$ times. As a result, the $NK$ pointers occupy non-conflicting positions in a $N \times N \times K$ cube, as shown in Fig. 6.19. In other words, if $(i, j, k)$ is occupied by a pointer, all the three lines originating from that point are free of other pointers. To ensure that the same cell is not served by more than one space element, $Q_{i,j}$ is immediately decremented when $\pi_{i,j}^{(k)}$ is set to one, for any $k$. Clearly, such a concurrent scheme works only because the queue states are inspected in an orderly fashion by each element, and since the algorithm is greedy. As there are $N$ steps in each SE slot, 3DMM runs in $O(\frac{N}{K})$ time.

**Theorem 13.** *A 3D maximal matching is sufficient for asymptotic 100% throughput in a $N \times N$ CIOQ-P switch, for all flows $(i, j)$ belonging to non-oversubscribed outputs, as long as the speedup $s \geq 2$, and $K \leq N$, i.e.,*

$$\text{(CIOQ-P, \{3DMM\})} \overset{T,f_3}{\simeq} \text{(OQ, \{WC\})}, \quad s \geq 2, K \leq N, T\text{: SLLN}$$

*Proof.* For each $(i,j)$, and external slot $n$, define $T_{i,j}$ as:

$$T_{i,j}(n) = \sum_l Q_{i,l}(n) + \sum_l Q_{l,j}(n)$$

Let $D_{T_{i,j}}(n)$ represent the cumulative departures from row $i$ and column $j$ of $Q$, until timeslot $n$. Notice that, given $Q_{i,j}(n) \geq K$, *every* space element either serves $(i,j)$ once, or a distinct cell from row $i$ and/or column $j$, in the interval $[n, n+\frac{K}{s})$, i.e., in $K$ internal slots. This is because $K$ slots are sufficient, for each element, to visit every location $(i,j)$ in $Q$, unless row $i$ and/or column $j$ are already connected by that element in its current configuration. Therefore,

$$\begin{aligned} \forall i,j,n \qquad T_{i,j}(n) &= \sum_k A_{i,k}(n) + \sum_k A_{k,j}(n) - D_{T_{i,j}}(n) \\ D_{T_{i,j}}(n+\frac{K}{s}) &\geq D_{T_{i,j}}(n) + K, \quad \text{whenever } Q_{i,j}(n) \geq K \end{aligned}$$

As a result, for $s = 2$, the above equations translate exactly to (5.23) and (5.24), respectively, in the fluid limit. The rest of the proof is identical to the proof of Theorem 8. □

We conclude by noting that, while a general algorithm set $\mathcal{A}$ might not easily lend itself to concurrent dispatch, template-based methods and greedy matchings that proceed in an orderly fashion may be adapted, to lower the complexity by a factor of $K$ with respect to sequential dispatch. For the algorithms shown here, we have demonstrated functional equivalence with an OQ switch, up to level $f_3$, for $K \leq N$. In fact, concurrent BVN templates and 3DMM may be integrated into a multi-phase SFA combination, with $s \geq 2$, for simultaneously providing QoS and optimal throughput.

### 6.2.3 Balanced Matchings

To complete our analysis of CIOQ-P switches, we propose two low-complexity methods to achieve comparable performance with a CIOQ switch, without the limitations posed by the concurrent dispatch schemes of the previous section. We leverage the fact that each space element of a CIOQ-P switch may be viewed as the central stage of a $N \times N$ CIOQ switch, with interface

**Algorithm 22.** *CIOQ-P: Shadow and Stripe*

```
Initialize:
    1. For all $(i,j), i \leq N, j \leq N,\ Q_{i,j} \leftarrow 0$
    2. For all $(i,j), i \leq N, j \leq N,\ \Pi_{i,j} \leftarrow 0$ (Batches for QoS)
    3. $M \leftarrow$ 1/GCD($R$)

Cell Enqueue:
    1. If cell belongs to $(i,j)$, increment $Q_{i,j}$

Every external epoch $n = mKM$:
    For all $(i,j), i \leq N, j \leq N$,
    1. $\Pi_{i,j} = \min(R_{i,j}MK, Q_{i,j})$
    2. If $Q_{i,j}$ mod $K \neq 0$, pad $\Pi_{i,j}$

Every internal epoch $n = mK$:
    1. For all $(i,j)$, $G_{i,j} \leftarrow 1_{\{Q_{i,j} \geq K\}}$ (Consider full envelopes)
    2. For all $(i,j)$, $G'_{i,j} \leftarrow 1_{\{\Pi_{i,j} \geq K\}}$
    3. $\pi^* \leftarrow$ Apply $\mathcal{A}$ on $G, G'$

Every internal slot $n$:
    1. $k \leftarrow n$ mod $K + 1$
    2. $\pi^{(k)} \leftarrow \pi^*$
    3. $Q \leftarrow Q - \pi^{(k)}$ (Dispatch through SE $k$)
```

Figure 6.20: CIOQ-P: Emulating an envelope-based CIOQ matching using striping

rates of $1/K$ times that of the switch under consideration. Accordingly, if each input-output pair $(i,j)$, with average rate $\lambda_{i,j}$, offers $\lambda_{i,j}/K$ to each element, we may expect the throughput performance (except exact emulation) of a general algorithm set $\mathcal{A}$ to be equivalent to that in a CIOQ switch. Essentially, the offered load is balanced over all the space elements, on an input-output pair basis. We call such methods *balanced matchings*.

### Striping

Consider a $N \times N$ CIOQ-P switch with VOQ and $K$ space elements. We introduce a sequential method called *striping*, which translates a general algorithm set $\mathcal{A}$ to a balanced matching, as shown in Fig. 6.20. Such a load-balancing technique is commonly used in other areas like parallel disk arrays. Given $\mathcal{A}$, we essentially shadow a reference CIOQ switch, operating under $\mathcal{A}$ on envelopes [54] (see pg. 62) of size $K$, with speedup $s^*$. A centralized arbiter first computes a matching $\pi^*$ by running $\mathcal{A}$ on the VOQ state matrix, at a frequency of $s^*/K$ per external slot. The method only considers queues with at least a full envelope of $K$ cells. In practice, the matching may be computed using a temporal pipeline, in which the configuration generated in $[(m-1)\frac{K}{s^*}, m\frac{K}{s^*})$ is consumed by the VOQ system at epoch $m\frac{K}{s^*}$. For a constant $s^*$, therefore, the complexity becomes $O(\frac{\xi}{K})$ on the external clock, if each invocation of $\mathcal{A}$ runs in $O(\xi)$ time.

To ensure that an envelope worth of cells are dispatched for every connected pair in $\pi^*$, the same configuration is assigned to each space element in a sequential fashion, in the ensuing internal slots. Thereafter, each of those elements transfer a single cell from the connected envelopes. In essence, after an envelope-based matching is generated, we break the envelopes and assign a single cell to each element. Consequently, the overall matching complexity becomes $O(1+\frac{\xi}{K})$.

**Theorem 14.** *A CIOQ-P switch, with $K$ space elements, exactly emulates a CIOQ switch, operating under $\mathcal{A}$ on envelopes of size $K$, with speedup $s^*$, by shadowing $\mathcal{A}$ to compute envelope-based matchings $\pi^*$, followed by striping $\pi^*$ over all the space elements, provided $s \geq s^*$, i.e.,*

$$\textit{(CIOQ-P, }\{\textit{Shadow-}\mathcal{A}\textit{, } K\textit{-Striping}\}\textit{)} \overset{T,f_5}{\simeq} \textit{(CIOQ, }\mathcal{A} \bigcup \{K\textit{-Envelopes}\}\textit{)}, \quad s \geq s^*$$

*Proof.* The envelope-based matchings $\pi^*$ are ready at epochs $m\frac{K}{s^*}, \forall m > 0$, of the external clock, exactly as in the reference CIOQ switch with envelope matching. Furthermore, both switches serve the same envelopes, under identical arrivals, provided a single envelope is fully assigned, for every connection made at epoch $m\frac{K}{s^*}$, in the interval $[m\frac{K}{s^*}, (m+1)\frac{K}{s^*})$. Since it takes $K$ consecutive internal slots to assign all the cells in connected envelopes, we achieve exact emulation as long as $s\frac{K}{s^*} \geq K$, or $s \geq s^*$. □

We hasten to add that exact emulation holds only for the departures from the VOQ system. Since each space element takes an additional $\delta = (\frac{K}{s} - \frac{1}{s^*})$ slots to transfer a cell, with respect to the reference switch, the CIOQ-P emulates the reference with a lag of $\delta$. The above result holds for a general set $\mathcal{A}$, thereby allowing to inherit the properties of $\mathcal{A}$ on a CIOQ switch, while lowering the run-time complexity. A direct consequence of the main result in [54] is that queue stability is not affected in envelope-based matching, with respect to a cell-based matching. In terms of functional equivalences, therefore,

$$\forall k, (\text{CIOQ}, \mathcal{A} \cup \{k\text{-Envelopes}\}) \overset{f_4}{\simeq} (\text{CIOQ}, \mathcal{A}) \tag{6.18}$$

The speedup requirement in both cases are identical. Recall that the primary disadvantage of envelope-based matching is a higher, possibly infinite, cell delay, as a flow may have a perpetually unserved partial envelope. In practice, this situation may be rectified by using the fix suggested for packet-based matchings in [36], namely, using a modest increase in speedup. If the cell-based scheme requires a speedup of $s$, the envelope-based variant may operate at $(1+\epsilon)s$, where $\epsilon$ is arbitrarily small, so that partial envelopes are considered in the matching once every $(1+\frac{1}{\epsilon})$ internal slots. From (6.18), together with the containment (4.5) and transitivity (4.6) properties of equivalences, we obtain the following corollary of Theorem 14.

**Corollary 18.** $\forall f_i, i \neq 5$, *(CIOQ, $\mathcal{A}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$)*, $s \geq s^* \quad \Rightarrow$

*(CIOQ-P, {Shadow-$\mathcal{A}$, $K$-Striping})* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$)*, $s \geq s^*$

Much as in sequential dispatch, this allows us to translate the performance results for a CIOQ switch, from the literature and Chapter 5, to a CIOQ-P switch, without independent analysis of each algorithm. For example, an MWM or critical matching, with $s \geq 1$ and lower complexity with respect to sequential dispatch, may be used for 100% throughput to admissible arrivals. Similarly, a simple maximal matching, with $s \geq 2$, and without the restriction of $K \leq N$ in 3DMM, suffices for the same using a comparable run-time. Matchings such as SOQF, not easily adaptable to concurrent dispatch, may now be used for similar functional equivalence with an OQ switch.

Note that $\mathcal{A}$ subsumes QoS-capable algorithms, such as BVN decomposition ($s \geq 1$) and rate-shaping combined with batch-mode critical ($s \geq 1$) or maximal matching ($s \geq 2$). To ensure that batches reside on envelope boundaries, the batching is performed at epochs $mKM$, $m > 0$, as opposed to $mM$ in a cell-based CIOQ switch. In addition, as shown in Fig. 6.20, batches with partial envelopes are padded to prevent infinite delays. No additional speedup is required

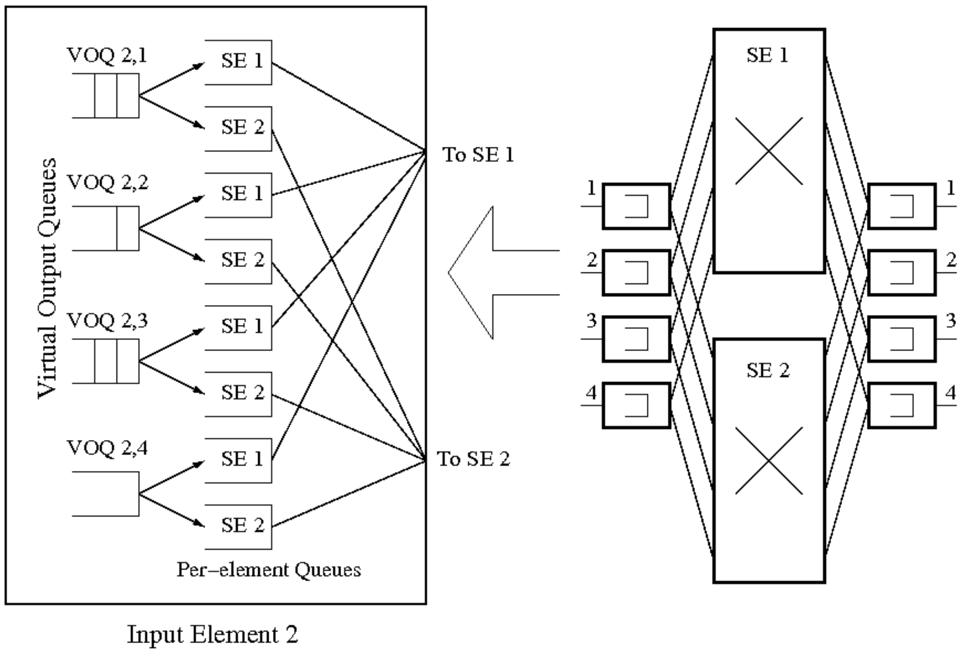


Figure 6.21: CIOQ-P: Per-element virtual output queues

to account for the wasted SE slots, since such slots are already earmarked as part of the virtual bandwidth trunks, of the respective flows with partial envelopes. Consequently, the scheduling latency becomes $2KM$, and the delay bounds in (5.7) and (5.8) hold, with $2/r$ replaced by $2K/r$. In other words, for leaky-bucket constrained input-output flows, the bound is given by

$$D_{i,j} \leq \frac{\sigma_{i,j}}{R_{i,j}} + \frac{2K}{r} \tag{6.19}$$

As with concurrent BVN decomposition (6.17), the delay with striping is worse compared to sequential dispatch. Finally, note that $\mathcal{A}$ may correspond to an SFA combination for the combined provision of QoS and optimal throughput.

**Equal Dispatch**

In order to remedy the primary issue in striping, namely, that of unbounded latency, we introduce a concurrent method called *equal dispatch* to translate $\mathcal{A}$ into a balanced matching. The benefit comes at the expense of a slightly increased implementation overhead. Instead of allowing a striping operation to balance the offered load of each input-output pair, over the $K$ space elements on an envelope basis, we explicitly distribute the load to $K$ different sets of queues, on cell arrival, and let each space element operate $\mathcal{A}$ concurrently in the cell-mode, on a single set.

The queueing structure in each memory element is modified as shown in Fig. 6.21. Instead of a single VOQ per pair $(i, j)$, we maintain a separate queue $Q_{i,j}^{(k)}$, $k \leq K$, per space element for each pair. We call these queues *space-element* (SE) VOQ. The arriving traffic for $(i, j)$ is split evenly cell by cell, using a round-robin pointer, among the $K$ queues for that pair. Therefore, if $A^{(k)}(n)$ represents the cumulative arrivals into the SE VOQs for element $k$, we obtain

$$\forall i, j, k, n, \quad A_{i,j}^{(k)}(n) = \left\lfloor \frac{A_{i,j}(n)}{K} \right\rfloor + 1_{\{A_{i,j}(n) \bmod K \geq k\}}, \tag{6.20}$$

where $A(n)$ is the arrival matrix for the input-output pairs. Clearly, by definition (2.1), the average arrival rates into $Q_{i,j}^{(k)}$, $\forall k$ is equal to $\lambda_{i,j}/K$. Therefore, the average-rate matrices for all the space elements are equal, and are given by

$$\forall k, \;\; \lambda^{(k)} = \frac{1}{K}\lambda \tag{6.21}$$

Each space element $k$ computes matchings $\pi^{(k)}$ concurrently, once per SE slot, using the given algorithm set $\mathcal{A}$, as shown in Fig. 6.22. The matchings are based only on the state of $Q^{(k)}$, and are consumed only by the latter. Essentially, such a queueing structure allows to distribute the traffic belonging to each pair $(i, j)$ to $K$ concurrent planes, each being served by a separate space element. If $D_{i,j}^{(k)}$ represents the cumulative departures for pair $(i, j)$ through $k$, then the evolution of each SE VOQ matrix may be described by $(A^{(k)}, D^{(k)}, Q^{(k)})$. These are related according to (3.2), with superscript $k$, and $D^{(k)}$ is given by

$$\forall k, n, \;\; D^{(k)}(n) \geq \sum_{m=1}^{\lfloor s\frac{n}{K} \rfloor} \pi^{(k)}(m) \tag{6.22}$$

For implementation efficiency, and to avoid cell mis-sequencing, the SE VOQs for each pair $(i, j)$ need not be physically separate. In fact, all the cells may be enqueued into the same VOQ structure as before, and only per-SE *occupancy* need to be maintained. We can keep track of such occupancies using simple counters, as in Fig. 6.22. The counters are viewed as actual queue lengths for matching (and analysis) purposes, and when $\pi_{i,j}^{(k)} = 1$ for any $k$, a cell is

**Algorithm 23.** *CIOQ-P: Equal Dispatch*

```
Initialize:                  1. For all $(i,j), i \leq N, j \leq N$, do steps 1a–d
                                1a. $Q_{i,j} \leftarrow 0$
                                1b. For all $k \leq K$, $Q^{(k)}_{i,j} \leftarrow 0$ (Per-SE occupancy)
                                1c. $P_{i,j} \leftarrow 1$ (Round-robin pointer)
                                1d. For all $k \leq K$, $\Pi^{(k)}_{i,j} \leftarrow 0$   (Batches for QoS)
                             2. $M \leftarrow$ 1/GCD($R$)

Cell Enqueue:                If cell belongs to $(i,j)$:
                             1. Increment $Q_{i,j}$   (Enqueue cell)
                             2. Increment $Q^{(P_{i,j})}$ (Distribute cell to the correct SE)
                             3. Increment $P_{i,j}$ mod $K$

Internal epoch $n = mKMs + k$:  1. For all $(i,j), i \leq N, j \leq N$, $\Pi^{(k)}_{i,j} = \min(R_{i,j}M, Q^{(k)}_{i,j})$
$(m > 0, k \leq K)$          (Create independent batches for each SE)

                             For every space element $k$ in parallel:
Every SE slot $n$:           1. $\pi^{(k)} \leftarrow$ Apply $\mathcal{A}$ on $Q^{(k)}, \Pi^{(k)}$
                             2. $Q^{(k)} \leftarrow Q^{(k)} - \pi^{(k)}$ (Decrement SE counters)

Every internal slot $n$:     1. $k \leftarrow n$ mod $K + 1$ (Select space element)
                             2. $Q \leftarrow Q - \pi^{(k)}$ (Dispatch through SE $k$)
```

Figure 6.22: CIOQ-P: Emulating a CIOQ using equal dispatch

drawn from the respective physical VOQ. Since each space element consists of an independent arbiter, running at a frequency of $s/K$ matchings per external timeslot, the run-time complexity of equal dispatch is $O(1+\frac{\xi}{K})$, if each invocation of $\mathcal{A}$ runs in $O(\xi)$ time. Therefore, the overall complexity is comparable to striping. The following straight-forward result establishes that the performance is also comparable.

**Theorem 15.** *Given a matching algorithm $\mathcal{A}$, if $\mathcal{A}$ suffices in a CIOQ switch for asymptotic 100% throughput to a set of input-output flows $\{(i,j)\}$, under some arrival process, then equal dispatch combined with a concurrent application of $\mathcal{A}$ on each space element suffices for the same, under identical arrivals, i.e,*

$$\begin{aligned}&\forall f_i, i \neq 5,\ \textit{(CIOQ, }\mathcal{A}\textit{)} \overset{T,f_i}{\simeq} \textit{(OQ, }\mathcal{A}'\textit{)},\quad s \geq s^* \quad \Rightarrow \\ &\textit{(CIOQ-P, }\{\textit{Equal Dispatch}\} \bigcup \mathcal{A}\textit{)} \overset{T,f_i}{\simeq} \textit{(OQ, }\mathcal{A}'\textit{)},\quad s \geq s^*\end{aligned}$$

*Proof.* Let $S$ be the set of flows $\{(i,j)\}$ for which $Q_{i,j}$ is stable in a $N \times N$ CIOQ switch, for arrival matrix $A(n)$, under $\mathcal{A}$ operating at $s^*$ matchings per external timeslot. Consider a $N \times N$ shadow CIOQ switch with interface rates of $1/K$, under arrival matrix $A^{(k)}(n)$, for any $k$. From (6.21), the average rates of arrival $\lambda^{(k)}$ equal $1/K$ times those into the original CIOQ switch. Normalized to the new interface capacity, the average rates are exactly equal. Therefore, the set $S$ is stable in the new CIOQ switch, under $\mathcal{A}$ operating at $s^*$ matchings per new external slot, or at $s^*/K$ matchings per old slot.

The departures through the space element $k$, for arrivals $A(n)$ into a CIOQ-P switch with unit interface rates, exactly mimic those from the shadow CIOQ switch, if $\mathcal{A}$ operates on $Q^{(k)}$ at a frequency of exactly $s^*/K$ matchings per external slot. Therefore, all the SE VOQ $\{Q^{(k)}_{i,j} | (i,j) \in S\}$ remain stable. Since this holds for any $k$, each SE VOQ that contains cells belonging to $S$ remains stable, for $s \geq s^*$. $\square$

In reality, it is easy, though voluminous, to rigorously prove Theorem 15, separately for each level of functional equivalence. For example, notice that if $\lambda$ is admissible in CIOQ-P, $\lambda^{(k)}$ would be admissible in the shadow CIOQ switch. The same is true for non-oversubscribed outputs and conforming flows. These observations may then be used for separately proving $f_2$,

$f_3$ and $f_4$ equivalence with an OQ switch, respectively. We rely on the fact that long-term queue stability, under $\mathcal{A}$, only depends upon the average rates. In addition, if $\mu$ is the average departure-rate matrix for offered rates $\lambda$ in a CIOQ switch, then, under the same matching algorithm and identical speedup, $\frac{1}{K}\mu$ would result for offered rates $\frac{1}{K}\lambda$ in a CIOQ switch with $1/K$ times the interface rate of the former.

The above theorem allows us to apply all the well-known stability results for CIOQ matchings, without the need for independent analysis, to a CIOQ-P switch. Specifically, we obtain the following results for equal dispatch (ED):

$$\begin{aligned}
&(\text{CIOQ-P, }\{\text{ED, MWM}\}) \overset{T,f_2}{\simeq} (\text{OQ, }\{\text{WC}\}),\ \ T\text{: SLLN}\\
&(\text{CIOQ-P, }\{\text{ED, LPF}\}) \overset{T,f_2}{\simeq} (\text{OQ, }\{\text{WC}\}),\ \ T\text{: i.i.d.}\\
&(\text{CIOQ-P, }\{\text{ED, Critical Matching}\}) \overset{T,f_2}{\simeq} (\text{OQ, }\{\text{WC}\}),\ \ T\text{: SLLN}\\
&(\text{CIOQ-P, }\{\text{ED, SPS Matching}\}) \overset{T,f_2}{\simeq} (\text{OQ, }\{\text{WC}\}),\ \ T\text{: SLLN, Uniform}\\
&(\text{CIOQ-P, }\{\text{ED, Maximal Matching}\}) \overset{T,f_3}{\simeq} (\text{OQ, }\{\text{WC}\}),\ \ s \geq 2,\ \ T\text{: SLLN}\\
&(\text{CIOQ-P, }\{\text{ED, SOQF, }S\}) \overset{T,f_4}{\simeq} (\text{OQ, }\{\text{WC, }S\}),\ \ s \geq 3,\ \ T\text{: SLLN } S\text{: Convergent}
\end{aligned}$$

The first two are consequences of the CIOQ results on MWM and LPF matching from [29, 79], respectively. The next four are respectively derived from Theorems 4, 5, 8, and 9. Note that SOQF will require per-SE virtual input queue (VIQ) counters. The SE $k$ VIQ for flow $(i, j)$ is incremented when a cell is received from element $k$ for pair $(i, j)$, and decremented on a round-robin basis, over the $K$ space elements, when the output scheduler $S$ serves $(i, j)$. Note also that, while equal dispatch is a concurrent method, the departure process is, in general, different from concurrent dispatch (Fig. 6.15) based on $\mathcal{A}$. For example, in concurrent SPS, the $K$ space elements together provide $K$ consecutive service turns to a pair $(i, j)$ in an interval of length $KN$. If $Q_{i,j}^{(k)} > 0$, $\forall k$, in equal dispatch with SPS, the departures will be identical. However, this may not always be the case, since, in the latter, while $K$ consecutive turns are indeed provided to $(i, j)$ in the same interval, a turn given by element $k$ can be consumed only by the corresponding SE VOQ, which may be empty even when $Q_{i,j} > 0$.

A direct proof for each of the above results, on individual matching algorithms, may be obtained as an extension of the respective CIOQ proof, applied separately on each space element and its corresponding SE VOQ system. For example, referring to critical matchings, we may chart the evolution of a phantom bucket for each space element (pg. 120), yielding the following extensions of (5.14) and (5.15):

$$\begin{aligned} \forall k,\ \bar{\mathcal{Z}}^{(k)}(t) &\leq \frac{t}{K} - \bar{D}_{\mathcal{Z}}^{(k)}(t) \\ \bar{D}_{\mathcal{Z}}^{(k)\prime}(t) &= \frac{1}{K}, \text{ if } \bar{\mathcal{Z}}^{(k)}(t) > 0 \end{aligned}$$

This leads to a negative drift on all the $K$ phantom buckets, for $s \geq 1$, leading to the stability of each SE VOQ. Similarly, referring to the $f_3$ result for maximal matching, for $s \geq 2$, (5.23) and (5.24) in the proof of Theorem 8 may be extended to each SE VOQ system as

$$\begin{aligned} \forall i,k,t\ \forall j \notin A \quad \bar{T}_{i,j}^{(k)}(t) &< \frac{2t}{K} - \bar{D}_{T_{i,j}}^{(k)}(t) \\ \bar{D}_{T_{i,j}}^{(k)\prime} &\geq \frac{2}{K}, \quad \text{whenever } \bar{Q}_{i,j}^{(k)}(t) > 0 \end{aligned}$$

Following the same reasoning as in pg. 133, this leads to the stability of all SE VOQ for non-oversubscribed outputs. Both these extensions are based on the invariant property of the respective algorithm on (6.22), and the effect of admissibility on (6.21). In light of Theorem 15, nevertheless, there is no need for such independent analyses.

Much as with striping, $\mathcal{A}$ subsumes QoS-capable algorithms, such as BVN decomposition ($s \geq 1$) and rate-shaping combined with batch-mode critical ($s \geq 1$) or maximal matching ($s \geq 2$). In contrast, however, the same sequence of templates is applied concurrently to each space element, operating on $Q^{(k)}$. For batch-mode online matchings, $K$ separate batch matrices $\Pi^{(k)}$ are assembled using reservations $\frac{1}{K}R$, essentially by batching $Q^{(k)}$ at external slot epochs $mKM$. Consequently, the scheduling latency remains $2KM$, and the delay bound for leaky-bucket shaped input-output flows is given by (6.19). As expected, $\mathcal{A}$ may also correspond to an SFA combination, as shown in Fig. 6.22, for the combined provision of optimal throughput and QoS.

To summarize, shadowing a CIOQ switch of comparable dimensions, followed by a sequential dispatch, is sufficient to inherit all the performance results of CIOQ. However, there is no benefit in matching complexity with respect to a CIOQ switch, despite using slower space elements, due to the requirement of a centralized arbiter. Concurrently applying a matching algorithm, on each space element, directly serving the VOQ system, does reduce the complexity by a factor of $K$. Unfortunately, several known algorithms do not lend easily to a concurrent version, with the exception of some template-based and well-ordered greedy methods. To overcome this limitation, striping may be used to shadow an envelope-mode CIOQ switch and inherit most of its performance results, for general matching algorithms. While striping also lowers complexity, the main disadvantage is the possibility of unbounded latency, which may be addressed using a modest speedup, or using cell-based equal dispatch at the expense of a slightly higher implementation overhead. The low-complexity concurrent and equal dispatch methods may actually be used in a CIOQ switch to implement arbitrarily complex matching algorithms, as an alternative to envelope-based matching [54]. The approach would be to compute $K$ configurations in parallel and use them, in sequence, on the single space element of CIOQ. We did not specifically consider multicast traffic, since recycling-based multicast, as in a CIOQ switch, may be directly applied to CIOQ-P, without any further enhancements.

## 6.3 General MSM Switches

A general space-memory-space (G-MSM) switch is obtained by simultaneously applying aggregation, on the memory elements, and spatial pipelining, on the space element of a CIOQ switch. As shown in Fig. 6.23, structurally speaking, this design most resembles a circuit Clos network, and hence has become synonymous with terms like Clos-network packet switch, in the literature (e.g., [85, 93, 92]). In fact, G-MSM switches are more common, in practice and in the literature, than CIOQ-A or CIOQ-P switches. We chose to address the latter in isolation, since, as shall be seen in this section, the methods to be employed in a G-MSM switch may be derived as a natural combination of those used individually by CIOQ-A and CIOQ-P designs.

A $N \times N$ G-MSM switch $(N, [MSM], P, K, s)$ contains $P$ first-stage memory elements of dimensions $N/P \times K$, $K$ instances of $P \times P$ space elements in the center, and $P$ third-stage elements of dimensions $K \times N/P$. We assume that $N/P$ is an integer and $1 < P < N$. The first-stage elements may either employ virtual element queueing (VEQ) or virtual input-output queueing (VIOQ), similar to a CIOQ-A switch, as shown in Fig. 6.2(a) and (c), respectively. If VIOQ is employed, as seen before, we may expect closer emulation of a CIOQ switch. However, the space elements are required to decompose each $N \times N$ configuration into several $P \times P$ ones.

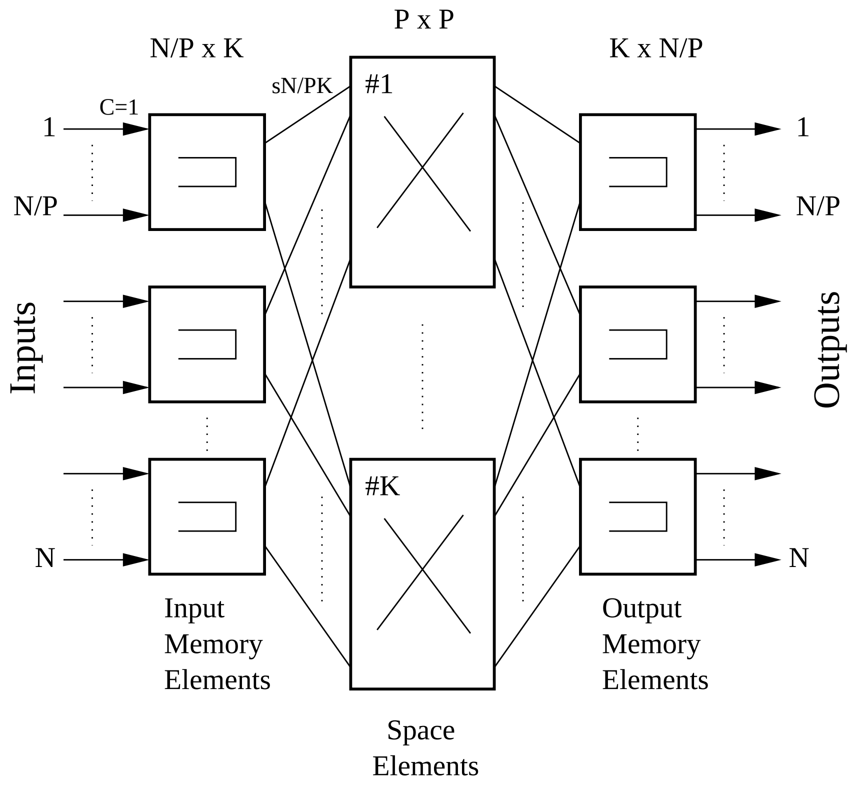


Figure 6.23: G-MSM: CIOQ switch with aggregation and pipelining

If equal dispatch is used, among the $K$ space elements, the input elements also require per-SE queueing, as in Fig. 6.21. We note, without elaboration, that the service turns given to each VEQ or VIOQ may be distributed to finer-grain flows using virtual schedulers in a hierarchical fashion, as described in Sec. 3.3.2.

As is evident, this architecture combines the benefits of CIOQ-A and CIOQ-P. Normalized to an external interface capacity of one cell per timeslot, the internal links operate at $sN/PK$. Thus, the space elements are smaller than in a CIOQ switch, but not necessarily (if $N \leq PK$) faster as in CIOQ-A. For sequential dispatch methods, the arbitration frequency remains as high as $sN/P$ matchings per external slot. However, this may be reduced to a frequency of $sN/PK$ by using concurrent or balanced matchings over all the space elements. The complexity of each matching may be further reduced by employing VEQ, and matching on the resulting smaller queue-state matrix. In addition, G-MSM retains native support for subports as in a CIOQ-A switch (pg. 181). The VEQ/VIOQ system is simultaneously served by $K$ space elements, necessitating a memory bandwidth, in general, of $(1+s)\frac{N}{P}$ times the external capacity, comparable to a CIOQ-A switch. This remains the sole disadvantage of this design.

As $s\frac{N}{P}$ cells may be transferred, per external timeslot, from/to each memory element, we refer to the time unit of $P/sN$ as an internal timeslot. In other words, there are $s\frac{N}{P}$ internal slots for every external one, each associated with a distinct $P \times P$ matching $\pi(n)$ serving the VEQ/VIOQ system. While one cell may be dispatched by an input element in each internal slot, the serving space element takes a total of $K$ internal slots, or $KP/sN$ external ones, to transfer the cell to the respective output element. We refer to this transfer time as the space element (SE) slot. As in a CIOQ-P switch, we assume that the SE slots are staggered, such that an SE slot $n$ for space element $k$ occupies the interval $[(n-1)K+k, nK+k)$ of internal slots. Let $\pi^{(k)}(n)$ be the configuration, satisfying (2.5), of space element $k$ at its SE slot $n$. Then, $\pi^{(k)}(n)$ is related to $\pi(n)$ exactly as in a CIOQ-P switch:

$$\forall n, \quad \pi(n) = \pi^{(k)}(\left\lceil \frac{n}{K} \right\rceil), \qquad \text{where } \ k = n \bmod K + 1. \tag{6.23}$$

For calculating departures, we assume that a cell leaves the first-stage element at the beginning of the SE slot. A sequential dispatch method computes $\pi(n)$ in each internal slot and assigns it to the appropriate $\pi^{(k)}(n)$, while a concurrent or balanced dispatch method computes $\pi^{(k)}(n)$ independently for each space element, using parallel arbiters, and directly provides service turns to the VEQ/VIOQ system. The complexity of each such procedure would depend on the employed queueing strategy.

### System Evolution

In virtual element queueing, a cell belonging to input-output flow $(i, j)$ is enqueued into VEQ $(\lceil \frac{i}{N/P} \rceil, \lceil \frac{j}{N/P} \rceil)$. The evolution of the VEQ system may be described by $P \times P$ matrices $(A(n), D(n), Q(n))$, denoting the cumulative arrivals, cumulative departures and current VEQ length, respectively. In other words, $Q(n)$ has the same dimensions as $\pi(n)$. If the $N \times N$ matrix $A^*(n)$ denotes the cumulative arrivals for each input-output flow until external slot $n$, with associated average rates $\lambda^*$, then $(A^*, \lambda^*)$ is related to $(A, \lambda)$ as in (6.1). Furthermore, if $\lambda^*$ is admissible, then the line and column sums of $\lambda$ sum up to no greater than $N/P$ (6.2). If per-SE queueing is employed, the cumulative arrivals and average rates into the SE queues continue to be governed by (6.20) and (6.21). It is easy to verify that the queue evolution is constrained by:

$$\begin{aligned}
\forall n, \quad Q(n) &= A(n) - D(n) && (6.24)\\
D(n) &= \sum_{m=1}^{s\frac{N}{P}n} \pi(m) &&\\
&\geq \sum_{k=1}^{K} \sum_{m=1}^{\lfloor s\frac{N}{PK}n \rfloor} \pi^{(k)}(m) && (6.25)
\end{aligned}$$

Note that VEQ does not rule out hierarchical output queueing as in Fig. 6.9. It merely implies that the matching is performed on an element-pair basis. Also, note that (6.25) is derived from (6.23), considering only SE slots that are completed in all space elements, for notational convenience.

In virtual input-output queueing, a cell belonging to input-output flow $(i, j)$ is enqueued into VIOQ $(i, j)$. The evolution of the VIOQ system may be described by $N \times N$ matrices $(A(n), D(n), Q(n))$, defined similarly as above. The main difference, with respect to VEQ, is that the $N \times N$ matrix $Q(n)$ is used to compute a $P \times P$ matching $\pi(n)$. This is achieved essentially by shadowing some $N \times N$ switch to derive a configuration $\pi^*$, followed by a decomposition into several $P \times P$ configurations to be used in a sequence. Accordingly, the queue evolution is constrained by (6.24), and a combination of (6.7) and (6.14):

$$\forall i, j \leq P,\ \forall n \quad \sum_{k=\frac{N}{P}(i-1)+1}^{\frac{N}{P}i} \ \sum_{l=\frac{N}{P}(j-1)+1}^{\frac{N}{P}j} D_{k,l}(n) \ \geq \ \sum_{k=1}^{K} \sum_{k=1}^{\lfloor s\frac{N}{PK}n \rfloor} \pi_{i,j}(k) \qquad (6.26)$$

We shall not explicitly analyze the departures and queue states in a G-MSM, instead relying on extending and combining the individual results on CIOQ-A and CIOQ-P. The above constraints on the system evolution were presented for the sake of completeness.

## Matching Algorithms: Alternatives

Matching algorithms for a G-MSM switch are a combination of the shadow-and-decompose method (Fig. 6.4) or direct VEQ matching (Fig. 6.7) to account for the aggregation transformation, and one of sequential dispatch (Fig. 6.14), concurrent dispatch (Fig. 6.15), striping

| $\mathcal{A}$ / $\mathcal{P}$ | **Shadow-Decompose** | **VEQ Matching** |
|---|---|---|
| **Sequential Dispatch** | $f_5, O(h(N) + \frac{N}{P}g(P))$ | $f_2, O(\frac{N}{P}h(P))$ |
| **Striping/Equal Dispatch** | $f_4, O(\frac{1}{K}h(N) + \frac{N}{PK}g(P))$ | $f_2, O(\frac{N}{PK}h(P))$ |
| **Concurrent Dispatch** | $f_3, O(\frac{1}{K}h(N) + \frac{N}{PK}g(P))$ | $f_2, O(\frac{N}{PK}h(P))$ |

Table 6.4: G-MSM: Equivalence with CIOQ, and complexity for various combinations

(Fig. 6.20) or equal dispatch (Fig. 6.22) to account for the pipelining transformation. Recall (pg. 85) that a G-MSM may be obtained by starting with a CIOQ switch and applying aggregation and pipelining as follows:

$$(N, [MSM], N, 1, s^*) \xrightarrow{a} (N, [MSM], P, 1, s_1) \xrightarrow{p} (N, [MSM], P, K, s) \quad \text{or}$$
$$(N, [MSM], N, 1, s^*) \xrightarrow{p} (N, [MSM], N, K, s_2) \xrightarrow{a} (N, [MSM], P, K, s)$$

Let $\mathcal{A}$ be the algorithm set that provides $f_a$ equivalence with the original CIOQ switch (with speedup $s^*$), following the arrow marked $a$, using speedup $s_a$. Similarly, let $\mathcal{P}$ be the set that provides $f_p$ equivalence with the CIOQ switch, following the arrow marked $p$, using speedup $s_p$. It follows that a combined application of $\mathcal{A}$ and $\mathcal{P}$ will provide the lower of the two equivalences, using the higher of the two speedup values. This leads us to the following informal[4] result.

**Proposition 4.** *Let a reference CIOQ switch employ $\mathcal{A}'$ with speedup $s^*$. Then,*

$$\begin{aligned}
&\textit{(CIOQ-A, } \mathcal{A}\textit{)} \overset{T,f_a}{\simeq} \textit{(CIOQ, } \mathcal{A}'\textit{)}, \quad s \geq s_a \\
\wedge \quad &\textit{(CIOQ-P, } \mathcal{P}\textit{)} \overset{T,f_p}{\simeq} \textit{(CIOQ, } \mathcal{A}'\textit{)}, \quad s \geq s_p \quad \Rightarrow \\
&\textit{(G-MSM, } \mathcal{A} \bigcup \mathcal{P}\textit{)} \overset{T,f}{\simeq} \textit{(CIOQ, } \mathcal{A}'\textit{), where } s \geq \max(s_a, s_p),\ f \leq \min(f_a, f_p)
\end{aligned}$$

The above provides an intuition behind the achievable performance for various combinations. If the reference CIOQ switch is functionally equivalent with some ideal switch, at a level no higher than $f$, the employment of $\mathcal{A} \cup \mathcal{P}$, in G-MSM, allows to inherit that equivalence, due to the containment (4.5) and transitivity (4.6) properties.

[4]This result is kept informal because the calculus of functional equivalence is not mature enough to account for a sequence of transformations. Formal results for individual instances of $\mathcal{A}$ and $\mathcal{P}$ are easily obtained on a case-by-case basis.

Table. 6.4 shows the gamut of performance, and associated run-time complexity, for various such combinations, based on the individual results on CIOQ-A and CIOQ-P. The reference for equivalence is a cell-based CIOQ switch. Here $O(h(n))$ denotes the complexity of the matching algorithm being emulated, on an $(n \times n)$ queue-state matrix, and $O(g(n))$ is the complexity of the chosen algorithm that decomposes a $(n \times n)$ aggregate matrix. For instance, $h(n)$ equals 1 for SPS, $n$ for EREW maximal, $n^2$ for maximal, $n^{2.5}$ for critical, $n^2 \log n$ for SOQF and $n^3$ for maximum weight matching. Similarly, $g(n)$ equals $n^2$ for maximal and $n^{2.5}$ for critical decomposition. The combinations in the first column require VIOQ, as opposed to VEQ in the second column. Furthermore, the ones in the second row require per-SE queueing.

Notice that, for asymptotic $100\%$ throughput to admissible traffic, matching on virtual element queues suffices, with the concurrent and balanced dispatch methods affording the lowest complexity. For uniform traffic, concurrent SPS (or round-robin) is sufficient with $s \geq 1$, while for non-uniform traffic, a concurrent EREW maximal (3DMM) matching suffices for the same, with $s \geq 2$. Lower speedup is made possible by critical matching on SE-VEQ, striping the results of a critical matching on VEQ, or more simply, a critical matching followed by sequential dispatch, at the expense of higher complexity. For isolating instability in the presence of inadmissible traffic, however, VIOQ is required for known matchings. Note that these are based on analytically proven results on CIOQ switches, and the theorems on aggregation and pipelining, from the previous section, that allow to transfer those results to a G-MSM.

### 6.3.1 Shadowing Approaches

Fig. 6.24 shows four methods for shadowing a $N \times N$ switch, followed by a decomposition into a sequence of $P \times P$ configurations, suitable for a G-MSM switch. Instead of providing the precise algorithms, we have merely shown the flow of events for each of the combinations. The reader is referred to the respective CIOQ-A and CIOQ-P algorithms for further details. Each method requires virtual input-output queueing. In addition, the equal-dispatch method requires per-space element VIOQ. Note that if the cells served by the shadowing operation are immediately transferred to temporary on-chip FIFOs, the external memory bandwidth for all such techniques goes down to $(1 + s^*)$ times the interface capacity.

**Algorithm 24.** *G-MSM: Shadow and Decompose*

```
(a) Shadow-$\mathcal{A}$-Decompose-Sequential Dispatch:
    1. $\pi^*(m) \leftarrow$ Apply $\mathcal{A}$ on VIOQ at epochs $mT, \;\; T = \frac{sN}{s^*P}, m > 0$
    2. Decompose $\pi^*(m)$ into $\pi(n), \;\; n = mT, mT+1, \ldots, (m+1)T-1$
    3. $\pi^{(k)}(\lceil \frac{n}{K} \rceil) \leftarrow \pi(n), \;\; k = n \bmod K + 1$

(b) Shadow-$\mathcal{A}$-Decompose-Stripe:
    1. $\pi^*(m) \leftarrow$ Apply $\mathcal{A}$ on full envelopes in VIOQ at epochs $mKT, \;\; m > 0$
    2. Decompose $\pi^*(m)$ into $\pi_e(l), \;\; l = mT, mT+1, \ldots, (m+1)T-1$
    3. $\pi(n) \leftarrow \pi_e(l), \;\; n = lK, lK+1, \ldots, (l+1)K-1$
    4. $\pi^{(k)}(\lceil \frac{n}{K} \rceil) \leftarrow \pi(n), \;\; k = n \bmod K + 1$

(c) Equal Dispatch-Shadow-$\mathcal{A}$-Decompose:
    For all space elements $k$ in parallel:
    1. $\pi^{*(k)}(m) \leftarrow$ Apply $\mathcal{A}$ on SE-VIOQ at SE epochs $mT, \;\; m > 0$
    2. Decompose $\pi^{*(k)}(m)$ into $\pi^{(k)}(n)$, SE slot $n = mT, \ldots, (m+1)T-1$

(d) Concurrent-$\mathcal{A}$-Decompose:
    For all space elements $k$ in parallel:
    1. $\pi^{*(k)}(m) \leftarrow$ Apply $\mathcal{A}$ on VIOQ at SE epochs $mT, \;\; m > 0$
    2. Decompose $\pi^{*(k)}(m)$ into $\pi^{(k)}(n)$, SE slot $n = mT, \ldots, (m+1)T-1$
```

Figure 6.24: G-MSM: Combination methods based on shadowing and decomposition

As shown in Fig. 6.24(a), we may shadow a CIOQ switch employing $\mathcal{A}$ (Fig. 6.4), with speedup $s^*$, followed by a critical ($s \geq s^*$) or maximal decomposition ($s \geq s^*(2 - \frac{P}{N})$) to yield a sequence of configurations $\pi(n)$, using a centralized arbiter. These configurations may then be assigned to the staggered SE slots of the $K$ space elements, using sequential dispatch (Fig. 6.14) without the need for additional speedup. From Theorems 10 and 12, the combination exactly emulates the reference CIOQ switch. The proofs of these theorems may be put together to give us the following result.

**Corollary 19.** *(CIOQ, $\mathcal{A}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$),* $s \geq s^* \quad \Rightarrow$
*(G-MSM, $\{$Shadow-$\mathcal{A}$, Critical Decomposition, Sequential Dispatch$\}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$),* $s \geq s^*$

Maximal decomposition yields a similar result with about twice the speedup. Note that this result holds any general $\mathcal{A}$, providing the strongest throughput performance for a G-MSM switch. For example, the switch provides asymptotic 100% throughput for all admissible traffic for $\mathcal{A} = \{\text{Critical Matching}\}$, without speedup. Similarly, $\mathcal{A} = \{\text{Maximal Matching}\}$ suffices with $s \geq 2$. The method also subsumes rate-shaping with batch-mode maximal or critical matching for QoS. The primary drawback is that there is no complexity advantage despite using smaller and slower space elements. For example, critical matching would run in $O(N^{2.5})$ time, and maximal matching in $O(N^2 + NP^{1.5})$ time.

Decomposing balanced matchings, including striping and equal dispatch, allows to support a general $\mathcal{A}$, while retaining most of the throughput performance, namely, all the functional equivalence results of the reference CIOQ switch except exact emulation. As shown in Fig. 6.24(b), we may shadow a CIOQ switch (Fig. 6.4) employing $\mathcal{A}$ on full envelopes of $K$ cells, followed by a maximal or critical decomposition. The resulting configurations are uniformly assigned (Fig. 6.20) to each space element, each of which transfers a single cell for a connected envelope. Alternatively, to avoid the unbounded latency issue in striping, arriving cells may be equally distributed to each space element (Fig. 6.22). The $K$ space elements may then concurrently shadow a CIOQ switch, employing $\mathcal{A}$, with interface rates of $1/K$, as shown in Fig. 6.24(c). $\mathcal{A}$ subsumes QoS-capable matching algorithms, with a scheduling latency of $K$ times that of the sequential dispatch method. Notice that, with respect to the latter, we reduce the run-time complexity by a factor of $K$. Theorem 10, with 14 and 15, gives us the following

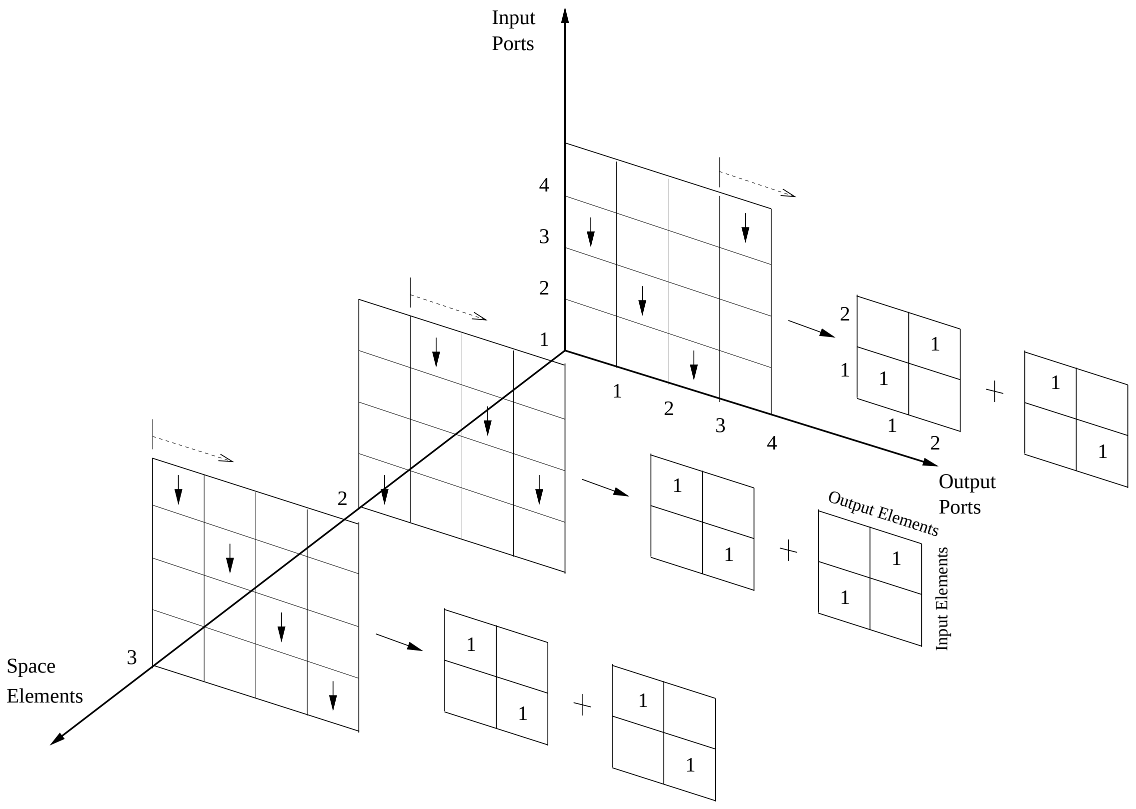


Figure 6.25: G-MSM: 3D maximal matching with critical decomposition, $P = 2, K = 3$

results:

**Corollary 20.** $\forall f_i, i \neq 5$, *(CIOQ, $\mathcal{A}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$)*, $s \geq s^* \quad \Rightarrow$
*(G-MSM, $\{$Shadow-$\mathcal{A}$, Critical Decomposition, $K$-Striping$\}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$)*, $s \geq s^*$

**Corollary 21.** $\forall f_i, i \neq 5$, *(CIOQ, $\mathcal{A}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$)*, $s \geq s^* \quad \Rightarrow$
*(G-MSM, $\{$Equal Dispatch, Shadow-$\mathcal{A}$, Critical Decomposition$\}$)* $\overset{T,f_i}{\simeq}$ *(OQ, $\mathcal{A}'$)*, $s \geq s^*$

The lowest implementation overhead, and a comparable run-time complexity as the above two methods, may be achieved by concurrently running $\mathcal{A}$ (Fig. 6.15) directly on the VIOQ, for each of the space elements, followed by parallel decomposition operations, as shown in Fig. 6.24(d). This essentially shadows a CIOQ-P switch using concurrent dispatch. An example is illustrated in Fig. 6.25, in which a concurrent 3DMM (Fig. 6.18) is followed by a critical decomposition on each space element.

Unfortunately, this approach is not applicable to a general $\mathcal{A}$. Accordingly, the results so far are restricted to asymptotic 100% throughput for uniform traffic using concurrent SPS, and to partially admissible traffic using 3DMM. The utility of concurrent SPS (Fig. 6.17) on VIOQ, with a complexity dominated by decomposition, is debatable since the same performance may be had using concurrent SPS on VEQ, with a lower run-time. Among established results for concurrent dispatch, 3DMM provides the best performance. Specifically, from Theorem 13, 3DMM may be used to provide 100% throughput to non-oversubscribed outputs, i.e., $f_3$ equivalence with a work-conserving OQ switch.

**Corollary 22.** *(G-MSM, $\{$3DMM, Critical Decomposition$\}$) $\overset{T,f_3}{\simeq}$ (OQ, $\{$WC$\}$), $s \geq 2$, $K \leq N$, $T$: SLLN*

The run-time complexity of $O(NP^{1.5}/K)$ is again dominated by the critical decomposition. In fact, with a higher speedup of $s \geq 4 - 2\frac{P}{N}$, the complexity may be brought down to $O(\frac{N}{K})$, using an EREW maximal matching to decompose the $N \times N$ shadow configurations.

### 6.3.2 Matching on Virtual Element Queues

Matching on virtual element queues, as shown in Fig. 6.26, reduces the matching complexity, compared to the corresponding shadowing schemes, by taking advantage of the smaller size of the space elements. The basic approach is to compute matchings based directly on the $P \times P$ VEQ matrix, as in CIOQ-A (Fig. 6.7), combined with a sequential, concurrent or balanced dispatch to distribute the matchings over the $K$ space elements. Again, we have not shown the precise algorithms here, and the reader is referred to the respective CIOQ-A and CIOQ-P algorithms for further details. The main disadvantage of any VEQ-based method is that the throughput performance, for known results, is unsatisfactory in the presence of inadmissible traffic. Nevertheless, it suffices for asymptotic 100% throughput to admissible arrivals, and to provide virtual bandwidth trunks specified by an admissible rate-reservation matrix.

As shown in Fig. 6.26(a), we may directly apply the chosen matching algorithm $\mathcal{A}$ on the VEQ matrix, once in every internal timeslot, to yield a sequence of $P \times P$ configurations. These may then be assigned to the staggered SE slots of the $K$ space elements, using sequential dispatch

**Algorithm 25.** *G-MSM: VEQ Matching*

```
(a) VEQ-$\mathcal{A}$-Sequential Dispatch:
    1. $\pi(n) \leftarrow$ Apply $\mathcal{A}$ on VEQ at each internal slot $n$, $n > 0$
    2. $\pi^{(k)}(\lceil \frac{n}{K} \rceil) \leftarrow \pi(n), \;\; k = n \bmod K + 1$

(b) VEQ-$\mathcal{A}$-Stripe:
    1. $\pi_e(m) \leftarrow$ Apply $\mathcal{A}$ on full envelopes in VEQ at epochs $mK, \;\; m > 0$
    2. $\pi(n) \leftarrow \pi_e(m), \;\; n = mK, mK+1, \ldots, (m+1)K - 1$
    3. $\pi^{(k)}(\lceil \frac{n}{K} \rceil) \leftarrow \pi(n), \;\; k = n \bmod K + 1$

(c) Equal Dispatch-VEQ-$\mathcal{A}$:
    For all space elements $k$ in parallel:
    1. $\pi^{(k)}(n) \leftarrow$ Apply $\mathcal{A}$ on SE-VEQ at each SE slot $n, \;\; n > 0$

(d) Concurrent-VEQ-$\mathcal{A}$:
    For all space elements $k$ in parallel:
    1. $\pi^{(k)}(n) \leftarrow$ Apply $\mathcal{A}$ on VEQ at each SE slot $n, \;\; n > 0$
```

Figure 6.26: G-MSM: Combination methods for matching on virtual element queues

(Fig. 6.14), without the need for additional speedup. From Theorems 11 and 12, the combination ensures VEQ stability for admissible traffic, provided $\mathcal{A}$ ensures the same in a CIOQ switch. The proofs of these theorems may be put together to yield the following result.

**Corollary 23.** *(CIOQ, $\mathcal{A}$) $\overset{T,f_2}{\simeq}$ (OQ, $\{$WC$\}$), $s \geq s^*$ $\Rightarrow$*
*(G-MSM, $\mathcal{A} \bigcup \{$Sequential Dispatch$\}$) $\overset{T,f_2}{\simeq}$ (OQ, $\{$WC$\}$), $s \geq s^*$*

For example, a critical matching or MWM, without speedup, or maximal matching with $s \geq 2$, on the VEQ matrix, followed by a sequential dispatch of the configurations to the $K$ space elements, suffices for queue stability. Similarly, a SPS (or a single-step round-robin) suffices for uniform traffic. The complexity of these methods remain the same as in a CIOQ-A (Table 6.3) as these methods do not rely on space element parallelism.

For a slightly higher implementation overhead, balanced matchings, including striping

(Fig. 6.20) and equal dispatch (Fig. 6.22), allows to provide the same performance, while leveraging parallelism to reduce the complexity. For example, cells may be equally dispatched, on arrivals, to per-space element VEQ, and concurrent arbiters may run critical/MWM or maximal matching on the latter, to provide 100% throughput. The run-time complexity then goes down by a factor of $K$, with respect to the values in Table. 6.3. Theorem 11, with 14 and 15, gives us the following results:

**Corollary 24.** *(CIOQ, $\mathcal{A}$)* $\overset{T,f_2}{\simeq}$ *(OQ, $\{$WC$\}$),* $s \geq s^*$ $\Rightarrow$
*(G-MSM, $\mathcal{A} \bigcup \{K$-Striping$\}$)* $\overset{T,f_2}{\simeq}$ *(OQ, $\{$WC$\}$),* $s \geq s^*$

**Corollary 25.** *(CIOQ, $\mathcal{A}$)* $\overset{T,f_2}{\simeq}$ *(OQ, $\{$WC$\}$),* $s \geq s^*$ $\Rightarrow$
*(G-MSM, $\mathcal{A} \bigcup \{$Equal Dispatch$\}$)* $\overset{T,f_2}{\simeq}$ *(OQ, $\{$WC$\}$),* $s \geq s^*$

Since the best results for virtual element queueing are restricted to $f_2$ equivalence with an OQ switch, it is an ideal candidate for concurrent matching. In fact, VEQ stability may be achieved, at a complexity comparable to striping and equal dispatch, without the added implementation overhead of the latter. Provided $K \leq N$, a 3DMM operating concurrently on the VEQ matrix, with $s \geq 2$, is sufficient for 100% throughput to any admissible traffic. This result may be easily established by combining the proofs of Theorems 11 and 13. The resulting complexity remains $O(\frac{N}{K})$ as in shadowing and decomposing a $N \times N$ 3DMM (pg. 214), but the speedup required is lower. For uniform traffic, a concurrent[5] SPS matching suffices for 100% throughput, as a consequence of Theorem 11 and Corollary 17. The resulting complexity is $O(\frac{N}{PK})$ on the external clock. Note that, for non-uniform traffic, with $s < 2$ or $K > N$, striping and equal dispatch remain viable alternatives.

**Corollary 26.** *(G-MSM, $\{$3DMM$\}$)* $\overset{T,f_2}{\simeq}$ *(OQ, $\{$WC$\}$),* $s \geq 2$*,* $K \leq N$*, T: SLLN*

**Corollary 27.** *(G-MSM, $\{$Concurrent SPS$\}$)* $\overset{T,f_2}{\simeq}$ *(OQ, $\{$WC$\}$), T: SLLN, Uniform*

[5]As a historical note, the Atlanta/$\pi$ chipsets employed an iterative variant, i.e., several steps, of SPS on virtual-element state, combined with both sequential and concurrent dispatch, in different versions. A speedup $s > 1$ was typically used, combined with input-output pair backpressure, to better serve non-uniform traffic and account for the finite buffers. All the performance results were based on simulations, which provided the inspiration for the analytical work here.

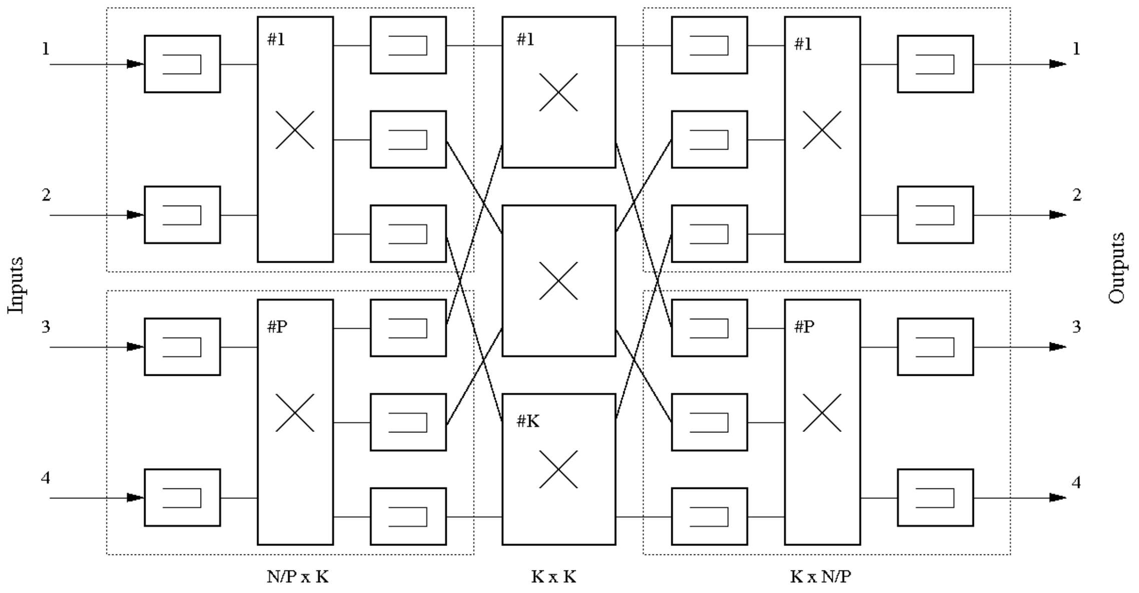


Figure 6.27: Recursive G-MSM switch: $N = 4, P = 2, K = 3$

### 6.3.3 Recursive G-MSM

The sole disadvantage of the G-MSM design is its relatively high memory bandwidth, which in general equals $\frac{N}{P}$ times that of a CIOQ switch. In practice, this is addressed by implementing the VIOQ/VEQ system as small on-chip queues [19, 21], with backpressure between stages. Specifically, the queues in the output elements block input-output pairs whose occupancy exceed a certain threshold. Similarly, the queues in the input elements send feedback to each port processor (Fig. 2.1), which contains the external memory to buffer the backlog. In such a scheme, the memory bandwidth becomes comparable to a CIOQ switch. Unfortunately, analytical results on the throughput performance of such backpressure-based schemes have remained elusive[6].

Another intriguing alternative, which retains the performance results of a CIOQ switch, at least for admissible traffic, is to recursively construct a G-MSM switch, as shown in Fig. 6.27. We start with a G-MSM switch $(N, [MSM], P, K, s)$, employing an equal dispatch method and a well-chosen matching algorithm $\mathcal{A}$, applied concurrently on the per-space element queues by the $K$ central elements. Let $K$ be chosen such that $\frac{N}{P} \leq K$. For the sake of this discussion, let the

[6]Some of the ongoing research work on buffered crossbars may be applicable, to remedy this situation

input memory elements of the G-MSM switch employ virtual element queueing. Furthermore, the SE queues are kept *physically* separate in external memory. From Corollary 25, if the $K$ space elements apply critical matching, with $s \geq 1$, or maximal matching, with $s \geq 2$ on the respective SE-VEQ matrices, all the queues remain stable for admissible traffic. The memory elements of this base G-MSM switch are shown as dotted in Fig. 6.27.

We may then replace each of the memory elements of the original G-MSM design with a CIOQ (or another G-MSM) switch. Specifically, the first-stage memory elements are replaced by $\frac{N}{P} \times K$ CIOQ switches, and the third-stage by $K \times \frac{N}{P}$ switches. The resulting design is a seven-stage multi-path switch, which is not fully connected. The main task of stage-2 space elements is to ensure that arriving cells, for each input-output pair, are equally dispatched to each stage-4 space element. Each stage-3 memory element has a single output, corresponding to one of the $K$ central elements. Accordingly, the first-stage elements implement virtual output queueing, where an output refers to a central element, and not the eventual output. Thus, each stage-2 space element performs a matching on a $\frac{N}{P} \times K$ VOQ matrix. Let the speedup required for this operation be $s'$. The third-stage elements implement virtual element queueing, where an element refers to a dotted block. The central space elements apply $\mathcal{A}$ concurrently on the respective $P \times P$ VEQ matrices, with the same speedup $s$ as in a G-MSM switch. Stages 5, 6, and 7 are mirror images of 3, 2, and 1, respectively.

Notice that the links between stages 1-2, 2-3, 5-6, and 6-7 operate at a capacity of $s'$, normalized to the external link rate. The links between stages 3-4 and 4-5 operate at $s\frac{N}{PK}$ as in G-MSM. Therefore, the memory bandwidth required in the third and fifth stage elements equal $(s' + s\frac{N}{PK})$. Similarly, the first and seventh stage elements require $(1 + s')$. These are now free of the multiplicative factor imposed by aggregation, in G-MSM, and compare favorably with the memory bandwidth of a CIOQ switch, inspite of using much smaller elements. Provided the matching algorithms in the second and sixth-stage space elements ensure queue stability in the preceding stages, the switch continues to provide asymptotic 100% throughput. The benefits of recursion come at the cost of possible cell mis-sequencing–a common issue in all multi-path switches, which we address separately in Sec. 7.4. This situation arises because the $K$ SE-VEQs, for each element pair, need to be kept physically separate to make recursion feasible.

| Stage | Type | Method |
|---|---|---|
| 1 | Memory | Route: For each input-output pair $(i, j)$, equal dispatch to $K$ destinations |
| | | Queue: Enqueue into VOQ $(i, k)$, one for each stage-4 element |
| 2 | Space | SPS matching on stage-1 VOQ |
| 3 | Memory | Enqueue into VEQ, one for each dotted block in Fig. 6.27 |
| 4 | Space | Apply $\mathcal{A}$ (concurrently) on respective stage-3 VEQ matrix |
| | | $\mathcal{A}$: Maximal ($s \geq 2$), Critical/MWM ($s \geq 1$) |
| 5 | Memory | Enqueue into VOQ $(k, j)$, $k$ : Stage-4 element, $j$: Output |
| 6 | Space | SPS matching on stage-5 VOQ |
| 7 | Memory | Dispatch cell (memory is redundant, for SPS in stage 6) |

Table 6.5: Combination method for a 7-stage recursive G-MSM switch

We now show that, for admissible traffic, SPS matching with $s' = 1$, slightly adjusted to account for the asymmetric queue-state matrices, suffices in the second and sixth stages. Let the $\frac{N}{P} \times K$ matrix $\lambda^{(2,p)}$ denote the average rates into the virtual output queues, of the first stage, connected to stage-2 space element $p$. Similarly, let the $K \times \frac{N}{P}$ matrix $\lambda^{(6,p)}$ denote the average rates into the VOQ, of the fifth stage, connected to stage-6 space element $p$. Furthermore, let $\lambda$ be an admissible (2.2) $N \times N$ average-rate matrix for the entire system. Then, we have

$$\begin{aligned} \forall k \leq K,\ \forall p \leq P, \quad \lambda^{(2,p)}_{i,k} &= \frac{1}{K}\sum_{j=1}^{N} \lambda_{i,j} \\ &\leq \frac{1}{K}, \qquad \text{where } \frac{N}{P}(p-1) < i \leq \frac{N}{P}p \end{aligned}$$

The first equality is due to equal dispatch, and the second line holds irrespective of the admissibility of $\lambda$, due to the physical limitation of the input links. Since each entry is no greater than $1/K$, SPS (Theorem 5) suffices to keep the first stage stable. Similarly, for the stage-6 space element,

$$\begin{aligned} \forall k \leq K,\ \forall p \leq P, \quad \lambda^{(6,p)}_{k,j} &= \frac{1}{K}\sum_{i=1}^{N} \lambda_{i,j} \\ &\leq \frac{1}{K}, \qquad \text{where } \frac{N}{P}(p-1) < j \leq \frac{N}{P}p \end{aligned}$$

Here, the second row utilizes the fact that $\lambda$ is admissible. Due to the above, SPS with $s' = 1$ suffices to keep the fifth stage stable, ensuring 100% throughput to admissible traffic. Note that, for inadmissible traffic, the column sums of $\lambda^{(6,p)}$ remain lower than 1 for non-oversubscribed outputs. However, the sum of each *row* may exceed 1, hampering the deployment of maximal or SOQF matching for the isolation of instability. In fact, instability at a single output spreads uniformly across all the rows. Table. 6.5 summarizes the methods employed in each stage. Above, we have essentially proved the following proposition.

**Proposition 5.** *The combination method in Table. 6.5 is sufficient for asymptotic 100% throughput to admissible arrivals in a recursive G-MSM switch, provided $\mathcal{A}$ suffices for the same in a CIOQ switch.*

## 6.4 Summary and Discussion

We conclude our coverage of single-path buffered Clos switches by summarizing the main results. We showed how a CIOQ switch may be converted into more general three-stage architectures, while retaining most (depending on the employed methods) of its throughput and QoS performance. Specifically, the transformations of aggregation, yielding a CIOQ-A (Sec. 6.1) switch, spatial pipelining, yielding CIOQ-P (Sec. 6.2), or both, yielding a G-MSM switch (Sec. 6.3) were presented and analyzed, as recipes for such architectures. These designs, especially the G-MSM one, are already found in the industry and in the literature. Most of the performance results, however, are based on simulation studies of algorithms tailor-made for such designs. We have instead presented *methods* that apply the well established (and the new ones from Chapter 5) results of CIOQ matching algorithms, and *analytically* showed them to be sufficient for throughput and QoS in single-path switches. In fact, the main message of this chapter is that there is no need, for the most part, to individually propose and analyze matching algorithms for such switches. Fig. 6.28 shows our results, in terms of the functional equivalence framework of Chapter 4.

The CIOQ-A switch allows us to use smaller albeit faster space elements, compared to a CIOQ switch of the same dimensions. We showed that such a switch retains all the performance results of CIOQ matching algorithms, by using virtual input-output queueing (Sec. 6.1.1) and the shadow-and-decompose method (Theorem 10). Optical space elements are perfect candidates for such designs, due to their higher speeds but smaller sizes, compared to electronic

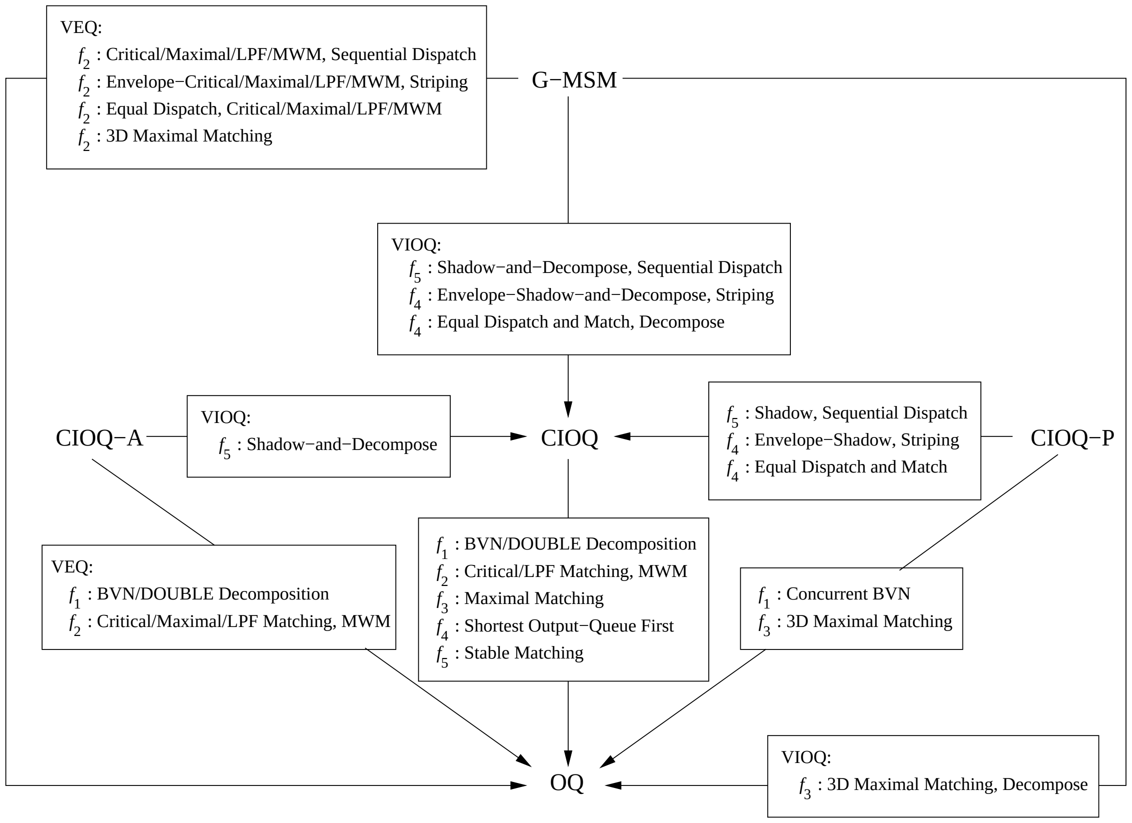

VEQ:
$f_2$ : Critical/Maximal/LPF/MWM, Sequential Dispatch
$f_2$ : Envelope−Critical/Maximal/LPF/MWM, Striping
$f_2$ : Equal Dispatch, Critical/Maximal/LPF/MWM
$f_2$ : 3D Maximal Matching
G−MSM
VIOQ:
$f_5$ : Shadow−and−Decompose, Sequential Dispatch
$f_4$ : Envelope−Shadow−and−Decompose, Striping
$f_4$ : Equal Dispatch and Match, Decompose
CIOQ−A
VIOQ:
$f_5$ : Shadow−and−Decompose
CIOQ
$f_5$ : Shadow, Sequential Dispatch
$f_4$ : Envelope−Shadow, Striping
$f_4$ : Equal Dispatch and Match
CIOQ−P
VEQ:
$f_1$ : BVN/DOUBLE Decomposition
$f_2$ : Critical/Maximal/LPF Matching, MWM
$f_1$ : BVN/DOUBLE Decomposition
$f_2$ : Critical/LPF Matching, MWM
$f_3$ : Maximal Matching
$f_4$ : Shortest Output−Queue First
$f_5$ : Stable Matching
$f_1$ : Concurrent BVN
$f_3$ : 3D Maximal Matching
OQ
VIOQ:
$f_3$ : 3D Maximal Matching, Decompose

Figure 6.28: Functional equivalences for single-path buffered Clos switches

crossbars. To address their high re-configuration overhead, we may combine shadowing with batch-mode decomposition while retaining the same performance (Corollary 11), at the expense of a higher latency. A lower matching complexity results by employing virtual element queueing (Sec. 6.1.1) and directly applying the base CIOQ matching algorithm on such queues. We showed (Theorem 11 and its derivative results) that this suffices for QoS and 100% throughput to admissible traffic. Providing optimal throughput in the presence of inadmissible traffic remains an open problem for VEQ. Extensions for natively supporting subports, with heterogeneous rates, and increasing the efficiency of recycling-based multicast support were also briefly covered.

A CIOQ-P switch provides the means to use several instances of relatively slower space elements, of the same size, compared to a CIOQ switch. We showed (Theorem 12) that such a design exactly emulates a CIOQ switch, with any base matching algorithm, using the sequential dispatch method, thereby inheriting its performance. This method requires a centralized arbiter and does not afford a lower arbitration frequency to go with the slower space elements. We proposed three methods, namely, concurrent matching, striping and equal dispatch to proportionately reduce the frequency. We showed that striping (Theorem 14) the outcome of a general envelope-based CIOQ algorithm, as well as explicitly distributing the offered load uniformly to all the space elements, i.e., equal dispatch (Theorem 15), retains almost all the performance of a CIOQ switch, except exact emulation. The tradeoff with striping is the possibility of unbounded latency, and with equal dispatch is a higher implementation overhead. Both provide similar QoS performance as a CIOQ switch, but with a significantly higher scheduling latency.

We recognized the benefit of allowing for a low arbitration frequency without being hampered by the tradeoffs of equal dispatch and striping, and proposed the concurrent dispatch method to realize it. Unfortunately, many of the known matching algorithms do not lend themselves to such an implementation. On the positive side, template-based schemes as well as orderly and greedy methods, in general, do allow a meaningful concurrent realization. Specifically, we showed (Corollary 16) that BVN decomposition may be used for bandwidth guarantees, and a concurrent SPS matching (Corollary 17) for 100% throughput to uniform traffic. We adapted the EREW maximal matching for a CIOQ switch to a three-dimensional greedy maximal matching (3DMM) algorithm, and proved its sufficiency (Theorem 13) for 100% throughput to non-uniform, including partially admissible, traffic, under certain restrictions on the dimensions of the elements. Providing better functional equivalence with an OQ switch, without such restrictions, remains an open problem for concurrent dispatch.

We claim here that the low-frequency methods of concurrent dispatch and equal dispatch, for CIOQ-P, may in fact be used in a CIOQ switch to reduce its matching complexity. Recognize the space-time duality of the spatial pipeline in CIOQ-P with a temporal pipeline in CIOQ. Without elaboration, we leave the reader with a tantalizing suggestion that these methods may be used in a CIOQ switch to reduce its arbitration complexity to $O(1+\epsilon)$, for an arbitrarily small $\epsilon$, by viewing the consecutive timeslots of a CIOQ switch as SE timeslots of a CIOQ-P switch, with a well-chosen number of space elements. We claim that these methods provide a better alternative to reduce matching complexity in input-queued switches as opposed to using envelope-based matchings [54], with its issues of unbounded latency.

Finally, we covered a G-MSM switch–a general memory-space-memory design obtained by simultaneously applying aggregation and pipelining to realize the benefits of both. The methods presented for a G-MSM were natural combinations of those for CIOQ-A and CIOQ-P. We showed (Corollaries 19, 20, 21, 22) that the shadow-and-decompose method, operating on virtual input-output queues, combined with sequential dispatch, striping, equal dispatch or concurrent dispatch may be used to achieve performance comparable to a CIOQ switch. The first offers strictest equivalence, but with a high matching frequency. The latter three afford a significantly lower frequency, resulting in lower complexity. We then showed that similar sequential, concurrent or balanced matchings may be performed on virtual element queues (Corollaries 23, 24, 25) to further lower the matching complexity, and provide 100% throughput to admissible traffic. Specifically, we analytically established that a concurrent SPS, or round-robin, on VEQ (Corollary 27) suffices for uniform traffic, and a 3DMM (Corollary 26) on VEQ for non-uniform traffic.

We opened up the intriguing possibility of recursion on BCS, by replacing the memory elements of a G-MSM by individual CIOQ switches (Sec. 6.3.3). The resulting 7-stage multi-path switch supports 100% throughput for admissible traffic (Proposition 5) using the combination method in Table 6.5. More beneficially, the required memory bandwidth becomes comparable to a CIOQ switch, thus addressing the single disadvantage of a G-MSM design. While we introduced this possibility, we did not cover such recursive designs in detail. Pending an evaluation of its practical utility, this may present an interesting topic for further research.

### 6.4.1 Related Work

The Atlanta/$\pi$ chipsets [19, 21, 22], of which the author was fortunate to be a contributor, was among the earliest switching products that employed a G-MSM design. Essentially, the matching algorithms used were similar to the EREW sub-maximal implementation (Fig. 5.11) of SPS matching, with sequential and concurrent dispatch (in different versions) among the central space elements.

No centralized arbiter is required for such matchings. Specifically, each input element may choose an output on a round-robin basis, and dispatch a cell if one exists for the unique and non-conflicting output indicated by its pointer. While the matching was on an element-pair basis, queues were kept separate for each input-output pair, for backpressure from the third to the first stages, essentially yielding a hierarchical output scheduling structure (pg. 177). Copious amounts of simulations indicated that the switch provided 100% throughput for uniform traffic, and also for some non-uniform patterns, with speedup. We have, in this work, established the theoretical basis for those observations. We now know that an EREW sub-maximal matching (Theorem 6 on SPS), combined with sequential (Theorem 12) or concurrent (Theorem 17) dispatch on VEQ suffices for uniform traffic, without speedup. Furthermore, the same suffices for slightly non-uniform traffic with speedup, specifically, a speedup of $k$ for average rates that are less than $k/N$ for each input-output pair.

**Concurrent Round-Robin Dispatch**

Oki et al. [85] proposed the concurrent round-robin dispatch (CRRD) algorithm for a G-MSM switch, and proved its sufficiency for 100% throughput to uniform traffic, using simulations. CRRD is an iterative algorithm, which essentially performs several steps of request-grant-accept (RGA, see pg. 59) operations, based on desynchronized round-robin pointers in the input elements. Much as the rationale for iSLIP [78], the intuition provided was that uniform traffic always keeps the pointers desynchronized leading to a maximal matching in each timeslot. We have analytically shown that less is needed. A single iteration of such schemes, evidenced by our results on EREW sub-maximal matchings, combined with sequential or concurrent dispatch, suffices for uniform traffic. The zeal to converge to a maximum-size matching ironically leads to worse performance, which then needs iteration as a remedy.

As a follow-up, the maximum weight matching dispatch (MWMD) was proposed by Rojas-Cessa et al. [93], who essentially showed that a MWM on VEQ, combined with sequential dispatch, is sufficient for 100% throughput to admissible traffic. Of course, this has been subsumed by our more general result on VEQ matching (Theorem 11) combined with sequential dispatch (Theorem 12), which established that *any* matching algorithm for input-queued switches, including critical, LPF, MWM, without speedup, and maximal, with speedup, that provides 100% throughput to admissible traffic, suffices for the above combination.

# Chapter 7

# Parallel Packet Switches

Our final focus is on multi-path buffered Clos switches, which are characterized by memory element(s) in the central stage, and require markedly different methods compared to the single-path designs of the previous two chapters. Specifically, we present and analyze a *parallel* packet switch (PPS) architecture, concentrating on load balancing and sequence control methods. While CIOQ switches and their derivatives are currently in the later stages of research, and common in deployment, much less is understood about the performance of a PPS. Though the initial steps towards that goal have been taken, including the contributions in this chapter, we are still away from deploying a truly high-performance PPS. There are some commercially available routers that employ clustering of several switching blades, and hence provide realizations of nominally multi-path switches, though the throughput and QoS performance of such products do not come close to our benchmark, the OQ switch.

We first introduce the PPS architecture, followed by a classification into flow-based and cell-based PPS. The former is a model for the aforementioned clustered routers, while the latter promises better throughput performance. For cell-based PPS, we propose and analyze *equal* and *fractional* dispatch methods for load balancing, in order to emulate an OQ switch. We then present a few sequence control schemes and analyze their suitability for such multi-path designs. Some of the results here, specifically, a few of the original contents in Sec. 7.3 and 7.4, were previously shared with the research community in a series of publications [17, 18, 64, 62, 63].

Our research[1] on this topic was interleaved by some pioneering contributions by Iyer and McKeown [49, 50], which are included as related work in Sec. 7.3. For completeness, we also review the load-balanced Birkhoff-Von Neumann (BVN) switch, which has emerged recently as an alternative multi-path BCS, in the discussion section.

## 7.1 Switch Architecture

A $N \times N$ PPS $(N, [xMx], N, K, s)$, previously shown in Fig. 4.6, is expanded in Fig. 7.1. Here $x$ may refer to a memory or a space element. We obtain such a switch by pooling the bandwidth resources of $K$ memory elements, and balancing the offered load among them. Consequently, there are $N$ elements of size $1 \times K$ in the first-stage, which perform load balancing and are called ingress *demultiplexors*. Similarly, there are $N$ elements of size $K \times 1$ in the third-stage, which reassemble the output traffic and are referred to as egress *multiplexors*. The $K$ central memory elements of dimensions $N \times N$ are called *core* elements. Normalized to an external capacity of one cell per timeslot, the internal links operate at $s/K$. The time required to transfer a cell on the latter is referred to as an *internal timeslot*, the size of which equals $K/s$ that of an external slot.

Notice that the architecture is structurally similar to a CIOQ-P switch. However, due to the presence of memory in the central stage, there is no simultaneous input and output contention as in CIOQ-P. Accordingly, the two dimensional scheduling, which we referred to as arbitration, necessary to account for the conflict-free property (2.5) in a $MSM$ design, may now be replaced by a uni-dimensional algorithm that merely matches an incoming cell to one of the $K$ possible core elements. Such matching may be performed by a centralized load balancer, or preferably by each demultiplexor in a concurrent fashion. Let the outcome of load balancing, in external slot $n$, be denoted by the $N \times K$ configuration $\pi(n)$, where $\pi_{i,k}(n) = 1$ implies that a cell arriving on input $i$ is dispatched to core element $k$. For unambiguous dispatch, we impose

$$\forall n,\ \forall i \leq N,\ \sum_{k=1}^{K} \pi_{i,k}(n) \leq 1 \tag{7.1}$$

[1] The architecture and performance studies detailed here were motivated by the $\pi$-group chip project, a prototype cell-based PPS co-architected and analyzed by the author in 1999–2002, as part of the commercial Atlanta/$\pi$ chipset effort at Bell Labs. Furthermore, part of the contents of Sec. 7.3 and 7.4 should be considered joint work with Denis Khotimsky, then at Bell Labs.

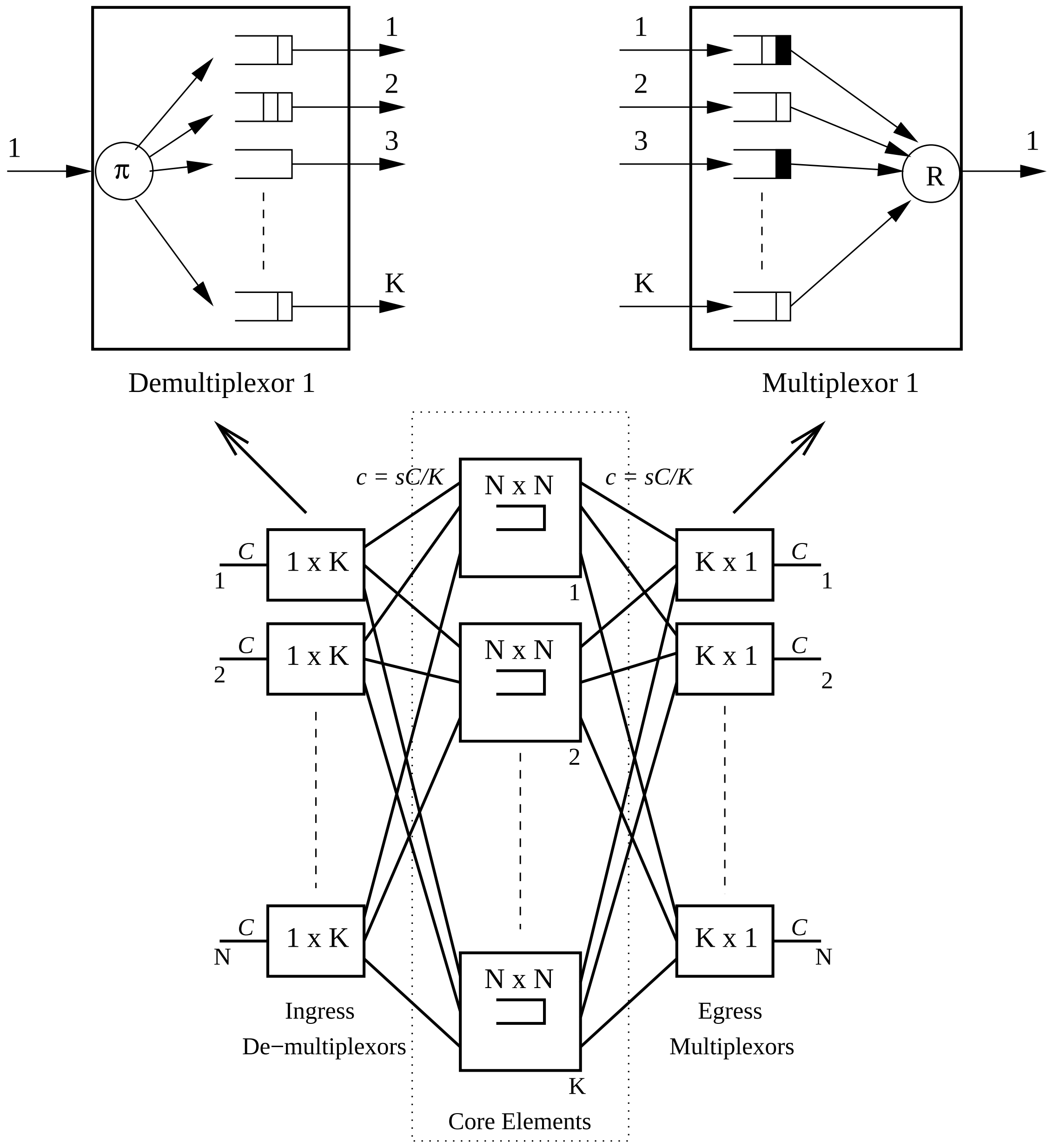


Figure 7.1: A parallel packet switch (PPS) with load balancers and re-assemblers

If the link from demultiplexor $i$ to core element $k$ is free, the cell may immediately be placed on the internal link. Otherwise, there is a contention for link $k$, and cells are enqueued into per-path *dispatch queues* in the demultiplexor. Notice that a transmission to link $k$ that begins at slot $n$ is completed by the end of slot $(n + \lceil \frac{K}{s} \rceil)$. Therefore, if an internal link is chosen at most once in those many consecutive slots, the demultiplexors do not need any queueing. In other words, we achieve memoryless dispatch as long as

$$\forall n,\ \forall i \leq N,\ \forall k \leq K, \quad \sum_{m=n}^{n+\lceil \frac{K}{s} \rceil} \pi_{i,k}(m) \leq 1 \tag{7.2}$$

We define an input-output *subflow* $(i, j, k)$ as the portion of traffic belonging to input-output pair $(i, j)$ that is dispatched to core element $k$. Let $A_{i,j}^{(k)}(n)$ denote the cumulative number of arrivals for subflow $(i, j, k)$, until external slot $n$. If $A_{i,j}(n)$ denotes the cumulative arrivals into the system for the pair $(i, j)$, the load balancing splits the incoming traffic as follows, which becomes an equality when the first stage is memoryless.

$$\forall i, j, k, n, \quad A_{i,j}^{(k)}(n) \leq \sum_{m=1}^{n} (A_{i,j}(m) - A_{i,j}(m-1))\pi_{i,k}(m) \tag{7.3}$$

In contrast to a CIOQ-P switch, however, cells belonging to the same input-output pair may experience significantly different delays in different paths, and hence may require resequencing in the multiplexor. More precisely, cells of subflow $(i, j, k)$ eventually contend for the output link $j$ of core element $k$. The contention experienced by subflow $(i, j, l)$ in core element $l$ may, in general, be different, resulting in several cells of flow $(i, j)$ being delivered to multiplexor $j$ out of sequence. If in-sequence delivery[2] is desired, cells may be enqueued into per-path *reassembly queues* in the multiplexor. A reassembler schedules cells out of those queues, restricting itself to those that are *eligible* for dispatch (shown shaded in Fig. 7.1). A cell is considered eligible if there are no other cells ahead of it in the arrival sequence that are still inside one of the core elements. Alternatively, the third stage may be memoryless, with the reassembler drawing cells from the heads of the respective core element queues, as and when they become eligible, and immediately dispatching them on the output link. Clearly, in either case, this depends upon the

[2]Bennett and Partridge, in [2], report that packet re-ordering in the Internet is typically not pathological network behavior, rather, it results due to incorrect network node functionality. As frequent losses play havoc on higher layer protocols such as TCP, maintaining in-sequence delivery should be considered a requirement for switching nodes.

ability to identify eligible cells as such. The reassembler is said to suffer from *redundant wait*, an undesirable scenario, if it is unable to correctly recognize eligible cells at all points in time.

So far, we have noted that the first stage may be comprised either of memory elements, with dispatch queues, or of space elements, if the load balancing satisfies (7.2). Similarly, the third stage may consist of memory elements, with reassembly queues, or of space elements, if the re-assemblers directly draw cells from the core element queues and immediately dispatch them. Note also that, if $s = 1$, and in-sequence delivery is not required, the third stage reduces to a trivial space element. In addition to these possibilities, we introduce a new type of element $\tilde{S}$, which stands for a memory element with a small fixed amount of memory. For all practical purposes, such an element may be considered a space element, as long as the memory can be implemented as on-chip (e.g., registers, static RAM) buffers. In this case, there is no need for external memory, and the on-chip buffers are no different from registers found within regular space elements, e.g., in the hardware pipeline. For our analysis, we will consider a memory size that depends only upon the switch dimensions, for arbitrary arrivals, as small enough so as to designate the element as $\tilde{S}$.

### 7.1.1 Benefits

The main benefits of the PPS architecture are modularity and component reuse. A $N \times N$ memory element with interface capacity of $s/K < 1$, used to build a lower-capacity switch, may now be reused to construct a $N \times N$ switch with an interface capacity of 1. Instead of building the higher capacity switch from scratch, one may add the latter to one's portfolio by merely building the requisite demultiplexors and multiplexors. The memory bandwidth of a core element is $(1 + N)s/K$ at each output, while that of the first and third stage elements are upper bounded by $(1 + s)$, if those elements contain external memory. This is the motivation behind the design of most of the commercially available clustered and stackable routers, in which several blades, often offered separately as stand-alone lower-capacity switches, are interconnected by a cluster controller that encompasses the demultiplexor and multiplexor functionality. Clearly, the bandwidth of the fastest memory in the system becomes $(1 + s)$ when

$$K \geq \frac{s(1 + N)}{1 + s}$$

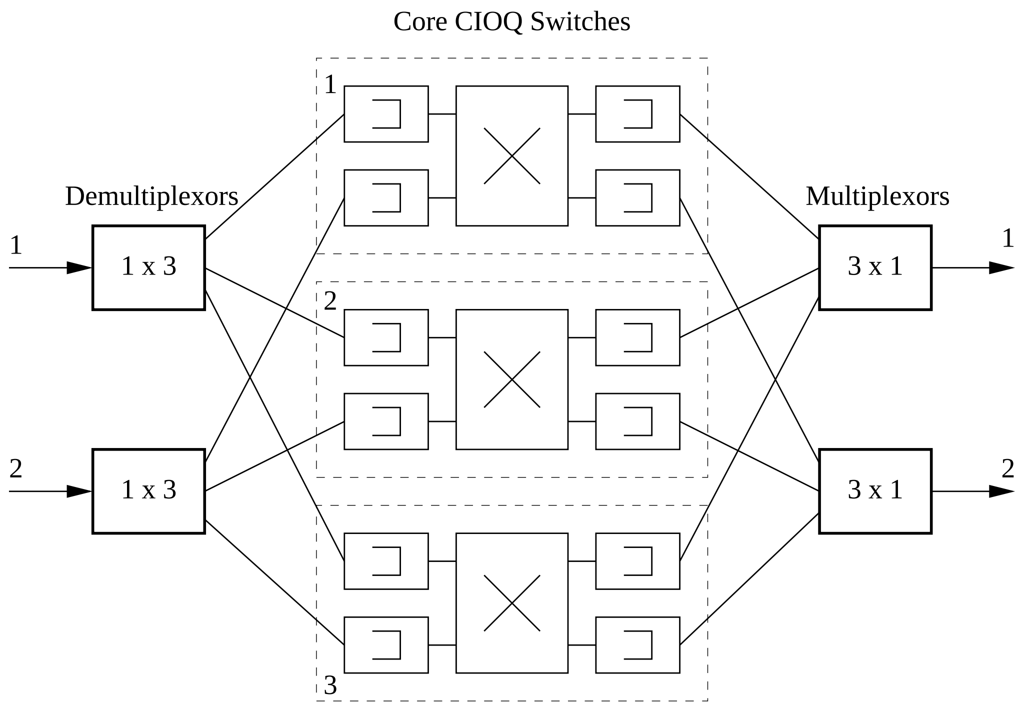


Figure 7.2: A $2 \times 2$ PPS with CIOQ switches for central memory elements

The benefit of reuse may be extended by replacing a core element with another switch. Provided it does not affect the throughput and QoS performance of a PPS, a core memory element may be replaced by a CIOQ switch, as shown in Fig. 7.2. In this case, the memory bandwidth required in the core drops to $(1 + s')s/K$, where $s'$ is the internal speedup of the chosen CIOQ switch. The fastest memory in the system, residing in the first and last stages, continues to run at $(1 + s)$, but without any restriction on the dimensions. In fact, all the $K$ central elements (dotted in the figure) may be realized by a single $KN \times KN$ CIOQ switch. We introduced such an architecture in [17], motivated by the goal to build a $8 \times 8$ OC-192 switch using a $32 \times 32$ OC-48 switch from the Atlanta/$\pi$ [22] family. Due to the similarity with classical inverse multiplexing of low-rate links, also called link bonding, we dubbed this an *inverse multiplexed* switch.

Notice that if component reuse is not a concern, we might prefer a plain CIOQ architecture over PPS, as it affords a comparable memory bandwidth without being hampered by issues such as cell sequencing. However, if the first *and* third stage elements of a PPS can be realized as space elements $S$ or $\tilde{S}$, the fastest memory in the system would run at $(1+s')s/K$, or $a/K$ for some constant $a$. As $K$ is a design parameter under our control, this would allow us to build switches in which the fastest external memory runs slower than the external interfaces. As memory bandwidths are not keeping pace with increases in link transmission rates, this represents the biggest promise of a PPS architecture. In fact, by appropriately choosing $K$, we might finally be able to remove the interdependence of interface rates with memory bandwidths. If the resulting PPS emulates an OQ switch, we may also recursively replace the core elements with PPS switches, allowing the usage of arbitrarily slow memories. While we consider this academic possibility, we do not explore a recursive PPS in this work.

To make the above (non-recursive) high-capacity low-memory bandwidth switch a reality, the methods in the switch need to provide optimal throughput and QoS, while (i) maintaining cell sequence; (ii) preferably allowing for a memoryless first and third stages, or at least memories that run slower than the external interfaces, and (iii) enabling the employment of a CIOQ switch in the core without degrading performance.

### 7.1.2 Queueing Strategies

#### Demultiplexor

A small-memory demultiplexor may use per-path FIFO for dispatch queues, irrespective of how the incoming cells are distributed, as shown in Fig. 7.1. A cell arriving at timeslot $n$, for input-output flow $(i,j)$, is enqueued into FIFO $(i,k)$, of length $X_{i,k}(n)$, where $k$ is such that $\pi_{i,k}(n) = 1$. Therefore, there are exactly $K$ FIFO queues in each demultiplexor. Let $X_{i,k}^{(j)}$ refer to the number of cells in FIFO $(i,k)$ that belong to flow $(i,j)$. Then, we may extend (7.3) as follows, where $D_{X_{i,k}}(n)$ refers to the cumulative departures from queue $(i,k)$ until timeslot $n$.

$$\forall i,j,k,n,\quad A_{i,j}^{(k)}(n) = \sum_{m=1}^{n} (A_{i,j}(m) - A_{i,j}(m-1))\pi_{i,k}(m) - X_{i,k}^{(j)}(n) \tag{7.4}$$

$$X_{i,k}(n) = \sum_{j=1}^{N} X_{i,k}^{(j)}(n) \tag{7.5}$$

$$X_{i,k}(n) = \sum_{j=1}^{N}\sum_{m=1}^{n} (A_{i,j}(m) - A_{i,j}(m-1))\pi_{i,k}(m) - D_{X_{i,k}}(n) \tag{7.6}$$

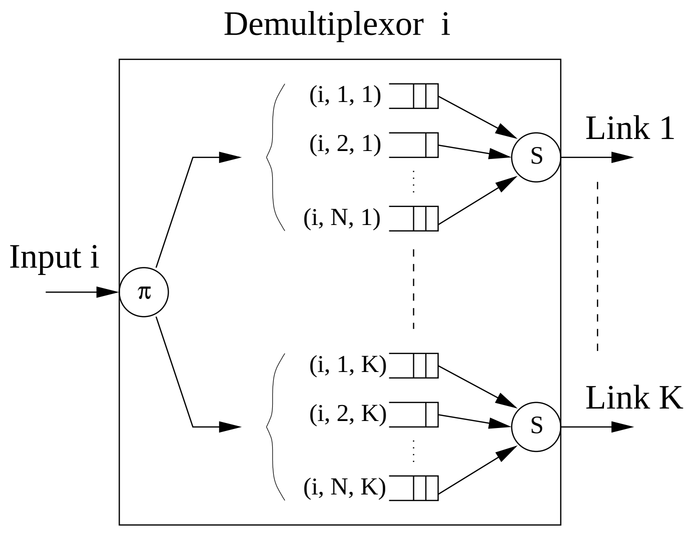


Figure 7.3: Ingress demultiplexor with per-path virtual output queueing

$$D_{X_{i,k}}(n + \left\lceil l\frac{K}{s} \right\rceil) \quad = \quad D_{X_{i,k}}(n) + l, \quad \forall l \text{ s.t. } X_{i,k}(n) \geq l \tag{7.7}$$

The last line assumes that the demultiplexor is work conserving on each internal link, i.e., as long as $X_{i,k} > 0$, link $k$ remains busy and transfers cells at a rate of $s/K$ per external slot. Note that cells belonging to different flows may be multiplexed into the same queue. Due to their small sizes, the stability of $X$ is moot, however, a cell destined to link $k$ may experience a worst-case delay of $|X_{i,k}|K$.

Clearly, if the demultiplexors are composed of memory elements, cells may experience unbounded delay with per-path FIFO queueing. While this may not affect the throughput of input-output pairs as a whole, it prevents any provisioning of bandwidth and delay guarantees. Hence, such a demultiplexor requires hierarchical queueing at each link, as shown in Fig. 7.3. We may either employ per-path VOQ or per-flow queueing for each link, depending upon the desired granularity of traffic isolation, with a work-conserving scheduler at each internal link. For per-path VOQ, a cell arriving at timeslot $n$, for input-output flow $(i, j)$, is enqueued into VOQ $(i, j, k)$, of length $X_{i,k}^{(j)}(n)$, where $k$ is such that $\pi_{i,k}(n) = 1$. Hence, there are $NK$ virtual output queues in each demultiplexor, one for each subflow. The evolution of the per-path VOQ (and similarly, of per-path flow queues, if employed) may be expressed as the following modification of the above

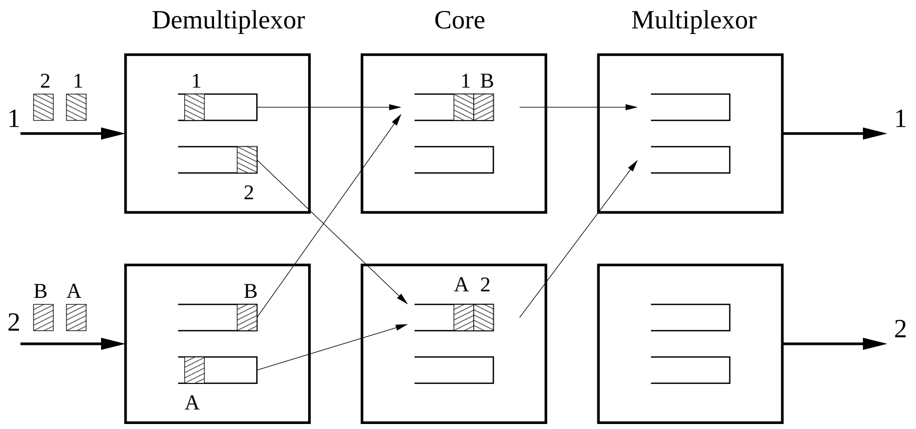


Figure 7.4: PPS: FIFO in the core may be incompatible with a buffered first stage

equations for per-path FIFO queueing.

$$\forall i,j,k,n,\ A_{i,j}^{(k)}(n) = \sum_{m=1}^{n}(A_{i,j}(m)-A_{i,j}(m-1))\pi_{i,k}(m) - X_{i,k}^{(j)}(n) \tag{7.8}$$

$$X_{i,k}^{(j)}(n) = \sum_{m=1}^{n}(A_{i,j}(m)-A_{i,j}(m-1))\pi_{i,k}(m) - D_{X_{i,k}^{(j)}}(n) \tag{7.9}$$

$$\sum_{j=1}^{N} D_{X_{i,k}^{(j)}}(n+\left\lceil l\frac{K}{s}\right\rceil) = \sum_{j=1}^{N} D_{X_{i,k}^{(j)}}(n)+l,\quad \forall l \text{ s.t. } \sum_{j=1}^{N} X_{i,k}^{(j)}(n) \geq l \tag{7.10}$$

Since there is no contention at an internal link, as long as the average arrival rate into the VOQ system at that link is less than $s/K$, all those queues remain stable. The link scheduler controls the bandwidth trunk and latency observed by each subflow.

### Core Elements

Core element $k$ receives cells for each subflow $(i,j,k), i \leq N, j \leq N$, with respective cumulative arrivals $A_{i,j}^{(k)}(n)$, where $n$ is the external timeslot. These may be enqueued into FIFO queues, one per output $j$, with length $Q_j^{(k)}(n)$. Let $D_j^{(k)}(n)$ be the cumulative departures from output $j$ until timeslot $n$, and $B_j^{(k)}(n)$ denote an *indicator* of whether the output is allowed to send a cell to multiplexor $j$. Then, the evolution of the $N$ FIFO queues in each core element may be given by

$$\forall j,k,\ Q_j^{(k)}(n) = \sum_{i=1}^{N} A_{i,j}^{(k)}(n) - D_j^{(k)}(n) \tag{7.11}$$

$$D_j^{(k)}(n + \left\lceil \frac{K}{s} \right\rceil) = D_j^{(k)}(n) + 1 \quad \text{if} \quad Q_j^{(k)}(n) B_j^{(k)}(n) > 0 \tag{7.12}$$

It is evident that if the average arrival rate into a FIFO is less than $s/K$, and if the core element is work conserving, that queue remains stable. Memoryless and small-memory multiplexors may explicitly indicate a preference for specific flows by setting $B_j^{(k)} = 1$. Such a *backpressure* technique might prevent work conserving behavior, in which case queue stability will need to be separately established.

An important observation is that FIFO queueing may be incompatible with a buffered demultiplexor, as shown in an example in Fig. 7.4. The figure shows two flows $(1, 1)$ and $(2, 1)$, both of which are split across multiple core elements. In general, depending upon the occupancies of the per-path queues in the demultiplexor, an older cell of one flow might find itself ahead of a more recent cell of another flow, in the same core FIFO queue. Since this may happen in multiple core elements simultaneously, we may arrive at a reassembly *deadlock* and mutual head-of-line blocking. Notice that this situation will not arise if the demultiplexors are memoryless. Furthermore, if they are implemented as $\tilde{S}$, with per-path FIFO size $B$, HOL blocking can be prevented by ensuring that each cell is delayed by exactly $BK/s$ slots, with respect to its arrival into the system, irrespective of how long it spends in the demultiplexor. Such a *delay equalization* technique has been used in other architectures with buffers in multiple stages, e.g., in a load-balanced BVN switch [10].

Alternatively, a cell belonging to subflow $(i, j, k)$ may be enqueued into a virtual input queue (VIQ) $(i, j)$, of length $Q_{i,j}^{(k)}(n)$, at the output $j$ of core element $k$. Let $D_{i,j}^{(k)}(n)$ and $B_{i,j}^{(k)}(n)$ denote the cumulative departures for VIQ $(i, j)$ and an input-output indicator, respectively. Then, the evolution of the $N^2$ VIQ system in each core element is given by

$$\forall i, j, k, \;\; Q_{i,j}^{(k)}(n) = A_{i,j}^{(k)}(n) - D_{i,j}^{(k)}(n) \tag{7.13}$$

$$\forall j, k, \;\; \sum_{i=1}^{N} D_{i,j}^{(k)}(n + \left\lceil \frac{K}{s} \right\rceil) = \sum_{i=1}^{N} D_{i,j}^{(k)}(n) + 1 \quad \text{if} \quad \sum_{i=1}^{N} Q_{i,j}^{(k)}(n) B_{i,j}^{(k)}(n) > 0 \tag{7.14}$$

The last line expresses the fact that if some flow $(i, j)$ has a backlogged cell, with $B_{i,j} = 1$, a cell (not necessarily for $(i, j)$) is dispatched to multiplexor $j$. A link scheduler, at each output $j$ of the core element, determines which subflow is served in each internal timeslot. Note that, for performance guarantees to finer grain flows, the VIQ system may be exchanged with per-flow

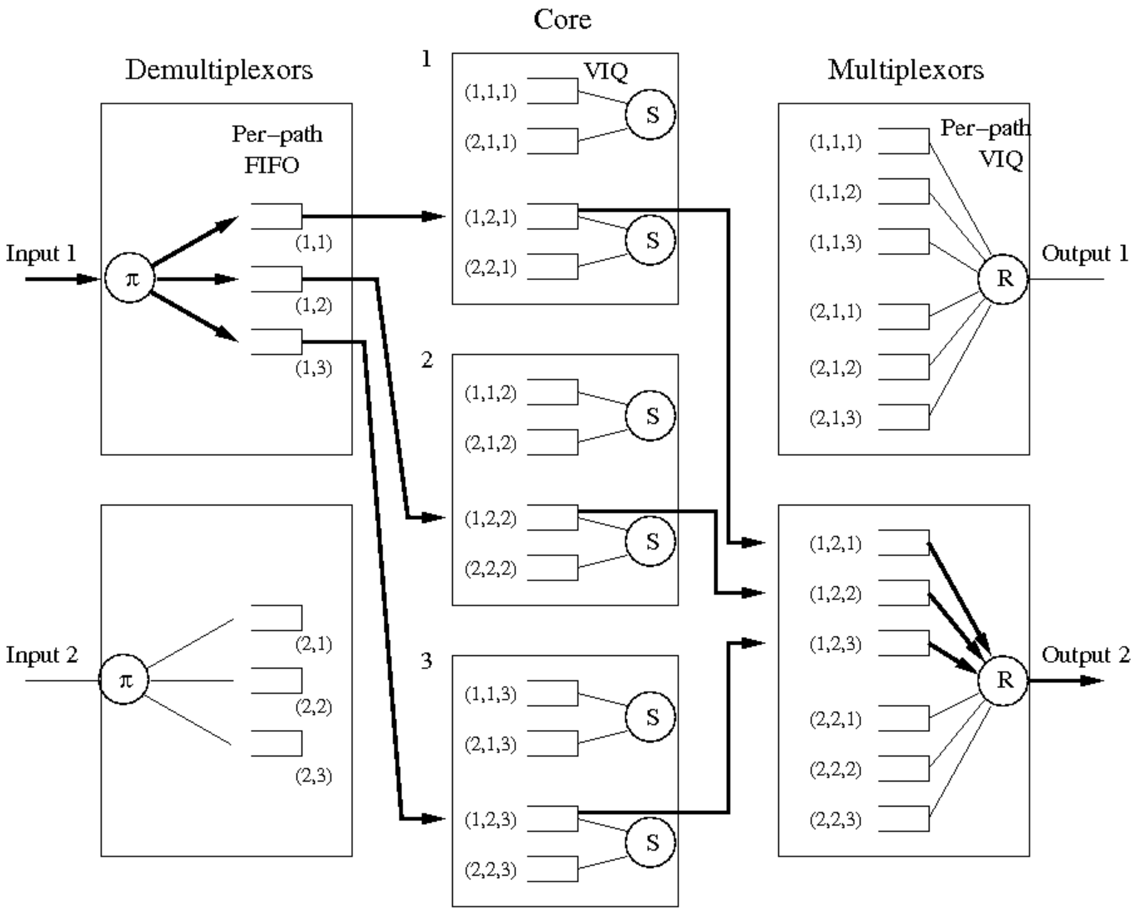


Figure 7.5: A PPS with virtual input queues in the core elements and multiplexors

queues at each output. Also, if the core elements are implemented as CIOQ switches, the queues covered in this section refer to their output-element queues.

**Multiplexor**

The queueing structure in buffered multiplexors has a one-to-one relationship with the core element structure. Specifically, if the core employs per-output FIFO queues, a multiplexor $j$ contains $K$ FIFO queues $(k, j)$, as shown in Fig. 7.1. Alternatively, it may employ per-path VIQ $(i, j, k)$ (or per-path flow queues for finer granularity) as shown in Fig. 7.5. The dark arrows in the figure illustrate the multiple paths, and the respective queues encountered on those paths, for flow $(1, 2)$.

Recognize that at all points in time, the first cell in sequence for flow $(i, j)$, still in the system, which we henceforth call the *head cell*, resides at the head of the multiplexor VIQ $(i, j, k)$ for some $k$, or at the head of VIQ $(i, j, k)$ in some core element $k$, or in the demultiplexor. If the head cell of any flow $(i, j)$ is already in multiplexor $j$, by definition, it contains cells eligible for dispatch. Let $E_{i,j}(n)$ be the indicator of the existence of eligible cells for flow $(i, j)$. Further, let $Z_{i,j}^{(k)}(n)$ be the length of VIQ $(i, j, k)$ at external slot $n$, and $D_{Z_{i,j}^{(k)}}(n)$ the cumulative departures. Then, we have

$$\forall i, j, k, n, \;\; Z_{i,j}^{(k)}(n) = D_{i,j}^{(k)}(n) - D_{Z_{i,j}^{(k)}}(n) \tag{7.15}$$

$$\forall j, n, \;\; \sum_{k=1}^{K}\sum_{i=1}^{N} D_{Z_{i,j}^{(k)}}(n+1) = \sum_{k=1}^{K}\sum_{i=1}^{N} D_{Z_{i,j}^{(k)}}(n) + 1, \quad \text{if } \sum_{i=1}^{N} E_{i,j}(n) > 0 \tag{7.16}$$

The first equation is true because the VIQ systems in the core and in the multiplexors work in tandem, and the cumulative arrivals into the third stage equals the cumulative departures from the second. The second equation merely states that if there are eligible cells in multiplexor $j$, one will be dispatched in each slot. Notice that this assumes the ability of the multiplexor to avoid redundant wait. The specific flow $(i, j)$, with $E_{i,j}(n) = 1$, which gets served at timeslot $n$, is determined by the output link scheduler. In other words, the scheduler at link $j$ first selects a flow $(i, j)$, and then dequeues the head cell. We emphasize that if the core elements employ VIQ (or per-flow queues), the multiplexor is *required* to at least maintain the same granularity of queueing. It is evident that if the multiplexor instead employs per-path FIFO queues, in-sequence delivery of cells is ruled out.

A small-memory multiplexor may employ the same queueing structure as above, with backpressure to the core elements. More precisely, if the size of VIQ $(i, j, k)$ is equal to $B$, a signal is sent to VIQ $(i, j, k)$ in core element $k$ to prevent further arrivals to that subflow, whenever $Z_{i,j}^{(k)} = B$. In other words, the tandem queueing systems are related to each other by

$$\forall i, j, k, n, \;\; B_{i,j}^{(k)}(n) = 1_{\{Z_{i,j}^{(k)}(n) < B\}}, \tag{7.17}$$

which might affect the stability of the core element queues. A memoryless multiplexor, on the other hand, may explicitly schedule cells directly from the core element, assuming it has the necessary information to do so, by setting the signal for the desired subflows. The received cells

**Algorithm 26.** *PPS: Static Dispatch*

```
Initialize:          1. For each flow $f$, assign $p(f)$

                     For each demultiplexor $i$ in parallel
Every Timeslot $n$:  If a cell arrives for $(i, j)$
                     1. $f \leftarrow$ Identify flow
                     2. $k \leftarrow p(f)$
                     3. $\pi_{i,k}(n) = 1$ (Dispatch to path $k$)
                     4. Enqueue into VOQ $(i, j, k)$
```

Figure 7.6: Flow-based PPS: Static dispatch method

are serialized and immediately placed on the output link. To indicate an unambiguous preference, and to ensure the memoryless property, the signals are constrained as follows:

$$\forall j, k, \quad \sum_{i=1}^{N} B_{i,j}^{(k)} = 1 \tag{7.18}$$

$$\forall j, \quad \sum_{k=1}^{K} \sum_{i=1}^{N} B_{i,j}^{(k)} \leq \frac{K}{s} \tag{7.19}$$

## 7.2 Flow-based PPS

A PPS $(N, [xMx], N, K, s)$ may statically assign a flow, from input $i$ to output $j$, to one of the possible $K$ paths between $(i, j)$. Once a path is assigned, each cell of the flow follows its predetermined route through the switch. We refer to such a switch as a flow-based PPS, and the method as *static dispatch*, illustrated in Fig. 7.6. Path determination, on each cell arrival for flow $f$, may be carried out using a persistent hash function or a pre-populated lookup table $p(f)$. Flows may correspond to the traffic belonging to input-output pairs, or finer-grain streams within those pairs, identified using fields in the cell header. Note that the figure does not show the dequeue and enqueue operations in the rest of the system, which proceeds as described in the previous section.

Let the average arrival-rate $\lambda_{i,j}$ (2.1), for each input-output pair $(i, j)$, normalized to the external link capacity, be known and admissible (2.2). Define $\lambda_{\max} = \max_{i,j}\{\lambda_{i,j}\}$ and $\lambda_{\min} =$

GCD$\{\lambda_{i,j}\}$. To ensure that each flow may fit in a single path, impose $\lambda_{\max} \leq \frac{s}{K}$, limited by the capacity of the internal links. A greedy Clos fitting algorithm (pg. 32) may be used to assign paths to each of the $N^2$ flows. This merely visits the central elements, one by one, for each flow $(i,j)$ in a sequential fashion, until a core element $k$ is found with sufficient bandwidth on both its input $i$ and output $j$. The well-established theory of multi-rate Clos networks (e.g., see [81]), from circuit switching, may directly be applied to yield the following sufficiency condition.

**Lemma 8.** *A flow-based PPS is guaranteed to find a path for an admissible flow, without rearranging existing path assignments, provided* $\lambda_{\max} \leq \frac{s}{K}$ *and*

$$s \geq (1 - \lambda_{\max})\frac{K}{\left\lceil \frac{K-1}{2} \right\rceil} + (\lambda_{\max} - \lambda_{\min})K \tag{7.20}$$

*Proof.* Consider the assignment of an admissible flow between $(i,j)$, with rate $\lambda_{\max}$. The total bandwidth already allocated at input $i$ (output $j$) cannot exceed $(1 - \lambda_{\max})$. Let $I$ be the number of links between demultiplexor $i$ and the core elements that *cannot* accommodate $\lambda_{\max}$. It follows that those links have an existing bandwidth assignment of at least $(\frac{s}{K} - \lambda_{\max} + \lambda_{\min})$. Therefore, $I$ is an integer such that

$$I(\frac{s}{K} - \lambda_{\max} + \lambda_{\min}) \quad \leq \quad (1 - \lambda_{\max}) \tag{7.21}$$

$$\text{or} \quad I \quad \leq \quad \left\lfloor \frac{1 - \lambda_{\max}}{\frac{s}{K} - \lambda_{\max} + \lambda_{\min}} \right\rfloor \tag{7.22}$$

The number of links $J$ between multiplexor $j$ and the core elements that cannot accommodate $\lambda_{\max}$ is also given by (7.22). Therefore, we are guaranteed to find a core element that can accommodate $\lambda_{\max}$, on both its input and output, provided

$$\begin{aligned} K &\geq I + J + 1 \\ &\geq 2\left\lfloor \frac{1 - \lambda_{\max}}{\frac{s}{K} - \lambda_{\max} + \lambda_{\min}} \right\rfloor + 1 \end{aligned} \tag{7.23}$$

Since $a \geq \lfloor b \rfloor$ implies $\lceil a \rceil \geq b$, we may rewrite (7.23) as

$$\frac{1 - \lambda_{\max}}{\frac{s}{K} - \lambda_{\max} + \lambda_{\min}} \leq \left\lceil \frac{K-1}{2} \right\rceil,$$

rearranging which, we obtain $s \geq (1 - \lambda_{\max})\frac{K}{\lceil \frac{K-1}{2} \rceil} + (\lambda_{\max} - \lambda_{\min})K$ □

The above proof, in reality, is a simple application of fitting additive scalars, in the interval $[\lambda_{\min}, \lambda_{\max}]$, in a circuit Clos network (3.1). Notice that result (7.20) continues to hold for fitting finer grain flows $f$, within input-output pairs $(i, j)$, with known average rates $\lambda_f$, provided they are admissible, i.e., if

$$\forall i \sum_j \sum_{f \in (i,j)} \lambda_f \leq 1 \quad \text{and} \quad \forall j \sum_i \sum_{f \in (i,j)} \lambda_f \leq 1 \tag{7.24}$$

Clos fitting ensures that the total (average) bandwidth assigned at the inputs and outputs of each core element does not exceed $s/K$, the internal link capacity. Therefore, irrespective of the queueing strategy, a work-conserving scheduler at each output of the demultiplexors, core elements and the multiplexors, is sufficient to guarantee the stability of the attached queues. Furthermore, since a flow is assigned to a unique path, there is no need for sequence control. Note, however, that the first stage needs to be comprised of memory elements because there may be an unbounded number of back-to-back cells arriving for the same flow, at a peak rate of 1, destined to the same internal link, with capacity $s/K$. Similarly, the third stage also needs memory to ensure work-conserving service at each core-element output. Therefore, from Lemma 8, we obtain the following result for flow-based PPS.

**Theorem 16.** *For a PPS with memory elements in the first and third stages, Clos fitting of admissible flows, with known average rates, is sufficient to ensure 100% throughput with speedup s, provided* $\lambda_{\max} \leq \frac{s}{K}$ *and*

$$s \geq (1 - \lambda_{\max}) \frac{K}{\left\lceil \frac{K-1}{2} \right\rceil} + (\lambda_{\max} - \lambda_{\min}) K$$

*That is, for the above speedup,*

$$(\text{PPS}, \{\text{Clos Fitting}\}) \overset{T, f_1}{\simeq} (\text{OQ}, \{\text{WC}\}), T\text{: SLLN}$$

Depending upon the relative values of $\lambda_{\max}$ and $\lambda_{\min}$, the required speedup may be prohibitive, due mainly to the fragmentation of bandwidth in the internal links. For instance, for $\lambda_{\max} = s/K$ and $\lambda_{\min} \to 0$, the minimum value of $s \to K$. In this situation, parallelization offers no benefits! On the other hand, if all the rates are equal, e.g., if $\lambda_{\max} = \lambda_{\min} = \frac{s}{K}$, we

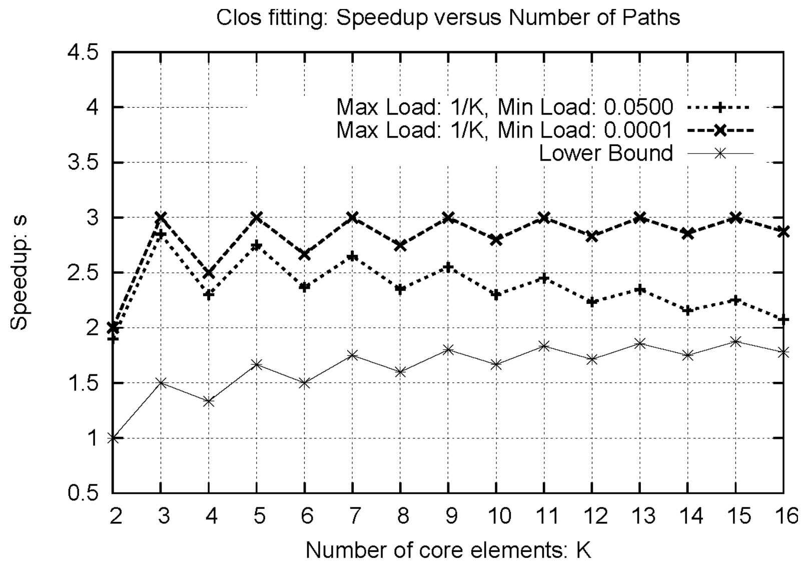


Figure 7.7: Clos fitting in PPS: Effect of unbalanced load on speedup

obtain[3]:

$$\begin{aligned} s &\geq \frac{(1-\frac{s}{K})K}{\left\lceil \frac{K-1}{2} \right\rceil} \\ \text{or } s &\geq \frac{K}{1+\left\lceil \frac{K-1}{2} \right\rceil} \end{aligned} \qquad (7.25)$$

More meaningfully, in a switch with a large number of fine-grain equal-bandwidth flows, as $\lambda_{\max} \to 0$ (and hence $\lambda_{\min} \to 0$), the required speedup is given by

$$s \geq \frac{K}{\left\lceil \frac{K-1}{2} \right\rceil}, \qquad (7.26)$$

which is equal to 2 for even $K$, and slightly higher for odd values of $K$. Fig. 7.7 shows an example of how the relative values of offered load affects the required speedup. As long as the minimum offered load is not insignificant, the speedup approaches the lower bound (7.25) for large $K$.

This may be directly applied to clustered routers, which assign paths for each IP flow, based upon a hash function of the source and destination fields in the packet header. The main

[3]Note that this example makes sense only when $F\frac{s}{K} \leq 1$, where $F$ is the maximum number of flows at an input or output

assumption is that if there are a large number of flows, each offering a load that equals a miniscule portion of the link capacity, a uniform hash function suffices without knowledge of the average rates. The performance of such functions have been studied in the past, in the context of multi-link load-balancing (e.g., [53, 6]). The relevant observation in such works is that such hash functions lead to instability if the load offered by the flows vary widely with respect to each other. For switch load-balancing, in our case, this manifests as bandwidth fragmentation, necessitating a higher speedup. Moreover, the hash function needs to ensure that no internal link is overloaded, a task that, in general, is difficult without knowledge of the average rates. We might infer such knowledge using measurements of instantaneous load on each internal link, but such techniques do not allow us to establish long-term stability.

Due to the above limitations, a flow-based PPS (or a clustered router) fails to provide 100% throughput, even for admissible traffic and a high speedup, if the offered load of each flow is not known. Nevertheless, Theorem 16 may be applied to provide bandwidth and delay guarantees in a flow-based PPS. For the guaranteed component of the traffic, rate reservation $R_f$ for each flow $f$ is known during admission control, which may be used to assign a path, using Clos fitting and a speedup specified by (7.20). Per-flow queueing is employed in each stage, with the link schedulers on the assigned path programmed to provide a service rate of at least $R_f$ to flow $f$. Then, if $\lambda_f \leq R_f$, the corresponding per-flow queues remain stable. Furthermore, if the flow is leaky-bucket constrained, with bucket size $\sigma_f$, the total delay experienced by a cell belonging to $f$ is upper bounded by

$$D_f \leq \frac{\sigma_f}{R_f} + \alpha_{i,k} + \beta_{j,k} + \gamma_j, \tag{7.27}$$

where $f \in (i, j)$, $k = p(f)$, $\alpha_{i,k}$ is the latency of link scheduler $k$ in demultiplexor $i$, $\beta_{j,k}$ of the scheduler at output $j$ of core element $k$, and $\gamma_j$ of the output link scheduler at multiplexor $j$. This is because the three stages may be viewed as an end-to-end path composed of latency-rate servers [104]. Recognize that a QoS-capable CIOQ switch may easily replace each core element, without violating the ability of the flow-based PPS to provide bandwidth guarantees. Since an additional contention point is encountered, a value of $2/\lambda_{\min}$ gets added (an easy extension of (5.8)) to the above delay bound if each central CIOQ employs batch-mode maximal matching or BVN decomposition.

**Flow Splitting**

The high speedup required to address bandwidth fragmentation may be alleviated by *splitting* a selected set of flows across multiple paths. Specifically, let $\lambda_f$ be the average rate associated with flow $f$ between some input-output pair $(i, j)$. Flow splitting results in a portion $\lambda_f^{(k)} \geq 0$ assigned to path $k$, i.e., between input $i$ and output $j$ of core element $k$, such that $\sum_k \lambda_f^{(k)} = \lambda_f$. The assignments are constrained by the available internal-link bandwidth at input $i$ and output $j$, namely,

$$\forall i \sum_j \sum_{f \in (i,j)} \lambda_f^{(k)} \leq \frac{s}{K} \quad \text{and} \quad \forall j \sum_i \sum_{f \in (i,j)} \lambda_f^{(k)} \leq \frac{s}{K} \tag{7.28}$$

To adhere to such an assignment, path determination, for each arriving cell, becomes more involved than the step shown in Fig. 7.6. For a split flow, a path is selected such that each core element $k$ receives a fraction $\lambda_f^{(k)}/\lambda_f$ of the arriving cells. In addition, such flows require resequencing of cells in the multiplexor.

Splitting may instead be performed using rate reservation $R_f$, as opposed to a known $\lambda_f$, in order to provide bandwidth guarantees to flow $f$. In that case, per-flow queueing is employed in each stage, with the link schedulers, on each assigned path $k$, programmed to provide a service rate of $R_f^{(k)}$ to flow $f$. We extend the delay bound in (7.27), without elaboration, to

$$D_f \leq \max_{k \text{ s.t. } R_f^{(k)} > 0} \left( \frac{\sigma_f}{R_f^{(k)}} + \alpha_{i,k} + \beta_{j,k} \right) + \gamma_j \tag{7.29}$$

Observe that a rather simple strategy of splitting each flow equally among all the paths, i.e., $\forall k, f$, $\lambda_f^{(k)} = \lambda_f/K$, satisfies the internal link constraint (7.28), for $s \geq 1$. In fact, for queue stability under admissible arrivals, even the knowledge of $\lambda^{(f)}$ is not required[4]. Cells may be dispatched in a round-robin fashion across all the $K$ paths, for each flow. Furthermore, such equal splitting may be carried out on coarse-grain input-output flows, or $N$ flows in each demultiplexor and multiplexor, with work-conserving schedulers in each stage, in order to maintain queue stability for each input-output pair as a whole, irrespective of the individual values of $\lambda_f$. On the other hand, for QoS guarantees, equal splitting requires the rate reservation of *each* flow to be divided equally across all paths. While path determination, for the incoming cells, may be

[4]This fact will be used in the next section to establish stability results for a cell-based PPS.

performed using round-robin per flow, without the knowledge of the rates, each scheduler on its path needs to be programmed with a service rate of $R_f/K$.

We may contemplate other strategies for flow fitting only if the number of split flows can be reduced. Specifically, for stability, the goal is to split the offered load for less than $N$ input-output pairs in each first (and third) stage element. For QoS guarantees, on the other hand, the number of split flows in each element may be any value less than $F$, where $F$ is the maximum number of flows on each external link. Minimizing the number of split flows is a variation of the classical bin-packing problem in operations research. While an exhaustive treatment of this minimization problem is outside the scope of this work, we introduce the first-fit Clos fitting algorithm, which may be used to reduce the speedup, with respect to (7.20), and can be applied to reduce the number of split flows. For each flow, this algorithm visits the core elements in sequence and first attempts to fit the flow without splitting. If the flow cannot be accommodated as such, it is split in some arbitrary fashion. While Theorem 16 was an application of multi-rate Clos networks, splitting brings it back into the realm of unit-capacity Clos networks.

**Theorem 17.** *For a PPS with memory elements in the first and third stages, and flow splitting, assuming the flow rates are known and admissible, the first-fit Clos fitting algorithm is sufficient to ensure 100% throughput, with speedup $s \geq 2 - \lambda_{\min}$. That is,*

$$\text{(PPS, \{First-fit Clos Fitting\})} \overset{T,f_1}{\simeq} \text{(OQ, \{WC\})}, T\text{: SLLN}, s \geq 2 - \lambda_{\min}$$

*Proof.* Let $C = 1/\lambda_{\min}$. Consider a circuit Clos network with $N$ first (third) stage elements with $C$ inputs (outputs) each, and $sC$ second-stage elements, with $N$ inputs and outputs each. Let the second-stage elements be partitioned into $K$ sets of size $sC/K$. For each flow $f$, $f \in (i, j)$ with rate $\lambda_f$, generate a simultaneous request for $C_f = \lambda_f/\lambda_{\min}$ circuits between elements $i$ and $j$ of the circuit Clos network. If $f$ is admissible in the PPS, all of the $C_f$ circuits are admissible in the latter. From Clos theorem, a path may be found for each circuit provided

$$\begin{aligned} sC &\geq 2C - 1 \\ \text{or} \quad s &\geq 2 - \lambda_{\min} \end{aligned}$$

For the $C_f$ requests, let $C_f^{(k)}$ denote the number of paths realized through second-stage elements in set $k$, $k \leq K$. Then, an assignment of $C_f^{(k)}\lambda_{\min}$ on core element $k$ ensures that (7.28) is never violated. Since $C_f = \sum_k C_f^{(k)}$, the flow is fully assigned. □

The above is a simple application of the space-time duality of Clos networks. Notice that a first-fit assignment can be made such that, when a flow between $(i, j)$ is split, at least one internal link $k$ (one set $k$ in the circuit equivalent) is fully saturated at either input $i$ or output $j$. As a result, no more than $2K$ flows need to be split at each demultiplexor and multiplexor. In a dynamic system, however, with a changing set of flows, this requires the existing set of split flows to be re-assigned whenever there is a new flow added to the system. Thus, for stability under known admissible rates, we may prefer first-fit Clos fitting over equal splitting if $N > 2K$. Similarly, Theorem 17 may be applied towards bandwidth guarantees, given rate-reservations for each flow, if $F > 2K$.

**Corollary 28.** *For admissible flows, a PPS with first-fit Clos fitting can assign paths by splitting no more than $2K$ flows, with a reassignment of no more than $2K$ existing flows on each flow addition, with $s \geq 2 - \lambda_{\min}$.*

To summarize, a flow-based PPS may statically assign a unique path to each flow, with known offered load, so as to ensure stability, or with known rate-reservation, to provide QoS guarantees, provided $\lambda_{\max} \leq \frac{s}{K}$ and the speedup satisfies (7.20). While this value might be high, the advantage is that there is no need for sequence control. To reduce the speedup to a value close to 2, we may choose to split upto $2K$ flows, and rearrange upto $2K$ on each flow addition. The penalties of such splitting include the cumbersome reassignment operation, and in-sequence reassembly for a small number of flows. Irrespective of splitting, the PPS requires memory in the first and third stages. Due to these factors, and its limited amenability to throughput optimization, the flow-based PPS model, which subsumes currently available clustered routers, cannot be considered a particularly high-performance design.

## 7.3 Cell-based PPS

We now present and analyze a cell-based PPS, as an alternative to a flow-based one, in order to address the throughput limitations of the latter. Specifically, we concentrate on input-output flows,

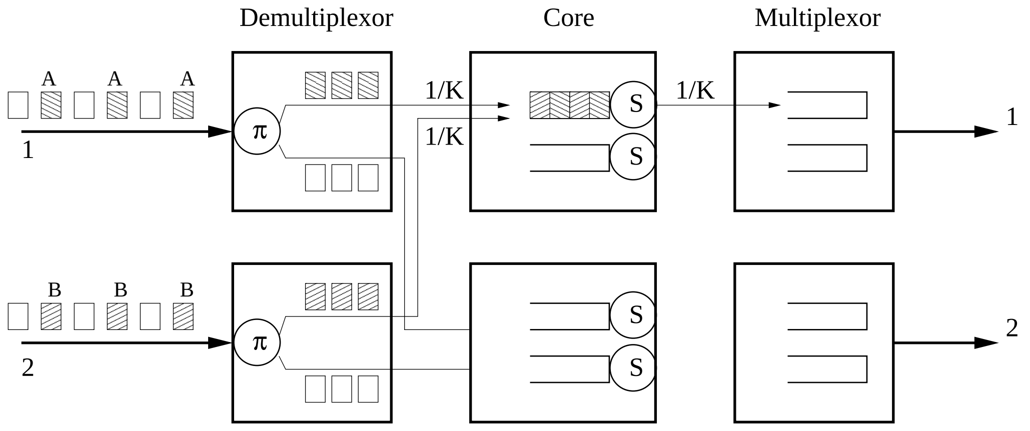


Figure 7.8: Cell-based PPS: Round-robin on arrivals leads to instability

with a goal to provide functional equivalence with an OQ switch, without explicit knowledge of their offered load. A secondary goal is to provide for a memoryless or small-memory first and third-stage elements, a possibility lacking in the flow-based model.

A cell-based PPS $(N, [xMx], N, K, s)$ dynamically assigns each arriving cell, from input $i$ to output $j$, to one of the possible $K$ paths between $(i, j)$. Since this decision is made on a cell-by-cell basis, inherently, the switch splits the traffic belonging to *each* flow, in some fashion determined by the path assignment method in the demultiplexor. The assignment results in a portion $\lambda_{i,j}^{(k)}$, of the offered average load $\lambda_{i,j}$ for each pair $(i, j)$, to be dispatched to each path $k \leq K$, such that $\sum_k \lambda_{i,j}^{(k)} = \lambda_{i,j}$. In other words, those portions are not pre-determined as in a flow-based PPS, rather, it just results as an outcome of cell-based path assignment. One obvious downside of this model is the need to perform in-sequence reassembly for each flow. This translates to a requirement of at least $N$ sequencing engines in each multiplexor.

We first rule out a tempting dispatch strategy, which might be contemplated to yield a memoryless first-stage. Recognize that a simple round-robin dispatch of the arriving cells, at each input, across the $K$ internal paths, removes the need for memory in the first stage, for $s = 1$. The dispatch rule is given by

$$\forall i, n, \quad \pi_{i,k}(n) = 1, \quad \text{iff } k = n \bmod K + 1$$

Therefore, $\forall n$, $\pi_{i,k}(n) = 1$ implies $\pi_{i,k}(n+k) = 0, k \leq K$, thereby ensuring that a path is never chosen before the prior cell to that path is fully dispatched. However, this may cause instability in some core elements, even for admissible traffic, as shown in Fig. 7.8. Here, flows $A$ and $B$, destined to the same output $j$ offer a load of $1/K$. If both the flows arrive at a constant bit-rate, all their cells may be dispatched to the same core element, whose output $j$ cannot handle the total offered load of $2/K$. No amount of speedup, less than the trivial value of $K$, is guaranteed to avoid such a concentration.

Fortunately, any algorithm that equalizes the offered load of each *flow*, across the $K$ paths, turns out to be a viable candidate to ensure stability. Though the switch dynamics are different in a PPS, the rationale behind balanced matchings for CIOQ-P may be beneficially applied here. Specifically, we show that PPS variants of the striping and equal dispatch methods suffice for relative stability with an OQ switch. For further intuition, consider a fluid model of a PPS switch, which divides the incoming fluid equally among all paths, at each instant. Clearly, the offered load for each subflow $(i, j, k)$ becomes $\lambda_{i,j}/K$, ensuring that the queues at each non-oversubscribed core-element output $j$ remain stable, for any work-conserving scheduler.

**Proposition 6.** *A fluid-model PPS, with equal dispatch of the incoming fluid across all the paths, at each instant, and a work-conserving scheduler at each link, is relatively stable ($f_3$) with respect to a fluid-model work-conserving OQ switch.*

Furthermore, the arrival patterns are identical at each core element, resulting in identical departures, assuming homogeneous and deterministic schedulers at their outputs. The central elements behave as mirror images of each other. These observations form the basis of the methods proposed in this section. Even though a cell-based PPS cannot balance the load at each infinitesimally small instant, we expect a similar approach, on a cell timeslot granularity, to yield similar long-term results. The additional challenge is to maintain those results without requiring memory elements in the first and third stages, while ensuring in-sequence delivery.

### 7.3.1 Envelope Striping

We first apply the striping method previously seen in the context of a CIOQ-P switch, in Sec. 6.2.3, to a cell-based PPS. Consider a PPS $(N, [xMx], N, K, s)$ with virtual output queues (VOQ) $(i, j)$, $j \leq N$, in each demultiplexor $i$. Let $X_{i,j}(n)$ denote the respective VOQ length at timeslot $n$. This

**Algorithm 27.** *Cell-based PPS: Envelope Striping*

```
Initialize:         1. For all $(i,j)$, $i \leq N$, $j \leq N$, $X_{i,j} \leftarrow 0$

                    For each demultiplexor $i$ in parallel
Every Timeslot $n$: If a cell arrives for $(i,j)$
                    1. Increment $X_{i,j}$ (Enqueue into VOQ)

                    If internal links are not busy
                    1. Select $(k,l)$ such that $X_{k,l} \geq K$
                    2. If $(k,l)$ found, do steps 2a-2b
                       2a. $X_{k,l} \leftarrow X_{k,l} - K$ (Dequeue $K$ cells)
                       2b. Dispatch a cell on each internal link
```

Figure 7.9: Cell-based PPS: Striping envelopes across the core elements

is a slight modification of the structure described in Sec. 7.1.2, in that the queues are not grouped on a per-path basis. As shown in Fig. 7.9, an incoming cell for flow $(i,j)$ is enqueued into VOQ $(i,j)$, and a path assignment is not made immediately. Instead, when all the internal links are simultaneously free, the demultiplexor selects a VOQ $(k,l)$ with an envelope worth of backlog, of size $K$. An entire envelope is simultaneously served, within $\lceil \frac{K}{s} \rceil$ external slots, by dispatching one cell from $(k,l)$ to each of the paths. Let $D_{X_{i,j}}(n)$ be the cumulative departures from VOQ $(i,j)$ until slot $n$. Then,

$$\forall i,n, \ \sum_{j=1}^{N} D_{X_{i,j}}(n + \left\lceil \frac{K}{s} \right\rceil) = \sum_{j=1}^{N} D_{X_{i,j}}(n) + K, \quad \text{if } \exists (k,l) \text{ s.t. } X_{k,l} \geq K \tag{7.30}$$

In other words, provided there is at least one full envelope, for any $s \geq 1$, $K$ cells are served from demultiplexor $i$ within $K$ external slots. This is sufficient for demultiplexor stability, irrespective of how $(k,l)$ is selected. In practice, the envelopes may be chosen in FIFO order of completed arrival. If there are no full envelopes, the maximum number of cells in the demultiplexor is upper bounded by $N(K-1)$. As with any envelope-based scheme, these may perpetually be starved if there are no further arrivals into those queues.

**Lemma 9.** *For a PPS under envelope striping and $s \geq 1$, the backlog at each demultiplexor is upper bounded by $N(K-1)+1$ cells.*

*Proof.* Let us designate an envelope arrival at slot $n$, if an arriving cell completes an envelope of $K$ cells. Similarly, an envelope departure is designated at slot $n$, if all the $K$ cells of an envelope finish their departure, on the $K$ internal links, at slot $n$. Consider the envelope arrival and departure processes, for $s = 1$.

Notice that the input link can sustain a rate of no greater than 1 envelope per $K$ slots. Since there may be a maximum of $N$ partial envelopes in a demultiplexor, with $K-1$ cells each, $N$ back-to-back cells may complete $N$ envelopes. In other words, the arrival process is a leaky-bucket [28] constrained source with sustained rate $1/K$, peak rate 1, and a maximum burst size of $N$. Fig. 7.10 shows the (interpolated) worst-case arrival curve for such a source, and the associated service curve, due to (7.30). The envelope backlog is the vertical distance between the two curves, which is upper bounded by

$$N - \frac{N-1}{K},$$

or $N(K-1)+1$ cells. □

These cell buffers may be shared among the $N$ flows. Due to this upper bound, which is independent of the arrival process, the demultiplexors may be implemented as small-memory elements, or $\tilde{S}$.

**Lemma 10.** *For a PPS under envelope striping and $s = 1$, the multiplexors may be memoryless.*

*Proof.* Let the cumulative arrivals, until timeslot $n$, for flow $(i,j)$ into core element $k$ be $A^{(k)}_{i,j}(n)$. Then, the striping process ensures that

$$\forall i,j,n,\ \forall k,l \leq K, \quad A^{(k)}_{i,j}(n) = A^{(l)}_{i,j}(n) \tag{7.31}$$

Therefore, each core element is subject to identical arrivals. Furthermore, the cells that arrive at input $i$, at slot $n$, one at each core element, belong to the same envelope and hence to the same flow. Therefore, assuming that the schedulers in the core elements are homogeneous, each such cell experiences the same delay.

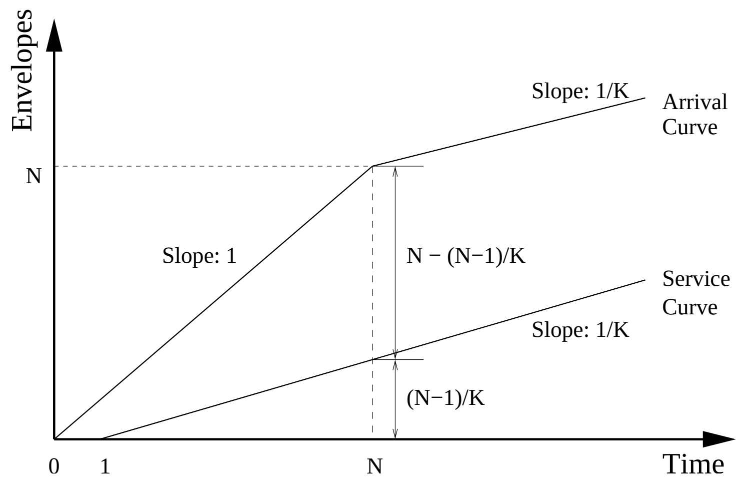


Figure 7.10: Arrival and service curves for envelopes in a PPS with striping

Consequently, cells belonging to the same envelope depart from the core element in the same slot, and can be immediately combined. For $s = 1$, therefore, all the arrivals from the core elements may be dispatched, without queueing, by the multiplexor. $\square$

Much as in the fluid model, the central elements behave as mirror images of each other. A significant side-effect is that no sequence control is needed. Note however that the above result does not hold for $s > 1$. While all the cells arriving into a multiplexor, in each slot, still belong to the same envelope, queueing will be required, to absorb the difference between the external link capacity and the aggregate capacity of the internal links, at each multiplexor. Nevertheless, with $s = 1$, most of our desired performance goals are met.

**Theorem 18.** *A cell-based PPS $(N, [\tilde{S}MS], N, K, 1)$, i.e., with memoryless multiplexors, demultiplexors with a buffer size bounded by $N(K-1)+1$ cells, and without speedup, operating under envelope striping and work-conserving core elements, provides asymptotic 100% throughput to flows destined to non-oversubscribed outputs, i.e.,*

$$\text{(PPS, }\{K\text{-Striping}\}) \overset{T,f_3}{\simeq} \text{(OQ, }\{\text{WC}\}),\ T\text{: SLLN},\ s = 1$$

*Proof.* Let the $N \times N$ matrices $(A_{i,j}^{(k)}(n), D_{i,j}^{(k)}(n), Q_{i,j}^{(k)}(n))$, defined in Sec. 7.1.2, describe the evolution of the VIQ system in core element $k$. Fix $s = 1$. Consider a non-oversubscribed output

$j$. By definition, the average arrival rates (2.1) for flows $(i, j)$, $i \leq N$, satisfy $\sum_i \lambda_{i,j} \leq 1$. From (7.31), we obtain

$$\begin{aligned} \lambda_{i,j}^{(k)} &= \lim_{n\to\infty} \frac{A_{i,j}^{(k)}(n)}{n} \\ &\leq \frac{1}{K} \lim_{n\to\infty} \frac{A_{i,j}(n)}{n} \\ &\leq 1/K \end{aligned}$$

Since there is no backpressure, therefore, from (7.14) and summing (7.13) over all inputs, the fluid limits (3.6) of $(A_{i,j}^{(k)}(n), D_{i,j}^{(k)}(n), Q_{i,j}^{(k)}(n))$ satisfy

$$\begin{aligned} \forall k,\ \sum_{i=1}^{N} \bar{Q}_{i,j}^{(k)}(t) &\leq \frac{t}{K} - \sum_{i=1}^{N} \bar{D}_{i,j}^{(k)}(t) \\ \sum_{i=1}^{N} \bar{D}_{i,j}^{(k)\prime}(t) &= 1/K, \quad \text{whenever} \quad \sum_{i=1}^{N} \bar{Q}_{i,j}^{(k)}(t) > 0, \end{aligned}$$

for all non-oversubscribed outputs $j$. From the Dai-Prabhakar fluid limit theorem, $\sum_{i=1}^{N} \bar{Q}_{i,j}^{(k)}(t) = 0$, $t \geq 0$. Therefore, all the queues at output $j$, for each core element, is rate-stable. From Lemmas 9 and 10, the queue lengths in the first and third stages are finite and zero, respectively. Therefore, every queue encountered by flows destined to $j$ remain stable, as in the reference OQ switch. □

Furthermore, assume, as we did in the analysis of CIOQ-P, that if $\mu$ is the average departure-rate matrix for offered rates $\lambda$ in an OQ switch, then, under the same scheduler, $\frac{1}{K}\mu$ would result for offered rates $\frac{1}{K}\lambda$. This is true for most flow-based schedulers such as weighted round-robin and WFQ-based ones. Then $f_4$ equivalence with the OQ switch follows, provided the output link schedulers in the core elements are the same as the link schedulers in the reference switch. These results continue to hold if a core element is replaced by a CIOQ switch that is functionally equivalent to an OQ switch, at the same level.

**Corollary 29.** *$((N, [\tilde{S}MS], N, K, 1)$, $\{K$-Striping$\})$ $\overset{T, f_4}{\simeq}$ (OQ, $\{$WC$\}$), T: SLLN*

Recognize that if bandwidth guarantees are desired for fine-grain flows $f$, with admissible rate reservations $R_f$, each core element scheduler may be programmed to provide trunks of

$R_f/K$. This may be used as a bandwidth trunk of $R_f$ by flow $f$, provided its arriving cells are distributed equally to each core element. Clearly, this necessitates creating envelopes and striping on a per-flow basis. In other words, a cell-based PPS may provide bandwidth guarantees with per-flow striping, and a total buffer size of $F(K-1)+1$ cells in each demultiplexor, where $F$ refers to the maximum number of flows associated with an input. This buffer size may prohibit the implementation of the first stage as $\tilde{S}$. Delay guarantees are beyond the scope of striping due to the unbounded latency associated with partially filled envelopes.

Thus, the envelope striping method, in a cell-based PPS, meets our goal of optimal throughput, without speedup. It allows for memoryless multiplexors and small-memory demultiplexors, and most beneficially, does not require explicit sequence control. The second-stage delay is inherently equalized for each per-path component of the same flow. Striping provides these desirable properties since, in essence, it emulates a fluid-model PPS with instantaneous equal dispatch (Proposition 6), but on an envelope timescale. Furthermore, bandwidth guarantees may be provided to individual fine-grain flows using per-flow envelope striping, but the buffers required in the first-stage grow as $O(F)$. The only disadvantage of this method is the unbounded delay associated with creating full envelopes. As opposed to a CIOQ switch with envelope-based matching, we cannot contemplate adding speedup to mitigate this issue, since, on doing so, the multiplexors will need to be implemented as memory elements.

### 7.3.2 Equal Dispatch

As we suggested for CIOQ-P switches, an equal dispatch method may be used as an alternative to striping, in order to address the unbounded latency issue of the latter. Indeed, we report here that all the material results, on throughput and QoS, for striping continue to hold for equal dispatch. Unlike a CIOQ-P switch, however, which benefits from a lack of any significant delay in the second stage, the penalty in a PPS, for such a dispatch, includes a cumbersome sequencing operation at each output.

Consider a PPS $(N, [xMx], N, K, s)$ with per-path FIFO queues $(i,k)$, $k \leq K$ in each demultiplexor $i$, as described in Sec. 7.1.2. Furthermore, let each core element $k$ employ virtual input queues (VIQ) $(i,j)$ for each subflow $(i,j,k)$, and let each multiplexor $j$ employ per-path

**Algorithm 28.** *Cell-based PPS: Equal Dispatch*

```
Initialize:          1. For all $(i,k)$, $i \le N$, $k \le K$, $X_{i,k} \leftarrow 0$
                     2. For all $(i,j)$, $P_{i,j} \leftarrow 1$

                     For each demultiplexor $i$ in parallel
Every Timeslot $n$:  If a cell arrives for $(i,j)$
                     1. $k \leftarrow P_{i,j}$
                     2. Increment $X_{i,k}$ (Enqueue into FIFO)
                     3. Increment $P_{i,j}$ mod $K$

                     For each internal link $k$ in parallel
                     If link not busy, and $X_{i,k} > 0$
                     1. Decrement $X_{i,k}$ (Dequeue from FIFO)
```

Figure 7.11: Cell-based PPS: Per-flow equal dispatch across core elements

VIQ $(i,j,k)$. Note that VIQ $(i,j)$ in core element $k$ operates in tandem with VIQ $(i,j,k)$ in multiplexor $j$, as shown in Fig. 7.5. As shown in Fig. 7.11, a multiplexor $i$ maintains a round-robin pointer for each flow $(i,j)$, $j \le N$. When a cell arrives for $(i,j)$, it is dispatched to path $k$, determined by its own round-robin pointer, by enqueueing it into FIFO $(i,k)$. The FIFOs are drained into their respective paths, in a work-conserving fashion, at a rate of $s/K$ cells per external slot. Clearly, therefore, the dispatch satisfies the following:

$$\begin{aligned}\forall i,k,n,\;\; \pi_{i,k}(n) &= 1, \qquad\quad \text{iff } \exists j \text{ s.t. } A_{i,j}(n) > A_{i,j}(n-1) \text{ and } P_{i,j}(n) = k \\ \forall i,j,n\;\; P_{i,j}(n+1) &= (P_{i,j}(n)+1) \bmod K,\;\; \text{if } A_{i,j}(n) > A_{i,j}(n-1)\end{aligned}$$

As a result, the arrivals $A^{(k)}_{i,j}$ into core element $k$, for flow $(i,j)$, satisfies (similar to (6.20) for CIOQ-P):

$$\forall i,j,k,n,\quad A^{(k)}_{i,j}(n) \le \left\lfloor \frac{A_{i,j}(n)}{K} \right\rfloor + 1_{\{A_{i,j}(n) \bmod K \ge k\}} \tag{7.32}$$

The inequality is because some cells of flow $(i,j)$ may still be in the demultiplexor at slot $n$. Note that this is a round-robin dispatch per flow, which is not the same as input-stream round-robin shown in Fig. 7.8. If the pointers for more than one flow become synchronized, back-to-back cells may be dispatched to the same per-path FIFO, at a peak rate of 1. As each FIFO is served at a sustained rate of $s/K$, whenever non-empty, this may lead to a per-path backlog in the demultiplexor. An example of such dispatch is shown in Fig. 7.12.

**Lemma 11.** *For a PPS under equal dispatch and $s \geq 1$, the backlog at each demultiplexor is upper bounded by $NK$ cells.*

*Proof.* The proof is similar to that of Lemma 9. The input link can sustain a rate of no greater than 1 cell per $K$ slots into each per-path FIFO. As $N$ consecutive cells, each belonging to a different flow, may enter a FIFO back-to-back, the cell arrival process into each FIFO is leaky-bucket constrained, with sustained rate $1/K$, peak rate 1, and a maximum burst size of $N$. Since each FIFO is served at a constant rate of $s/K$, for $s \geq 1$, the backlog is bounded by

$$N - \left\lfloor \frac{N-1}{K} \right\rfloor .$$

Therefore, each FIFO may contain no more than $N$ cells, at all instants, and the total backlog at the demultiplexor is upper bounded by $NK$ cells. □

As a result, the demultiplexors may be implemented as small-memory elements $\tilde{S}$. Note that this is true for any dispatch method that equalizes the per-flow load over any finite-sized sequence of back-to-back cells. For instance, we introduced the *uniform dispatch* method in [62], which ensures that no more than $\lceil \frac{M}{K} \rceil$ cells are distributed to the same path in a sequence of $M$ cells, for each flow. In that case, the backlog in a demultiplexor is upper bounded by $NM$ cells.

Consider a PPS without backpressure from the multiplexors to the core elements. Clearly, for $s = 1$, we may employ a memoryless multiplexor if in-sequence delivery is not required. The following result establishes that, with memories and cell sequencing in the third stage, equal dispatch suffices for relative stability with an OQ switch.

**Theorem 19.** *A cell-based PPS $(N, [\tilde{S}MM], N, K, s)$, with demultiplexors of buffer size $NK$ cells, and $s \geq 1$, operating under equal dispatch and work-conserving elements, provides asymptotic 100% throughput to flows destined to non-oversubscribed outputs, i.e.,*

$$\text{(PPS, \{Equal Dispatch\})} \overset{T, f_3}{\simeq} \text{(OQ, \{WC\})},\ T\text{: SLLN},\ s \geq 1$$

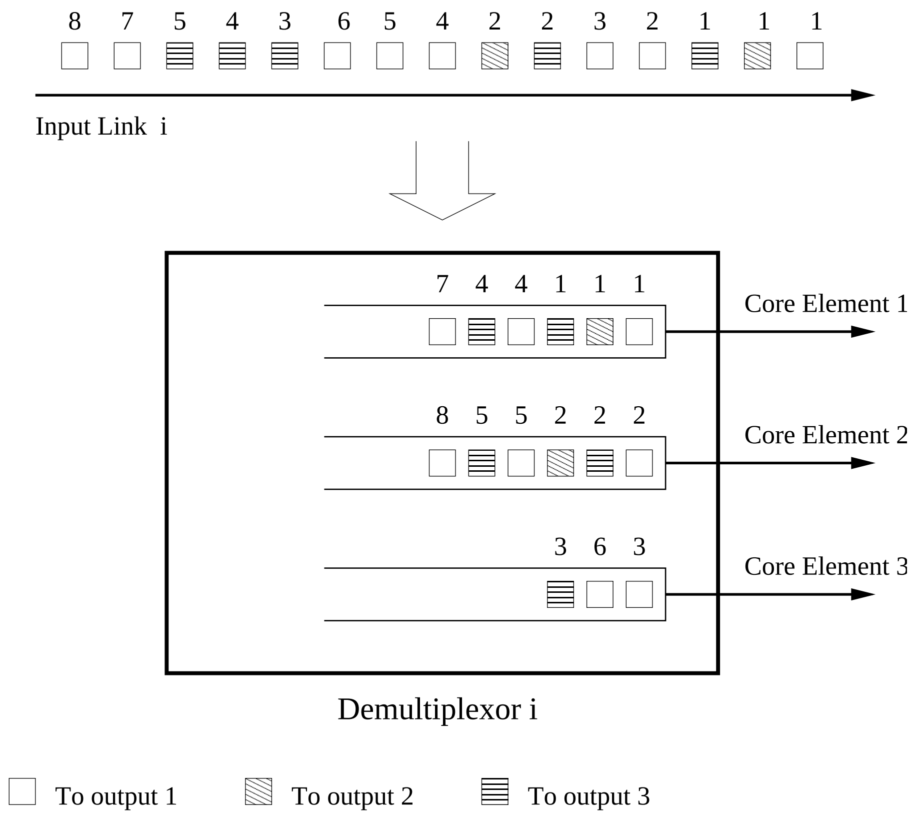


Figure 7.12: Example of per-flow equal dispatch in a PPS with $N = 3$, $K = 3$

*Proof.* First, consider core element $k$. From (7.32), the average rate of arrival into VIQ $(i, j, k)$ does not exceed $\lambda_{i,j}/K$, where $\lambda_{i,j}$ denotes the long-term average rate offered by flow $(i, j)$ (2.1). Summing (7.13) over all the inputs, we obtain

$$\forall j, k, n \quad \sum_{i=1}^{N} Q_{i,j}^{(k)}(n) = \sum_{i=1}^{N} A_{i,j}^{(k)}(n) - \sum_{i=1}^{N} D_{i,j}^{(k)}(n)$$

Combining with (7.14), since $B_{i,j}^{(k)}(n) = 1, \forall i, j, k, n$, due to the absence of backpressure, each output link $j$ dispatches cells at a rate of $s/K$, whenever $Q_{i,j}^{(k)} > 0$, for some $i$. For non-oversubscribed outputs $j$, the average rate of arrival cannot exceed $\sum_i \lambda_{i,j}/K \leq 1/K$. Therefore, each VIQ $(i, j, k)$ is rate-stable, for $s \geq 1$.

Next, consider multiplexor $j$ for a non-oversubscribed output. Let each VIQ $(i, j, k)$ be split into an eligible component, with length $W_{i,j}^{(k)}(n)$, and an ineligible component, with length

$Y_{i,j}^{(k)}$, waiting for cells ahead of its head, in arrival sequence for flow $(i, j)$, to arrive. Then, by definition,

$$\forall i, j, k, n, \;\; Z_{i,j}^{(k)}(n) = W_{i,j}^{(k)}(n) + Y_{i,j}^{(k)}(n) \tag{7.33}$$

For the eligible component, from (7.13), (7.15), and (7.16), we obtain

$$\begin{aligned}
\forall j, n, \;\; \sum_{k=1}^{K}\sum_{i=1}^{N} W_{i,j}^{(k)}(n) &\leq \sum_{k=1}^{K}\sum_{i=1}^{N} A_{i,j}^{(k)}(n) - \sum_{k=1}^{K}\sum_{i=1}^{N} D_{Z_{i,j}^{(k)}}(n) \\
\forall j, n, \;\; \sum_{k=1}^{K}\sum_{i=1}^{N} D_{Z_{i,j}^{(k)}}(n+1) &= \sum_{k=1}^{K}\sum_{i=1}^{N} D_{Z_{i,j}^{(k)}}(n) + 1, \;\; \text{if} \; \sum_{k=1}^{K}\sum_{i=1}^{N} W_{i,j}^{(k)}(n) > 0
\end{aligned}$$

For a non-oversubscribed output $j$, the average rate of arrival, associated with the second summation in the first line, cannot exceed 1. Therefore, the combined occupancy of $W_{i,j}^{(k)}$, $\forall i, k$, is stable, resulting in the stability of the eligible components of each VIQ.

The ineligible component of VIQ $(i, j, k)$ can increase only as long as the head cell, for flow $(i, j)$, remains inside some core element $l$, $l \neq k$. Since the VIQ $(i, j)$ in element $l$ is stable, for non-oversubscribed $j$, the expected delay of the head cell is finite. From Little's law, therefore, the expected value of $Y_{i,j}^{(k)}$ is also finite. Thus, $Q_{i,j}^{(k)}$ and $Z_{i,j}^{(k)}$ are both stable, $\forall i, k$. The final result, for $s \geq 1$, follows by combining this with Lemma 11. □

The proof relies on work-conserving behavior at each multiplexor, whenever it contains eligible cells. This implicitly assumes that the sequencing engine does not suffer from redundant wait. We provided an alternative proof in [64], which established that, if the maximum differential delay, across different paths, is bounded (as opposed to the expected value being finite) and the arrivals are bounded in a finite timescale, the multiplexor buffer-size can also be bounded. However, this bound may be too large to allow a small-memory implementation for the multiplexor, as it depends on arrival properties. Notice that, in addition to the requirement of a well-behaved sequencing engine, the proof only relies on the fact that the average arrival rate of a subflow $(i, j, k)$ does not exceed $\lambda_{i,j}/K$. Also, it is evident that the result continues to hold if the core elements are replaced by a CIOQ switch that provides wide-sense relative stability with an OQ switch[5]. We further stipulate, using the same rationale as Corollary 29, that, if the core

[5]The proof would be identical to the one for Theorem 19, with VIQ $(i, j)$ in the core replaced by a tandem of VOQ $(i, j)$ in the input element of the CIOQ switch and a VIQ $(i, j)$ in the output element, and a matching algorithm that ensures the stability of both.

element schedulers are identical to the one in a reference OQ switch, equal dispatch provides $f_4$ equivalence with the latter.

### Equal Dispatch and Backpressure

The equal dispatch method, without backpressure, provides throughput performance comparable to envelope striping, without the latency issue of the latter. However, the third stage requires memory elements to ensure work-conserving behavior in all stages. Note that a memory architecture may be designed with each VIQ implemented as a separate physical memory. Since a VIQ is accessed at most once in $K$ timeslots, due to equal dispatch, the bandwidth required for such memories is a fraction of the external link rate, thus realizing the memory bandwidth benefit of a PPS, inspite of a $\tilde{S}MM$ design. We now establish that the same throughput result holds even with small-memory third-stage elements, using backpressure.

Consider a multiplexor with a buffer size of exactly one cell per subflow. Therefore, each third-stage element contains no more than $NK$ cells, and a backpressure signal is asserted to VIQ $(i, j)$ in core element $k$, whenever $Z_{i,j} = 1$. In other words, we may modify (7.17) as

$$\forall i, j, k, n, \;\; B^{(k)}_{i,j}(n) = 1_{\{Z^{(k)}_{i,j}(n)=0\}} \tag{7.34}$$

For simplicity, we assume that $B^{(k)}_{i,j}$ is set to 1 at the instant that a non-empty VIQ $(i, j, k)$ is served at multiplexor $i$. Similarly, it is reset at the instant at which core element $k$ begins transmission of a cell from VIQ $(i, j)$. Furthermore, we assume that a core element is work-conserving (7.14) at each output $j$, whenever $B^{(k)}_{i,j} = 1$ for some non-empty VIQ $(i, j)$. Since the buffer size at each multiplexor is finite, we may claim 100% throughput for a flow $(i, j)$ by merely establishing the stability of VIQ $(i, j)$, with length $Q^{(k)}_{i,j}(n)$, in each core element $k$.

**Lemma 12.** *For a PPS under equal dispatch, with $s \geq 1$, whenever there is a cell at output $j$ of core element $k$, i.e., $\sum_i Q^{(k)}_{i,j}(n) > 0$, the inner product $\sum_i Q^{(k)}_{i,j}(n+T)B^{(k)}_{i,j}(n+T) > 0$, for some finite $T \geq 0$.*

*Proof.* Fix output $j$ and core element $k$. Let $\sum_i Q^{(k)}_{i,j}(n) > 0$ at timeslot $n$. If $\sum_i Q^{(k)}_{i,j}(n)B^{(k)}_{i,j}(n) > 0$, the result is proved with $T = 0$.

Let $\sum_i Q_{i,j}^{(k)}(n)B_{i,j}^{(k)}(n) = 0$. In other words, every non-empty VIQ at output $j$ is backpressured. Therefore, from (7.34), we obtain

$$Z_{i,j}^{(k)}(n) = 1, \quad \forall i \text{ s.t. } Q_{i,j}^{(k)}(n) > 0$$

*Case 1:* Let VIQ $(i, j, k)$ contain the head cell of flow $(i, j)$, for some $i$ such that $Q_{i,j}^{(k)}(n) > 0$. Then, flow $(i, j)$ will be served within slot $(n + \Delta)$, where $\Delta$ is the finite latency of the link scheduler in the multiplexor. Consequently, $B_{i,j}^{(k)}(n + T) = 1$ for some $T \leq \Delta$, satisfying the result.

*Case 2:* Let there be no head cells in VIQ $(i, j, k)$, for all $i$ such that $Q_{i,j}^{(k)}(n) > 0$. Due to equal dispatch, there may be at most $N(K-1)$ cells that need to be scheduled before one of VIQ $(i, j, k)$ contains a head cell. Since any head cell, at all points in time, is either at a multiplexor VIQ, or at the head of the respective core element queue, without being backpressured, or at the demultiplexor, it takes no more than

$$N\frac{K}{s} + N\frac{K}{s} + 1$$

slots to dispatch each such cell. The first term accounts for the delay in the demultiplexor, the second for the worst-case time to transmit a head cell from the core to the multiplexor, and the last for scheduling it on the external link. Therefore, for $s \geq 1$, and some $T \leq N(K-1)(2NK+1)$, $B_{i,j}^{(k)}(n + T) = 1$ for some $i$ such that $Q_{i,j}^{(k)}(n + T) > 0$. $\square$

**Theorem 20.** *A cell-based PPS $(N, [\tilde{S}M\tilde{S}], N, K, s)$, with demultiplexors and multiplexors of buffer size $NK$ cells, and $s \geq 1$, operating under equal dispatch and backpressure, provides asymptotic 100% throughput to flows destined to non-oversubscribed outputs, i.e.,*

$$\text{(PPS, \{Equal Dispatch\})} \overset{T,f_3}{\simeq} \text{(OQ, \{WC\})}, T\text{: SLLN}, s \geq 1$$

*Proof.* Let $j$ be a non-oversubscribed output. Let $(\bar{A}_{i,j}^{(k)}(t), \bar{D}_{i,j}^{(k)}(t), \bar{Q}_{i,j}^{(k)}(t))$ be the fluid limits (3.6) of the respective discrete-time variables. From Lemma 12, we obtain

$$\forall j, k, t, \ \lim_{r\to\infty} \frac{1}{r}\sum_{i=1}^{N} Q_{i,j}^{(k)}(rt - T) > 0 \Rightarrow \lim_{r\to\infty} \frac{1}{r}\sum_{i=1}^{N} Q_{i,j}^{(k)}(rt)B_{i,j}^{(k)}(rt) > 0$$

for some finite $T \geq 0$. Combining with the fluid limits of (7.13) and (7.14), the above leads to the following evolution of the VIQ system in the core elements.

$$\begin{aligned}
\forall j,k,t \ \sum_{i=1}^{N} \bar{Q}_{i,j}^{(k)}(t) &= \sum_{i=1}^{N} \lambda_{i,j}^{(k)} t - \sum_{i=1}^{N} \bar{D}_{i,j}^{(k)}(t) \\
&\leq \frac{t}{K} - \sum_{i=1}^{N} \bar{D}_{i,j}^{(k)}(t), \quad \text{if } j \text{ is not oversubscribed} \\
\sum_{i=1}^{N} \bar{D}_{i,j}^{(k)\prime}(t) &= \frac{s}{K} \quad \text{if } \sum_{i=1}^{N} \bar{Q}_{i,j}^{(k)}(t) > 0
\end{aligned}$$

Therefore, the aggregate of the VIQ system at each non-oversubscribed output, for all the core elements, is rate-stable for $s \geq 1$. □

We stipulate, using the same rationale as Corollary 29, that, if the core element schedulers are identical to the one in a reference OQ switch, equal dispatch provides $f_4$ equivalence with the latter, using backpressure. Unlike Theorem 19, however, we cannot extend the above result easily to a PPS with CIOQ switches for core elements. Lemma 12 does not hold anymore for a finite $T$ since a head cell may reside in the input element of the CIOQ. Even if the respective VOQ is stable, the delay becomes an unbounded variable, though with a finite expected value. We are uncertain whether $g(n) = f(n + T(n))$ with finite $E[T(n)]$ implies $\bar{g}(t) \rightarrow \bar{f}(t)$, a condition that is required in the last inequality of the above proof. It is likely that we are limited primarily by our proof methodology.

Much as with envelope striping, bandwidth guarantees may be provided to fine-grain flows $f$, with reservation $R_f$, by programming the incident output schedulers, in each core element, to provide a trunk of $R_f/K$. In addition, the respective multiplexor, with or without backpressure, is programmed to provide a trunk of $R_f$. Equal dispatch is carried out on a per-flow basis. This necessitates a buffer size of $FK$ cells in the demultiplexor, and in the multiplexor, in case backpressure is utilized, where $F$ is the maximum number of flows at each input and output. As an advantage over striping, delay bounds may be provided to conforming flows. For example, for a leaky-bucket constrained flow between $(i, j)$, with bucket size $\sigma_f$, the delay is bound by

$$D_f \leq FK + \frac{\sigma_f}{R_f} + \max_{k \leq K}(\beta_{j,k}) + \gamma_j,$$

**Algorithm 29.** *Cell-based PPS: Fractional Dispatch*

```
Initialize:          1. For all $(i,j)$, $P_{i,j} \leftarrow \{1, 2, \ldots, \lceil \frac{K}{s} \rceil - 1\}$

                     For each demultiplexor $i$ in parallel
Every Timeslot $n$:  If a cell arrives for $(i,j)$
                     1. Select $k \leq K$ s.t. $k \notin P_{i,j}$ and link $k$ not busy
                     2. $\pi_{i,k} \leftarrow 1$ (Dispatch to path $k$)
                     3. Remove head of $P_{i,j}$
                     4. Insert $k$ at the tail of $P_{i,j}$
```

Figure 7.13: Cell-based PPS: Per-flow fractional dispatch across core elements

where $\beta_{j,k}$ is the latency of the output scheduler $j$ in core element $k$, and $\gamma_j$ of the link scheduler in multiplexor $j$. The first term is because of the per-path FIFO dispatch in the demultiplexor, and the second due to the fact that each core element receives $1/K$ of a burst.

### 7.3.3 Fractional Dispatch

We now show that a speedup of $s > 1$ may be judiciously employed to render the first-stage elements memoryless, while maintaining the throughput and QoS properties of equal dispatch. Consider a PPS $(N, [SMM], N, K, s)$ operating under a *fractional dispatch* method, shown in Fig. 7.13. Each flow is associated with a list $P_{i,j}$, which keeps track of the previous $(\lceil \frac{K}{s} \rceil - 1)$ path assignments for that flow. When a cell arrives for $(i,j)$, a path $k$ is chosen such that it is not a member of $P_{i,j}$, and the internal link to core element $k$ is free. The cell is then immediately placed on link $k$, thus ensuring a memoryless first stage. If such a link is guaranteed to exist at all timeslots, each arriving cell is successfully assigned.

Notice that in any sequence of $\lceil \frac{K}{s} \rceil$ back-to-back arrivals, for each flow, the same core element is chosen no more than once. Therefore, the cumulative arrivals $A_{i,j}^{(k)}(n)$ into central-stage element $k$, for flow $(i,j)$, satisfies

$$\forall i, j, k, n, \quad A_{i,j}^{(k)}(n) \leq \frac{A_{i,j}(n)}{\lceil \frac{K}{s} \rceil} \tag{7.35}$$

Therefore, the average rate of arrival $\lambda_{i,j}^{(k)}$ for subflow $(i, j, k)$ cannot exceed $\lambda_{i,j}s/K$, where $\lambda_{i,j}$ denotes the long-term average rate offered by flow $(i, j)$ (2.1). In essence, we are fully utilizing the fact that each internal link can sustain a rate of $s/K$, and therefore we may relax the strict requirement of equal dispatch, which mandates that each element must experience a load of exactly $\lambda_{i,j}/K$ for each flow. Fractional dispatch, in other words, allows for a controlled unbalancing of the offered load. Note however that it is possible for a path to be excluded for an indefinite amount for time.

**Lemma 13.** *A PPS under fractional dispatch and memoryless demultiplexors is guaranteed to assign a path to each arriving cell, provided* $s \geq \frac{K}{\lceil K/2 \rceil}$.

*Proof.* Recognize that in each external timeslot, no more than $\lceil \frac{K}{s} \rceil - 1$ internal links may be busy. Therefore, there are at least $(K - \lceil \frac{K}{s} \rceil + 1)$ free paths on each cell arrival. Since the size of $P_{i,j}$ is exactly $\lceil \frac{K}{s} \rceil - 1$, a free path satisfying fractional dispatch is guaranteed to exist provided

$$\begin{aligned} K - \left\lceil \frac{K}{s} \right\rceil + 1 &\geq \left\lceil \frac{K}{s} \right\rceil \\ \text{or} \quad \left\lceil \frac{K}{s} \right\rceil &\leq \frac{K+1}{2}, \\ \text{i.e., provided} \quad \frac{K}{s} &\leq \left\lfloor \frac{K+1}{2} \right\rfloor \\ \text{or} \quad s &\geq \frac{K}{\left\lfloor \frac{K+1}{2} \right\rfloor} = \frac{K}{\left\lceil \frac{K}{2} \right\rceil} \end{aligned}$$

□

Fig. 7.14 shows the speedup required as $K$ varies. As $K$ increases, this compares favorably with the lower bound in Fig. 7.7 for static dispatch using Clos fitting, without the necessity of first-stage memory as in the latter. Notice also that the maximum subflow load is forced to be no greater than $s/K$, as opposed to it being a requirement in Clos fitting. Due to Lemma 13, we may extend Theorem 19 as follows, for a PPS with memories in the third stage and no backpressure.

**Theorem 21.** *A cell-based PPS* $(N, [SMM], N, K, s)$, *with memoryless demultiplexors, and* $s \geq \frac{K}{\lceil K/2 \rceil}$, *operating under fractional dispatch and work-conserving elements, provides asymptotic 100% throughput to flows destined to non-oversubscribed outputs, i.e.,*

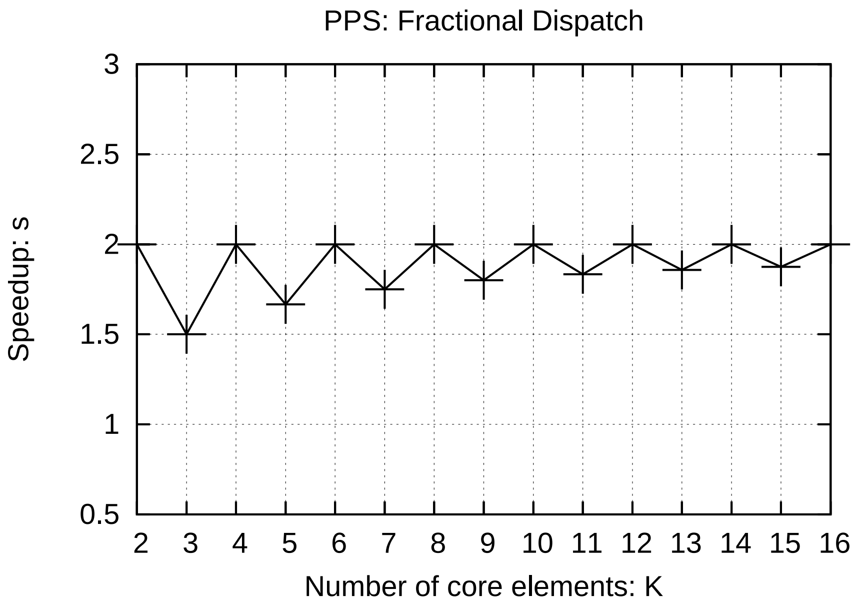


Figure 7.14: PPS: Speedup required for fractional dispatch

$$(\text{PPS}, \{\text{Fractional Dispatch}\}) \overset{T,f_3}{\simeq} (\text{OQ}, \{\text{WC}\}), T\text{: SLLN}, s \geq \frac{K}{\lceil \frac{K}{2} \rceil}$$

*Proof.* The proof is identical to that of Theorem 19. Each core element VIQ $(i, j)$, for non-oversubscribed $j$, is stable since the average rate of arrival into output $j$ cannot exceed $s/K$, equal to the service rate of the attached link. Each VIQ $(i, j, k)$ in multiplexor $j$ is also stable, since the average arrival rate cannot exceed 1, provided the sequencing engine does not suffer from redundant wait. □

Again, it is evident that the result continues to hold if the core elements are replaced by a CIOQ switch that provides wide-sense relative stability with an OQ switch. We further stipulate that if the core element schedulers are identical to the one in a reference OQ switch, fractional dispatch provides $f_4$ equivalence with the latter. Unfortunately, however, we have not been able to conclusively establish the above result for a PPS with small-memory multiplexors and back-pressure. Since a path may be excluded from dispatch for an indefinite amount of time, Case 2 of Lemma 12 does not hold anymore for a finite $T$. Consequently, we were unable to extend Theorem 20 towards fractional dispatch. Nevertheless, the above remains a significant result, since, if each VIQ in the multiplexor is implemented as a separate physical memory, each such memory is accessed no more than once every $\lceil \frac{K}{s} \rceil$ external slots, allowing us to realize the PPS memory-bandwidth benefit. As with equal dispatch, per-flow bandwidth and delay guarantees may be provided by performing fractional dispatch on a per-flow basis.

Combining the results of equal and fractional dispatch, we know that the buffer-size bound in the demultiplexors goes from $NK$ cells to 0 as the speedup increases from 1 to $\frac{K}{\lceil K/2 \rceil}$. An interesting open problem [62] is to determine how the bound varies for intermediate speedup values, for a continuum of policies between equal and fractional dispatch.

### 7.3.4 Related Work

Iyer and McKeown made some pioneering contributions [49, 50] towards the understanding of a PPS around the same timeframe of our research work. Specifically, in [49], the focus was on a PPS with memoryless first and third stages. It was shown that such a switch can exactly emulate an OQ switch with a FIFO discipline, using a speedup of approximately 2. The proposed emulation algorithm, called the *centralized PPS algorithm* (CPA), maintains a shadow OQ switch, and for every incoming cell at slot $n$, for flow $(i, j)$, determines the departure time $m$ in the reference. Similar to our proof of Theorem 21, a core element with a free input $i$ at slot $n$, and a free output $j$ at slot $m$, is guaranteed to exist, with a similar speedup as in our proof.

In [50], a distributed algorithm, resembling the equal dispatch method, was proposed towards the same goal. Similar to our stability results, it was shown that a PPS, without speedup and small-memory first and third stages, can emulate a FIFO OQ switch. To overcome the issue with FIFO queueing, in the core element, which we showed in Fig. 7.4, delay equalization [51] may be required at the inputs of the central stage. While these results are restricted to FIFO dispatch, they mirror our stability results for work-conserving schedulers. A similar load equalization algorithm has since been proposed by Wang et al., in [113], which specifically addresses a switch with a CIOQ central stage.

## 7.4 Sequence Control

Most multi-path switches[6], including the PPS and the recursive G-MSM seen in Sec. 6.3.3, suffer from out of sequence arrivals at the egress element, due to the differential delays experienced by the *subflows*. As opposed to a multi-path point-to-point environment, differential delays occur in the switched environment due to uneven cross-traffic experienced by the subflows. If $d_{\max}$ is the bound on differential delay, and $\mu$ the upper bound on the service rate received by a flow, then the amount of mis-sequencing, calculated as the volume of traffic that can arrive *before* the next-in-sequence cell, is bound by $d_{\max}\mu$. While it is beneficial to have dispatch schemes in which the output order is pre-computed, e.g., as in the CPA algorithm, or for which $d_{\max}$ is limited and can be accurately ascertained, in general, both those objectives might be infeasible. Accordingly, we need sequence control schemes that properly deal with the factors in the switching environment. Such schemes should necessarily be open-loop because of the latency associated with feedback-based schemes, and should work with a fixed overhead which is limited by the available bits in the local header of a cell.

In addition to possibly large or inestimable differential delays, two other factors, in practice, make open-loop sequence control within a switch challenging. The presence of finite buffers in the paths opens up the possibility of *cell loss*, which may go undetected by the re-assembler. This situation causes the output to idle even when an eligible cell is already present in the output element, the phenomenon we referred to as *redundant wait*. Presumably, this situation can be handled by using the information in the next cell that arrives from the same path, or if all the per-path re-assembly queues have at least one cell to offer. However, the effectiveness of such a strategy is diminished if the dispatch is uneven, especially if it causes certain paths to be *excluded* from the dispatch for an arbitrary duration, e.g., as in fractional dispatch. In the presence of excluded paths, redundant wait may also be handled by using a timeout based on the estimate $d_{\text{est}}$ of $d_{\max}$. The scheme then becomes heavily dependent on the accuracy of $d_{est}$. Specifically, if $d_{\max}$ is underestimated, then there is a possibility of *late arrivals* of cells which need to be handled gracefully, and if $d_{\max}$ is overestimated, the size of the re-assembly buffers increases,

[6]The contents of this section were previously presented in [64]. As a departure from the rest of the this work, this section contains simulation results. We urge the reader to use such results only to gain some flavor about the issues being addressed here.

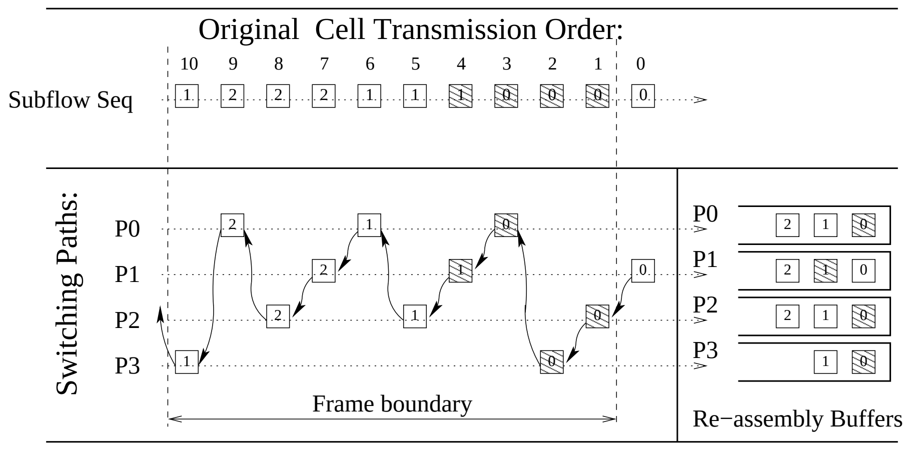


Figure 7.15: Sequence control using SCIMA

possibly leading to instability.

We presented and compared three open-loop schemes in [64]. The first is the classical total-order sequencing scheme, in which the ingress element simply appends a sequence number (or timestamp) to each cell before it is dispatched to a path. The range $N$ is limited by the available number of bits $B_{\mathrm{TOT}}$ in the fixed local header of the cell. The egress element uses a timeout based on $d_{\mathrm{est}}$ to detect losses in the presence of excluded paths. Clearly, the re-assembly is guaranteed to be unambiguous as long as the egress is never presented with two cells carrying the same timestamp, which is ensured as long as

$$N = 2^{B_{\mathrm{TOT}}} \geq d_{\mathrm{est}}\mu$$

Furthermore, the correctness of the re-assembly is guaranteed as long as $d_{\mathrm{est}}$ is an overestimate. The same ensures the stability of the system even in the presence of redundant wait since the total number of cells waiting to be re-assembled is bound by $d_{\mathrm{est}}\mu$. Therefore, total order sequencing is recommended, even in the presence of an arbitrary amount of cell losses, if the delay can be overestimated and the above value can be contained within the fixed cell header.

We proposed a sequencing scheme called Switched Connections Inverse Multiplexing for ATM (SCIMA) in [18] for multi-path systems, for which it is not possible to accurately estimate delay. The working of SCIMA is illustrated in Fig. 7.15. The ingress element uses a *predictive*

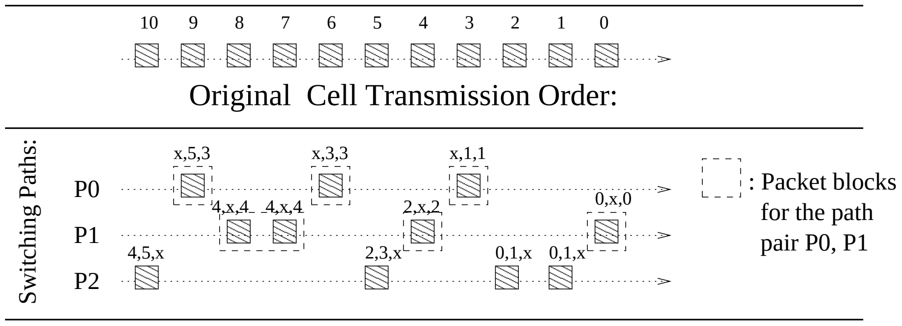


Figure 7.16: Sequence control using the rank-based protocol

dispatch scheme[7] and appends, to every header, a pointer to the path where the next cell of the same flow is to be scheduled, which we call the *NextPath* value. The egress element simply traverses through the NextPath chain to re-order the cells. In addition, each cell also contains a subflow sequence number, of range $S$, to correctly detect up to $S$ consecutive losses on a path. In order to recover the NextPath chain in the presence of losses, there are two separate means offered by the scheme. The cell header contains a history of $s(s \leq S)$ NextPath values belonging to cells that were previously transmitted on the same path. If $l(l \leq s)$ back-to-back losses are detected by the egress element, the re-assembly can proceed by looking at the $l$th value in the NextPath history.

In the case when $s < l \leq S$, the scheme uses what we refer to as the *frame recovery* mode. For this purpose, the ingress element delineates the incoming stream, of each flow, into frames and appends a current frame number to each cell. The first cell of each frame is explicitly marked (shaded in Fig. 7.15). When an unrecoverable loss burst is detected on a path, or when the re-assembly times out waiting for a cell on the expected path, the egress element *flushes* all the cells belonging to its current frame and re-synchronizes on the next frame for all the subflows. The overhead for SCIMA can be calculated as:

$$B_{\text{SCIMA}} = \log S + (s+1)\lceil \log K \rceil + f + 2,$$

where $F = 2^f$ is the range of the frame number, and the 2 extra bits are required for marking

[7]Predictive dispatch is feasible when the outgoing order can be pre-ascertained, as is the case with equal dispatch in a cell-based PPS, or static dispatch in a flow-based PPS with split flows.

| | **Delay Estimates** | **Loss Bounds** |
|---|---|---|
| **Total Order** | Overestimate $d_{est}$ for correctness; header size dependence | No dependence |
| **SCIMA** | Low, indirect dependence inside advanced frame recovery mode | Loss burst $l < s$ for provable correctness; $l < S$ for frame recovery |
| **Rank** | Overestimate $d_{est}$ for correctness | Loss burst size $l < 2^m - 2$ for provable correctness |

Table 7.1: Sequence Control: Dependence on loss bounds and delay estimates

cells. Note that the overhead does not depend on $d_{\text{est}}$. The frame sizes $S_f$ are chosen in such a way that frame numbers do not wrap around within $d_{\max}$, i.e.,

$$S_f \times F \gg d_{\text{est}}\mu$$

In the case of an inaccurate $d_{\text{est}}$, frame numbers allow to identify late arrivals. The downside is that a large $S_f$ leads to a large number of already enqueued cells being flushed inside the recovery mode. Therefore, this scheme is recommended when it is not possible to accurately estimate delay, however, only if the loss burst on a path can be bounded by $S$ to ensure correct re-assembly.

The rank-based protocol [61], illustrated in Fig. 7.16, also relies on bounding the loss burst, but without the need for flushing cells. This scheme defines a *packet block* on a path $A$ with respect to another path $B$ as any sequence of cells belonging to a flow which are transmitted over path $A$ with no interleaving cells of the same flow transmitted on path $B$. Packet blocks are constructed for every one of the $K(K-1)$ path pairs, for each flow. The ingress element enumerates the blocks for every pair, and appends a vector of packet block numbers into every header. The $i$th component of the vector is the number of the current block with respect to (the other) path $i$. If each component is of size $m$, then the cell overhead for this scheme is calculated as

$$B_{\text{Rank}} = m \cdot (K-1)$$

The egress element makes a pair-wise comparison, of the vector components, of the head cells of each subflow to determine the next-in-sequence cell. The scheme also uses a $d_{\text{est}}$ to time out

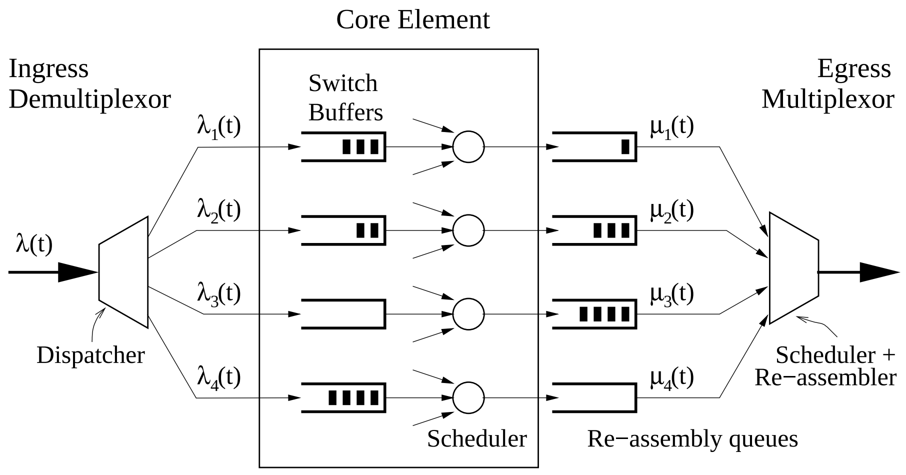


Figure 7.17: The split flow model for sequence-control analysis

on empty per-path queues while attempting the pair-wise comparisons. Table 7.1 summarizes the findings of [64].

### 7.4.1 Simulation Results

We now present a few of the simulation results from [64], which show how delay estimates and cell loss affect sequencing in multi-path switches. We studied the behavior of a single flow in isolation and modeled the cross-traffic by varying the delay and loss ratios. The system as seen by a single flow is represented by two stages of queues, as shown in Fig. 7.17. The offered load is distributed by the ingress element in a controlled fashion to the multiple paths, and the cells belonging to subflow $i$ are enqueued into the $i$th core element queue. Some of the cells are lost in the core switch, and the rest undergo a controlled amount of delay before they are sent to the egress element, where they are enqueued into the corresponding re-assembly queue. The re-assembly process is scheduler driven, i.e., the output link scheduler polls the re-assembly queues at a constant allocated rate $\mu$, and on every visit, a re-assembler module examines the cell headers at the head of the queues. Either one of the cells is served, or the turn is skipped if the expected cell in sequence is deemed not to be present in any of the queues.

All the rates were normalized to the output scheduling rate, i.e., $\mu = 1$. The arrival

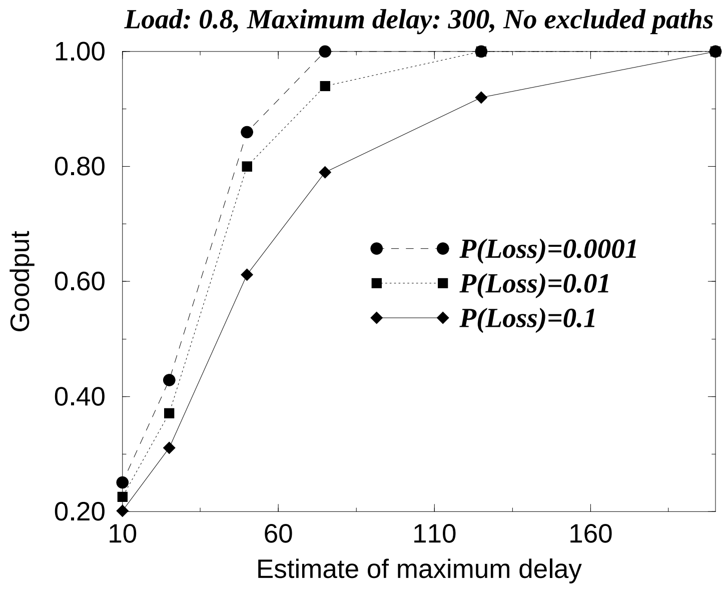


Figure 7.18: Performance of the rank-based protocol

process was chosen to be Bernoulli with a peak rate of 1 and an average rate of 0.8, and were dispatched to the paths using a uniform distribution. The core element losses were modeled using another Bernoulli process with mean $p_L$. The delay distribution in the queues was chosen to be uniform within $[1, 300]$ timeslots. The value of $p_L$ and the delay estimate $d_{\text{est}}$ form the variables of the experiment. The fixed cell overhead was chosen to be 8 bits, thus limiting the total order range to 256. In the case of SCIMA, we set $s = 1$, $F = 4$, and $S_f$ to twice the delay estimate. For the rank-based protocol, the size of each rank component was set to 2 bits. Admittedly, the arrival process is benign, yet it allows us to see the differences in the behaviors of the three protocols, with varying $d_{\text{est}}$ and $p_L$. The *goodput* is measured by counting the fraction of the arrived cells that leave the egress in proper order.

Fig. 7.18 shows the performance of the rank-based protocol as it varies with different underestimations of the maximum delay, for three values of $p_L$. At higher values of estimated delay, the goodput approaches unity, with most of the lost throughput due to cell discards on late arrivals. When the delay estimates become grossly lower than the actual maximum delay, the goodput falls drastically due to a high number of misordered cells. For the same value of $d_{\text{est}}$, the

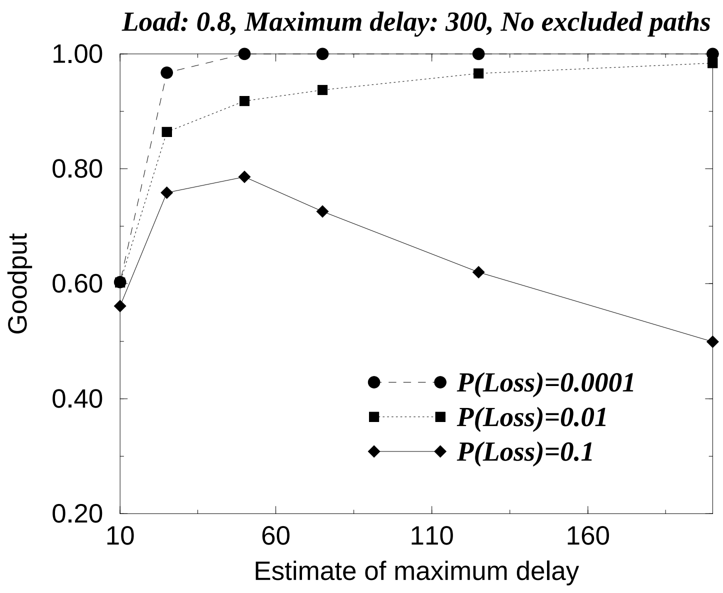


Figure 7.19: Performance of SCIMA

goodput improves with decreasing loss ratio, which is to be expected.

Fig. 7.19 shows the dependence of SCIMA on $d_{\text{est}}$. For low loss levels, the goodput approaches unity even for grossly underestimated delays. However, as losses increase, the goodput goes down because of a large number of cells being flushed in the frame recovery mode. In fact, when losses are high, the goodput *worsens* with increasing delay estimate because the frame sizes which are flushed become higher. Essentially, this tells us that SCIMA is less dependent on accurate delay estimates, but is very sensitive to core element losses.

Finally, as expected, we also found (not shown here) that, for a total-order scheme, there is no appreciable difference with increasing loss ratios. However, the goodput worsens progressively with higher underestimations.

## 7.5 Summary and Discussion

We introduced a parallel packet switch (PPS) architecture in this chapter, as a multi-path BCS, which allows to pool the bandwidth resources of several memory elements, and balance the offered load among them. We noted that, in addition to component reuse, a PPS becomes attractive if the first and third stages can be implemented as memoryless or small-memory elements, without losing performance, thereby allowing to support interface rates in excess of the fastest memory in the system.

We analyzed a flow-based PPS, which statically assigns flows to one of the available paths, based on Clos fitting of the offered-load values. Such a model is used by most clustered routers available today. We established (Theorem 16) that the speedup required depends on how the per-flow loads vary with each other, and may be prohibitively high for highly unbalanced traffic. In addition, there seems to be no possibility of deploying memoryless or small-memory first and third stages. The biggest advantage of a flow-based PPS is that there is no need for sequence control. While the static dispatch method suffices to provide bandwidth and delay guarantees, throughput optimization is inherently impossible without the prior knowledge of the offered load. The speedup may be reduced (Theorem 17) by splitting the load of a few flows, and reassigning the allocations of a few on the addition of a new flow to the system. However, this necessitates sequence control for the set of split flows.

In order to address the throughput limitations of a flow-based PPS, and to possibly realize the memory-bandwidth benefit of this model, we introduced a cell-based PPS, which balances the load of each flow across the available paths. We showed (Theorem 18) that assembling per-flow envelopes and striping them equally across all the paths is sufficient for relative stability with an ideal switch, without speedup. Most beneficially, striping requires no explicit sequence control despite the fact that each flow is split. Furthermore, we may employ a small-memory demultiplexor and a memoryless multiplexor, without losing performance. Striping also suffices for bandwidth guarantees, though at a cost of larger buffers in the first stage. The only disadvantage of this method is the possibility of infinite delay.

We proposed and analyzed an equal dispatch method, which equalizes the load of each flow across all the paths, on a cell-by-cell basis. While this resolves the latency issue of striping, it requires per-flow sequence control. Furthermore, to preserve optimal throughput, the sequencer needs to ensure that it does not suffer from redundant wait, which we were not able to conclusively establish for known open-loop sequence control schemes. Assuming an ideal sequencer,

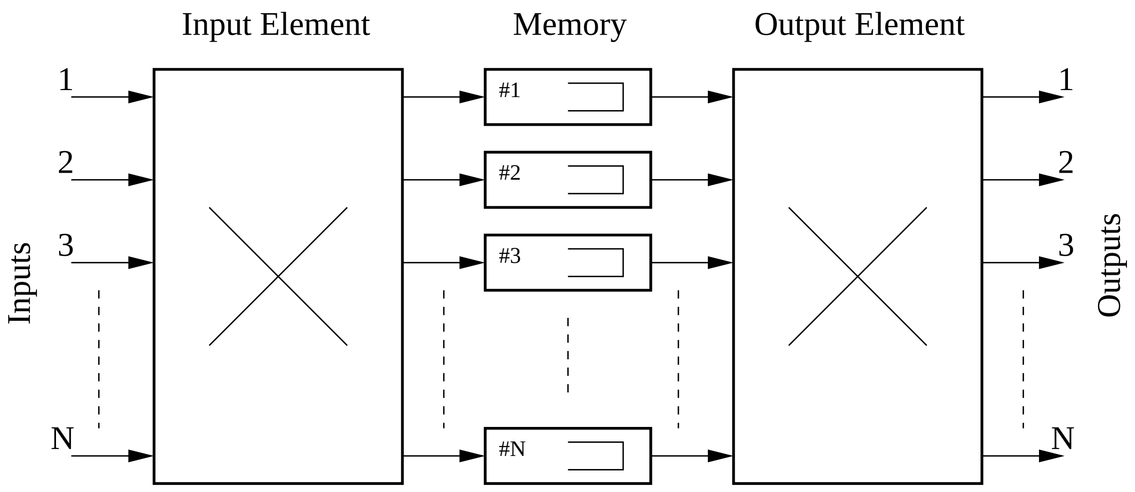


Figure 7.20: A load-balanced Birkhoff-Von Neumann switch

we proved (Theorem 19) that equal dispatch ensures relative stability with an ideal switch, without speedup, while employing small-memory demultiplexors. Further, the same performance holds (Theorem 20) with small-memory multiplexors, with the application of per-flow backpressure from the third to the second stage. Finally, we showed that the first stage may be made memoryless by employing a modest speedup (Theorem 21).

Clearly, a flow-based PPS with static dispatch provides the best option for bandwidth and delay guarantees, while a cell-based PPS with striping is best suited for optimal throughput, both methods avoiding the need for sequence control. An efficient way to provide both in the same switch is a subject that may benefit from further research.

### 7.5.1 Alternative Multi-path Architectures

A rather interesting multi-path architecture, called the load-balanced Birkhoff Von-Neumann (BVN) switch, was introduced by Chang et al. in [9]. It happens to fall into the category of multi-path BCS, as shown in Fig. 7.20. A $(N, [SMS], 1, N, 1)$ switch is composed of a single $N \times N$ space element in both the first and third stages. The second stage is composed of $N$ memory elements of dimensions $1 \times 1$. A cell arriving at input $i$ and destined to $j$ is dispatched, immediately on arrival, to one of the memory elements $k$, depending upon a perfect matching in the first-stage element in that slot. Subsequently, each cell contends for its output in its chosen memory element, which is served by the matching in the third stage.

A remarkable observation in [8] is that if the traffic arriving into the second stage is uniformly distributed, a perfect sequence (e.g., a SPS matching described in Sec. 5.3) would suffice to keep the queues in the second-stage stable. Furthermore, a perfect sequence in the first stage distributes the load, for arbitrary traffic patterns, in such a way that the traffic contending for the output element becomes uniform. In other words, an $O(1)$ perfect sequence takes arbitrary arrivals and homogenizes the input-output load, as seen from the third stage, and the latter, in turn, ensures stability by implementing another $O(1)$ perfect sequence, all without speedup. The main result may be paraphrased as follows.

**Theorem** ( Chang et al., Load-balanced BVN)**.** *A load-balanced BVN switch, with a perfect sequence in the first and third stages, provides asymptotic 100% throughput to all flows destined to a non-oversubscribed output, i.e.,*

$$\text{(BVN, \{Perfect Sequence\})} \overset{T, f_3}{\simeq} \text{(OQ, \{WC\})}, T\text{: SLLN}, s = 1$$

The memory bandwidth in the second stage is exactly twice the interface rate. Note that the objective of load balancing here is not to pool the resources of many smaller elements, as in a PPS, and consequently, the memories run at a speed comparable with a CIOQ switch. However, this multi-path design offers a variety of benefits. It offers the same throughput performance as CIOQ, under maximal matching and a speedup of 2, using an $O(1)$ matching and no speedup. Furthermore, the expected delay approaches that of an OQ switch since the latency of scheduling, in both space elements, is exactly $N$ slots. Using a slightly complex queueing structure, in which cells are arranged in each element on an input-output pair basis, creating a three-dimensional structure, such a switch also provides bandwidth and delay guarantees [57] to conforming flows.

In a following work [10], Chang et al. address the mis-sequencing problem in a load-balanced BVN. Suggestions of delay equalization at the input, and reassembly at the output, are made, much as in a PPS. This, however, requires small buffers at the two edges of the switch, essentially converting it into a 5-stage design. A more complex algorithm, using three-dimensional queues in the second stage, was suggested by Keslassy and McKeown in [59], which prevents mis-sequencing by intelligently selecting cells in an in-sequence fashion, whenever a memory element is served. A similar multi-path architecture was proposed by Prakash and Aziz in [91], to emulate an OQ switch, in which it was proved that $4N$ memory elements in the second stage, and an algorithm that can compute the desired departure time on cell arrival, suffices for OQ emulation.

# Chapter 8

# Conclusions

We addressed the subject of constructing, characterizing, and analyzing high-capacity packet switches using an interconnection of lower-capacity logical elements. Building scalable switches is, by no means, a new topic. However, this work was motivated by a displeasure with the ad-hoc nature in which it has been previously tackled. The modus operandi, so far, has been to propose and analyze different architectures in isolation, often with varying degrees of performance satisfaction, and a few dubious methods of arriving at them. We started with two logical building blocks, namely, memory and space elements, and methodically exhibited the various ways to interconnect them, each associated with well-specified design constraints. We proposed high-performance switching algorithms for each, with an exhaustive analysis using a uniform performance framework. This work is also meant to provide some perspective on the current state of research in switching, and share some of the lessons learned by the author while architecting the commercial Atlanta/$\pi$ switching chipsets.

We started with two questions: what qualifies as the logical architecture of a multi-module packet switch, and what models can be used to characterize optimal performance of a packet switch design? Inspired by similar work in circuit switching, specifically, on the classical three-stage Clos network, we established a taxonomy of three-stage packet switches, called *Buffered Clos Switches* (BCS). To characterize the performance of their algorithms, we introduced a new performance framework, of multi-level *functional equivalence* with an ideal switch. The latter is similar to the levels of blocking used to characterize the performance of circuit switches. In this work, we restricted our performance goals to the ability of a switch to provide per-flow virtual bandwidth trunks for Quality of Service, and its ability to maintain stable queues for throughput. With a taxonomy and performance framework in place, we analyzed the different items in the BCS class, proposing new algorithms, wherever the literature was lacking.

All the contributions contained here are original, except the surveys in Chap. 3, Sec. 7.3.4 and Sec. 7.5.1. Moreover, these are individual contributions by the author, though some parts of Sec. 5.1, Sec. 7.3 and Sec. 7.4 received significant inputs from co-researchers. Also, note that some of the work here has been published before in [17, 18, 62, 63, 64, 67, 22, 66].

## 8.1 Summary of Contributions

The following are the specific contributions of this work.

**Survey of Formal Methods** We presented some of the notable performance results of switching algorithms in the literature, placing them in a new perspective and categorizing them so as to enable meaningful comparisons. We focused on the methods from graph theory, combinatorics and stochastics, which may be used as tools in switching analysis. Special attention was paid to matrix decomposition techniques and stochastic stability, based mainly on Lyapunov functions and fluid limits, which provided us with a simple means of analysis in the rest of the dissertation. The survey was primarily restricted to IQ and CIOQ switches, which form the starting point of our BCS taxonomy.

**Taxonomy and Functional Equivalence** We formally introduced a new taxonomy of three-stage multi-module packet switches, composed of atomic building blocks, called Buffered Clos Switches. We suggested a formal notation to specify a switch design, and enumerated the feasible items that make up the class. We then provided three basic transformations, namely, aggregation, pipelining and parallelization, in order to generate complex items from a basic design. In addition to the taxonomy, a new performance framework was introduced, based on proving functional equivalence with a well-known ideal switch, without restrictions on specific arrival patterns. Existing algorithms from the above survey were then placed into this framework. While the taxonomy and performance framework are novel, note that many of the existing switches already map into items in this class. The rest of the dissertation was a methodical analysis of each item in the taxonomy.

**Combined Input-Output Queueing Switches** We furthered the art in CIOQ switching, already well-researched, by presenting and analyzing matching algorithms for stronger performance than is currently established. We first analytically showed that maximal matching in CIOQ is analogous to circuit fitting in a Clos network, using which we proved that a simple rate-shaping method combined with maximal matching is sufficient for bandwidth and delay guarantees. Next, we established a fundamental result, using fluid limits, proving that a critical matching suffices for 100% throughput to admissible arrivals, without speedup. This has long been conjectured due to a similar result for bounded traffic, but never established so far, the best comparable algorithm being the far more complex maximum weight matching. We then exposed the fact that a simple sub-maximal matching suffices for arrivals that have uniformly distributed destinations, explaining some of the simulations-based observations in the literature.

We next addressed stricter functional equivalence with an OQ switch by proving that a maximal matching behaves well even in the presence of inadmissible traffic, and suffices to isolate instability to oversubscribed ports. Lastly, we proposed a specific instance of maximal matching, called shortest output-queue first, which does the same on an input-output pair basis, thus establishing per-flow stability in the presence of inadmissible traffic. The last two contributions are unprecedented in that the comparable existing results are mainly restricted to admissible traffic. We concluded by putting the QoS and throughput results together in a two-step method called switched fair-airport matching.

**CIOQ with Aggregation and Pipelining** We showed how the transformations of aggregation and pipelining may be beneficially applied to CIOQ switches, in order to generate more scalable designs. Some work already exists in the literature for such switches, but these are mainly in isolation, with purpose-built algorithms and analyses, primarily based on simulations. We showed analytically that the well-established CIOQ results may be extended to these transformations, thereby inheriting its performance.

Specifically, we proved that a shadow-and-decompose method, in switches with aggregation, may be used to exactly emulate a CIOQ switch. For lower complexity, the same matching algorithms as in CIOQ may be applied using a different queueing structure, but at the cost of some throughput performance. Similarly, we showed that shadowing a CIOQ switch, followed by sequential dispatch, suffices in a switch with pipelining. To lower complexity, we applied simple striping and equal dispatch methods on well-known CIOQ matching algorithms, without performance degradation. We also presented a few concurrent dispatch algorithms, for switches with pipelining, specifically a three-dimensional maximal matching, which was shown to be sufficient for relative stability with an ideal switch. The results on aggregation and pipelining were then combined in order to characterize the performance of general memory-space-memory switches. Finally, we briefly presented and analyzed a recursively constructed BCS for further scalability.

**Parallel Packet Switches** We introduced a new parallel packet switch (PPS) architecture, a multi-path BCS, as a means to pool the capacity of small memory elements, and balancing the offered load among them. We classified such designs into flow-based and cell-based PPS. For flow-based PPS, we showed that a static dispatch method suffices for fitting admissible offered loads, across the available paths, though with a high speedup for severely unbalanced traffic. The speedup may be reduced by splitting a few flows among multiple paths, at the cost of implementing sequence control. While we established the applicability of a flow-based PPS for QoS guarantees, it relies on a prior knowledge of offered loads, in order to optimize throughput, possibly an impractical assumption.

A cell-based PPS was presented as an alternative, so as to provide better throughput performance. Similar to a switch with pipelining, we proposed envelope striping for load balancing, and proved that this suffices for relative stability with an ideal switch, without the need for sequence control. Furthermore, this method allows us to employ zero or very small buffers in the high-speed portions of the switch. Next, we presented equal and fractional dispatch methods, and proved the same throughput result. The latter exhibits better latency than striping, but at the cost of implementing sequence control. We concluded by exposing the problems associated with open-loop sequence control in such switches, an issue that might eventually hamper their deployment.

## 8.2 Topics for Further Research

In a journey through phantom buckets, imaginary queues and three-dimensional Sudoku-like matchings, we encountered a few ancillary questions, which were not addressed to satisfaction, as they landed outside the immediate scope of this work. Some of these provide interesting challenges for future research. An exciting sequel could be a study of more sophisticated and scalable designs based on a similar treatment on other multi-stage circuit-switching interconnection networks. For example, a buffered Cantor network seems like an interesting candidate for research. Similarly, we brushed upon the possibility of recursively constructing multi-stage architectures. We introduced a recursive G-MSM, but did not pursue it in detail. Such a switch might be able to retain all the desirable properties of a G-MSM, while employing even smaller elements. Another candidate for recursion is the PPS. We question whether it is possible to build a switch with arbitrarily slow memories by recursively replacing each core element of a PPS with a smaller PPS. These designs need further analysis before they become serious candidates for implementation.

In the context of CIOQ matching algorithms, we introduced the *Batch-Continuous hypothesis*, which, if proven, may end up as one of the most useful results to aid in analysis. We conjectured that a matching algorithm that ensures bounded delay to bounded arrivals will also ensure stable queues for unbounded arrivals. If this is shown to be true, we may be able to avoid a cumbersome probabilistic analysis for each new matching policy under consideration. Instead, the combinatorial properties of the policy may be used to directly infer its performance under stochastic traffic. Another topic for future research is the exploration of statistical multiplexing results for popular matching algorithms. We also touched on the employment of backpressure from the output to the input elements, in order to approximate the SOQF policy, for per-flow stability, in the presence of inadmissible traffic. While we provided the intuition behind such a choice, the throughput properties for such backpressure is not known.

For switches with pipelining, we noted that several well-known matching policies do not lend themselves to concurrent dispatch. In fact, the only ones that we came up with were concurrent versions of the orderly EREW maximal and SPS matchings. Therefore, we indicated a preference for balanced matchings such as striping and equal dispatch, each associated with some penalties. It might be beneficial to explore the application of more sophisticated parallel algorithms, to gain better performance, with low speedup, using concurrent matching. We also noted the possibility of using the low-complexity dispatch methods, of a switch with spatial pipelining, towards a temporal pipeline in a CIOQ. We may thereby decrease its matching complexity, without the penalties associated with envelope-based schemes in the literature. As it may allow the practical implementation of complex matchings, this is a promising area to explore.

The interaction of a practical sequence-control scheme with the load balancing policy, in a multi-path switch, has not been addressed satisfactorily. Specifically, for a PPS with a CIOQ central-stage, an architecture that truly exposes all the benefits of parallelization, the differential delay between two paths may be inestimable. This is one of the main reasons that has halted the commercial deployment of PPS, except for the arguably low-performance clustered model. Another issue is the combined provision of QoS and throughput in a PPS, without the need for large buffers in the high-speed portions. For the policies we considered, the demultiplexors and multiplexors of such a PPS would require $O(F)$ amount of buffering, which may be impractical to implement on-chip, not to mention per-flow sequencing. Finally, a more thorough analysis is required for a PPS with CIOQ central stages. We mainly concentrated on memory elements in the second stage, and made subjective arguments for the above combination. In practice, however, it is indeed such a combination that is attractive for deployment.

The performance of the resident algorithms for multicast traffic, for each item in the taxonomy, is another rich topic for future research.

## New Directions

There are some interesting open challenges in switching, which the research community has recently begun to address, all related to the quality of stability and finiteness of memory. Most of the throughput analysis in switching, including our own in this work, is based on establishing stability in an infinite-memory system. The rationale is that the tail of the queue-length distributions determines the losses we may expect in a finite-memory equivalent. This rationale, however, opens up several challenging questions.

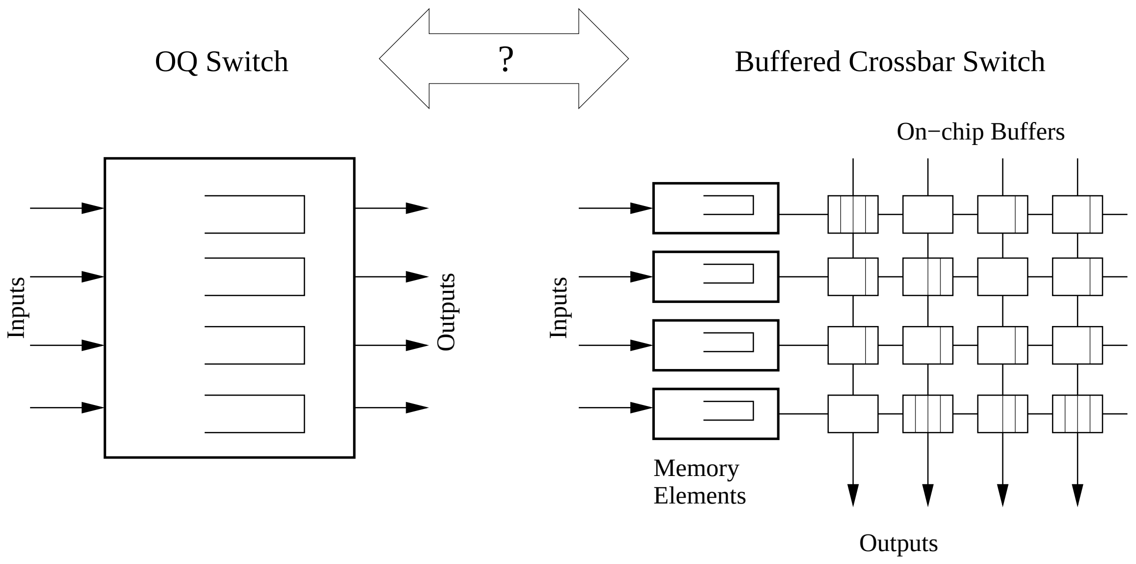


Figure 8.1: Is a buffered crossbar functionally equivalent to an OQ switch?

The first is an accurate characterization of the distributions, for meaningful arrival processes. Two algorithms that provide stability to infinite-sized queues may result in widely varying distributions. For example, a maximum weight matching and a critical matching show preference for large queues, while a maximal matching is insensitive to the size of the backlogs. The initial work by Leonardi et al. [71] and, Shah and Kopikare [98] need to be extended to other policies and more general arrivals. A second issue is that a finite memory, with an active buffer management scheme, might change the outcome of matching policies with respect to an infinite-queue system. In this case, even a characterization of queue-length distributions is rendered useless. The problem then is to understand the interaction of buffer management with scheduling. This is a complicated problem even for link schedulers, which we addressed in [66], let alone for two-dimensional switch schedulers. Some initial work on this has been done by Sarkar in [96]. The third issue is the very nature of stability. Most of such results are based on *long-term* averages, and are rendered utterly meaningless for real traffic. A short-term calculus needs to be devised that is best-suited for real traffic. In its absence, several researchers use simulation studies to determine loss ratios in a finite buffer for specific arrival patterns. However, it is often difficult to ascertain whether an observed behavior is an outcome of the policy under consideration, or due to the nature of the chosen patterns.

Disillusioned by the complexity of high-performance matching algorithms in CIOQ, researchers have recently renewed their focus on output queueing. Specifically, the new focus is on buffered crossbars, which allow distributed scheduling at each output, much as an OQ switch. The basis of the current exploration is shown in Fig. 8.1. We know that a $N \times N$ OQ switch may be implemented using $N^2$ large memories. If those memories are placed at the crosspoints of a space element, with restricted sizes and backpressure, we obtain a buffered crossbar[1]. Now, there are only $N$ large memories, which, however, cannot be work-conserving. The QoS properties of such an architecture has been adequately addressed by Stephens and Zhang in [103], by Chiussi and Francini in [13, 35, 14], and by us in [16]. The current emphasis is on throughput in comparison to an ideal switch.

[1]The memory elements of the Atlanta/$\pi$ chipsets were in fact buffered crossbars. It was not fashionable then to call them as such.

# Bibliography

[1] T. Anderson, S. Owicki, J. Saxe, C. Thacker, “High speed Switch Scheduling for Local Area Networks,” *ACM Trans. Computer Systems*, Nov 1993.

[2] J. Bennett and C. Partridge, “Packet reordering is not pathological network behavior,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 789–798, 1999.

[3] J. C. R. Bennett and H. Zhang, “WF2Q: Worst-Case Fair Weighted Fair Queueing,” in *Proc. IEEE Infocom ‘96*, pp. 120–128, San Francisco, Mar 1996.

[4] J. C. R. Bennett and H. Zhang, “Hierarchical Packet Fair Queueing Algorithms,” in *Proc. ACM Sigcomm ‘96*, pp. 143–156, Stanford, CA, Aug 1996.

[5] J. Cao, W. S. Cleveland, D. Lin, D. X. Sun, “On the Non-stationarity of Internet Traffic,” in *Proc. ACM Sigmetrics ‘01*, pp. 102–112, Cambridge, MA, Jun 2001.

[6] Z. Cao, Z. Wang, E. Zegura, “Performance of Hashing-based Schemes for Internet Load Balancing,” in *Proc. IEEE Infocom ‘00*, Tel Aviv, pp. 332–341, Mar 2000.

[7] C. S. Chang, J. W. Chen, H. Y. Huang, “On Service Guarantees for Input-Buffered crossbar switches: A capacity decomposition approach by Birkhoff and Von-Neumann,” in *Proc. IEEE Intl. Workshop on QoS*, pp. 79–86, London 1999.

[8] C. S. Chang, J. W. Chen, H. Y. Huang, “Birkhoff-Von Neumann Input-Buffered Crossbar Switches,” in *Proc. IEEE Infocom ‘00*, Tel Aviv, pp. 1614–1623, Mar 2000.

[9] C. S. Chang, D. S. Lee, Y. S. Jou, “Load Balanced Birkhoff-Von Neumann Switches, Part I: One-stage Buffering,” *Computer Communications*, vol. 25, pp. 611–622, 2002.

[10] C. S. Chang, D. S. Lee, C. M. Lien, “Load Balanced Birkhoff-Von Neumann Switches, Part II: Multi-stage Buffering,” *Computer Communications*, vol. 25, pp. 623–634, 2002.

[11] H. J. Chao, “Saturn: A Terabit Packet Switch using Dual Round-Robin,” *IEEE Communications Magazine*, vol. 38, pp. 78–84, Dec 2000.

[12] F. M. Chiussi, “Design, Performance and Implementation of a Three-Stage Banyan-based Architecture with Input and Output buffers for Large Fast Packet Switches,” *Ph.D. Dissertation*, Stanford University, Jul 1993.

[13] F. M. Chiussi and A. Francini, “Scalable Electronic Packet Switches,” *IEEE Journal on Sel. Areas of Comm.*, vol. 21, no. 4, pp. 486–500, May 2003.

[14] F. M. Chiussi and A. Francini, “A Distributed Scheduling Architecture for Scalable Packet Switches,” *IEEE Journal on Sel. Areas of Comm.*, vol. 18, no. 12, pp. 2665–2683, Dec 2000.

[15] F. M. Chiussi, A. Francini, G. Galante, E. Leonardi, “A Novel Highly-Scalable Matching Policy for Input-Queued Switches with Multiclass Traffic,” in *Proc. IEEE Globecom ‘02*, Taipei, Taiwan, Nov 2002.

[16] F. M. Chiussi, A. Francini, D. A. Khotimsky, S. Krishnan, “Feedback Control in a Distributed Scheduling Architecture,” in *Proc. IEEE Globecom 2000*, San Francisco, Oct 2000.

[17] F. M. Chiussi, D. A. Khotimsky, S. Krishnan, “Generalized Inverse Multiplexing of Switched ATM Connections,” in *Proc. IEEE Globecom ‘98*, vol. 5, pp. 3134–3140, Sydney, Nov 1998.

[18] F. M. Chiussi, D. A. Khotimsky, S. Krishnan, “Advanced Frame Recovery in Switched Connections Inverse Multiplexing for ATM,” in *Proc. Intl. Conf. on ATM*, Colmar, France, Jun 1999.

[19] F. M. Chiussi, J. G. Kneuer, V. P. Kumar, “Low-cost Scalable switching solutions for Broadband Networking: the ATLANTA architecture and chipset,” *IEEE Communications Magazine*, vol. 35, no. 12, pp. 44–53, Dec 1997.

[20] F. M. Chiussi, Y. Xia, V. P. Kumar, “Backpressure in Shared-Memory-based ATM Switches under Multiplexed Bursty Sources,” in *Proc. Infocom ‘96*, pp. 830–843, San Francisco, Mar 1996.

[21] F. M. Chiussi, *et al.*, “A Chipset for Scalable QoS-Preserving Protocol-Independent Packet Switch Fabrics,” in *Proc. Intl. Solid-State Circuits Conf. 2001*, San Francisco, Feb 2001.

[22] F. M. Chiussi, *et al.*, “A Family of ASIC devices for Next Generation Distributed Packet Switches with QoS support for IP and ATM,” in *Proc. Hot Interconnects 9*, pg. 145–149, Stanford, CA, Aug 2001.

[23] A. K. Choudhury and E. L. Hahne, “Dynamic Queue Length Thresholds in a Shared Memory ATM Switch,” in *Proc. IEEE Infocom ‘96*, pp. 679–687, San Francisco, Mar 1996.

[24] S-T. Chuang, A. Goel, N. McKeown, B. Prabhakar, “Matching Output Queueing with a Combined Input Output Queued Switch,” *IEEE J. Selected Areas of Communications*, vol. 17, no. 6, pp. 1030–1039, Dec 1999.

[25] C. Clos, “A Study of Non-blocking Switching Networks,” *Bell System Technical Journal*, vol. 32, Mar 1953.

[26] R. Cole, “Parallel Merge Sort,” *SIAM Journal of Computing*, vol. 17, no. 4, pp. 770–785, 1988.

[27] T. H. Corman, C. E. Leiserson, R. L. Rivest, “Introduction to Algorithms,” MIT Press, 1990.

[28] R. L. Cruz, “A Calculus for Network Delay, Part I: Network Elements in Isolation,” *IEEE Trans. Information Theory*, vol. 37, no. 1, pp. 114–131, Jan 1991.

[29] J. G. Dai and B. Prabhakar, “The Throughput of Data Switches with and without Speedup,” in *Proc. IEEE Infocom 2000*, pp. 556–564, Tel Aviv, Mar 2000.

[30] A. Demers, S. Keshav, S. Shenker, “Analysis and Simulation of a Fair Queueing Algorithm,” in *Proc. ACM Sigcomm ‘89*, pp. 1–12, Austin, TX, Sep 1989.

[31] M. Devault, J. Cochennec, M. Servel, “The Prelude ATD Experiment: Assessments and Future Prospects,” *IEEE J. Selected Areas in Comm.*, vol. 6, no. 9, pp. 1528–1537, Dec 1988.

[32] M. Fayyazi, D. Kaeli, W. Meleis, “Parallel Maximum Weight Bipartite Matching Algorithms for Scheduling in Input-Queued Switches,” in *Proc. IEEE Intl. Parallel and Distributed Proc. Symposium ‘04*, pp. 26–30, Santa Fe, New Mexico, Apr 2004.

[33] S. Floyd and V. Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” *IEEE Transactions on Networking*, pp. 397–413, Aug 1993.

[34] G. J. Foschini and B. Gopinath, “Sharing Memory Optimally,” *IEEE Transactions on Communications*, vol. 3, pp. 352–360, Mar 1983.

[35] A. Francini and F. M. Chiussi, “Providing QoS Guarantees to Unicast and Multicast Flows in Multi-stage Packet Switches,” *IEEE Journal on Sel. Areas of Comm.*, vol. 20, no. 8, pp. 1589–1601, Oct 2002.

[36] Y. Ganjali, A. Keshavarzian, D. Shah, “Input-Queued Switches: Cell Switching vs. Packet Switching,” in *Proc. IEEE Infocom ‘03*, San Francisco, Apr 2003.

[37] L. Georgiadis, I. Cidon, R. Geurin, A. Khamisy, “Optimal Buffer Sharing,” *IEEE J. Selected Areas of Communications*, vol. 13, pp. 1229–1240, Sep 1995.

[38] P. Giaccone, E. Leonardi, B. Prabhakar, D. Shah, “Delay Bounds for Combined Input-Output Switches with Low Speedup,” *Performance Evaluation*, vol. 55, no. 1, pp. 113–128, Jan 2004.

[39] S. J. Golestani, “A Self-Clocked Fair Queueing scheme for Broadband Applications,” in *Proc. IEEE Infocom ‘94*, pp. 636–646, Toronto, Jun 1994.

[40] P. Goyal and H. M. Vin, “Fair Airport Scheduling Algorithms,” in *Proc. NOSSDAV ‘97*, pp. 272-283, St. Louis, Missouri, May 1997.

[41] M. Guo and R. Chang, “Multicast ATM Switches: Survey and Performance Evaluation,” *Computer Communication Review*, vol. 28, no. 2, pp. 98–131, Apr 1998.

[42] S. Gupta and A. Aziz, “Multicast Scheduling for Switches with Multiple Input-queues,” in *Proc. Hot Interconnects 10*, Stanford, CA, Aug 2002.

[43] M. Hall, Jr., “Combinatorial Theory,” John Wiley and Sons, *2nd Edition*, New York, 1986.

[44] J. E. Hopcroft and R. M. Karp, “An $n^{5/2}$ algorithm for maximum matching in bipartite graphs,” *SIAM Journal of Computation*, vol. 2, no. 4, pp. 225–231, 1973.

[45] J. Y. Hui, “Switching and Traffic Theory for Integrated Broadband Networks,” Kluwer Academic Press, 1990.

[46] A. Hung, G. Kesidis, N. McKeown, “ATM Input-Buffered switches with Guaranteed Rate property,” in *Proc. IEEE ISCC*, pp. 331–335, Athens, Greece, 1998.

[47] M. I. Irland, “Buffer Management in a Packet Switch,” *IEEE Trans. Communications*, vol. 26, pp. 328–337, Mar 1978.

[48] A. Israeli and Y. Shiloach, “An Improved Algorithm for Maximal Matching,” *Information Processing Letters*, vol. 22, no. 2, pp. 57–60, Jan 1986.

[49] S. Iyer, A. Awadallah, N. McKeown, “Analysis of a Packet Switch with Memories running slower than line rate,” in *Proc. IEEE Infocom 2000*, pp. 529–537, Tel Aviv, Mar 2000.

[50] S. Iyer and N. McKeown, “Making Parallel Packet Switches practical,” in *Proc. IEEE Infocom ‘01*, pp. 1680–1687, Anchorage, Alaska, Apr 2001.

[51] S. Iyer and N. McKeown, “Analysis of the Parallel Packet Switch Architecture,” *IEEE/ACM Transactions on Networking*, vol. 11, no. 2, pp. 314–324, Apr 2003.

[52] S. Iyer and N. McKeown, “Maximum Size Matching and Input Queued Switches,” in *Proc. of the 40th Allerton Conf. on Communications, Control and Computing*, Urbana, Illinois, Oct 2002.

[53] J. Y. Jo, Y. Kim, H. J. Chao, F. L. Merat, “Internet Traffic Load-balancing using Dynamic Hashing with Flow Volume,” in *Proc. SPIE ITCom 2002*, Boston, MA, Jul 2002.

[54] K. Kar, T. V. Lakshman, D. Stiliadis, L. Tassiulas, “Reduced Complexity Input-Buffered Switches,” in *Proc. Hot Interconnects 8*, Palo Alto, Aug 2000.

[55] M. Karol, M. Hluchyj, S. Morgan, “Input versus Output queueing on a Space Division Switch,” *IEEE Trans. Communications*, vol. 35, no. 12, pp. 1347–1356, Dec 1987.

[56] M. Katevenis, S. Sidoropoulus, C. Courcoubetis, “Weighted Round Robin Cell Multiplexing in a General Purpose ATM Switch chip,” *IEEE Journal on Sel. Areas in Comm.*, vol. 9, pp. 1265–1279, Oct 1991.

[57] I. Keslassy, “The Load-Balanced Router,” *Ph.D. Dissertation*, Stanford University, June 2004.

[58] I. Keslassy, M. Kodialam, T. V. Lakshman, D. Stiliadis, “On Guaranteed Smooth Scheduling for Input-Queued Switches,” in *Proc. IEEE Infocom ‘03*, San Francisco, Apr 2003.

[59] I. Keslassy and N. McKeown, “Maintaining Packet Order in Two-stage Switches,” in *Proc. IEEE Infocom ‘02*, New York, Jun 2002.

[60] I. Keslassy, R. Zhang-Shen, N. McKeown, “Maximum Size Matching is Unstable for any Packet Switch,” *IEEE Communication Letters*, vol. 10, no. 7, 496–498, Oct 2003.

[61] D. Khotimsky, “A Packet Re-sequencing Protocol for Fault-Tolerant Multi-path Transmission with Non-uniform Traffic Splitting,” in *Proc. IEEE Globecom ‘99*, Rio de Janeiro, Brazil, Dec 1999.

[62] D. A. Khotimsky and S. Krishnan, “Towards the Recognition of Parallel Packet Switches,” *Gigabit Networking Workshop at IEEE Infocom ‘01*, Anchorage, Alaska, Apr 2001.

[63] D. A. Khotimsky and S. Krishnan, “Stability Analysis of a Parallel Packet Switch with Bufferless input Demultiplexors,” in *Proc. ICC ‘01*, pp. 100–111, Helsinki, Jun 2001.

[64] D. A. Khotimsky and S. Krishnan, "Evaluation of Open-loop Sequence Control schemes for Multi-path Switches," in *Proc. Intl. Conf. Communications ICC '02*, New York, Apr 2002.

[65] P. Krishna, N. S. Patel, A. Charny, R. Simcoe, "On the speedup required for work-conserving crossbar switches," in *Proc. 6th Intl. Workshop on QoS*, pp. 225–234, Napa, CA, May 1998.

[66] S. Krishnan, A. K. Choudhury, F. M. Chiussi, "Dynamic Partitioning: A Mechanism for Shared-Memory Management," in *Proc. IEEE Infocom '99*, pp. 144–152, New York, NY, Mar 1999.

[67] S. Krishnan and H. G. Schulzrinne, "On Buffered Clos Switches," *Technical Report*, CUCS-023-02, Columbia University, Nov 2002.

[68] P. R. Kumar and S. P. Meyn, "Stability of queueing networks and scheduling policies," *IEEE Transactions on Automatic Control*, vol. 40, no. 2, Feb 1995.

[69] V. P. Kumar, T. V. Lakshman, D. Stiliadis, "Beyond Best-effort: Router Architectures for the Differentiated Services of Tomorrow's Internet," *IEEE Communications Magazine*, vol. 36, no. 5, May 1998.

[70] E. Leonardi, M. Mellia, F. Neri, M. A. Marsan, "On the Stability of Input-Queued switches with Speed-up," *IEEE/ACM Trans. Networking*, vol. 19, no. 1, pp. 104–118, Feb 2001.

[71] E. Leonardi, M. Mellia, F. Neri, M. A. Marsan, "Bounds on Average Delays and Queue Size Averages and Variances in Input-Queued Cell-based Switches," in *Proc. IEEE Infocom '01*, vol. 3, pp. 1095–1103, Anchorage, AK, Apr 2001.

[72] S. Li and N. Ansari, "Input-queued Switching with QoS Guarantees," in *Proc. IEEE Infocom '99*, pp. 1152–1159, New York, NY, Mar 1999.

[73] X. Li and M. Hamdi, "On Scheduling Optical Packet Switches with Reconfiguration Delay," *IEEE Journal on Sel. Areas of Comm.*, vol. 21, no. 7, pp. 1156–1164, Sep 2003.

[74] X. Li, Z. Zhou, M. Hamdi, “Space-Memory-Memory Architecture for Clos-Network Packet Switches,” in *Proc. IEEE ICC ‘05*, Seoul, S. Korea, May 2005.

[75] Z. Liu and R. Righter, “Scheduling Multicast Input-queued Switches,” *Journal of Scheduling*, John Wiley and Sons, vol, 2, no. 3, pp. 99–114, May 1999.

[76] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, F. Neri, “On the Throughput of Input-queued Cell-based Switches with Multicast Traffic,” in *Proc. IEEE Infocom ‘01*, pp. 1664–1672, Anchorage, Apr 2001.

[77] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, F. Neri, “Packet-Mode Scheduling in Input-Queued Cell-Based Switches,” *IEEE/ACM Trans. Networking*, vol. 10, no. 5, pp. 666–678, Oct 2002.

[78] N. McKeown, “The iSLIP Scheduling Algorithm for Input-Queued Switch,” *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, Apr 1999.

[79] N. McKeown, A. Mekkittikul, “A Practical Scheduling Algorithm to achieve 100% throughput in Input-Queued Switches,” in *Proc. IEEE Infocom ‘98*, pp. 792–799, San Francisco, CA, Apr 1998.

[80] N. McKeown, A. Mekkittikul, V. Anantharam, J. Walrand, “Achieving 100% Throughput in an Input-Queued Switch,” *IEEE Trans. Communications*, vol. 47, no. 8, pp. 1260–1272, Aug 1999.

[81] S. Melen and J. Tuner, “Non-blocking Multi-rate Networks,” *SIAM Journal of Computing*, vol. 18, no. 2, pp. 301–313, Apr 1989.

[82] S. P. Meyn and R. Tweedie, “Markov Chains and Stochastic Stability,” Springer, London, 1993.

[83] M. Mitzenmacher and E. Upfal, “Probability and Computing: Randomized Algorithms and Probabilistic Analysis,” Cambridge University Press, Jan 2005.

[84] K. Mulmuley, U. Vazirani, V. Vazirani, “Matching is as easy as matrix inversion,” *Combinatorica*, vol. 7, no. 1, pp. 105–113, 1987.

[85] E. Oki, Z. Jing, R. Rojas-Cessa, H. J. Chao, “Concurrent Round-Robin-based Dispatching Schemes for Clos-Network Switches,” *IEEE/ATM Trans. Networking*, vol. 10, no. 6, pp. 830–844, Dec 2002.

[86] C. H. Papadimitrious and K. Steiglitz, “Combinatorial Optimization: Algorithms and Complexity,” Prentice-Hall, 1982.

[87] A. K. Parekh and R. G. Gallagher, “A Generalized Processor Sharing approach to Flow Control in Integrated Service Networks– The Single Node Case,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344–357, Jun 1993.

[88] A. Pattavina, “Switching Theory: Architecture and Performance in Broadband ATM Networks,” John Wiley, W. Sussex, UK, 1998.

[89] V. Paxson and S. Floyd, “Wide-Area Traffic: The failure of Poisson Modeling,” in *Proc. ACM Sigcomm ‘94*, pp. 257–268, London, UK, Aug 1994.

[90] B. Prabhakar, N. McKeown, R. Ahuja, “Multicast Scheduling for Input-Queued Switches,” *IEEE Journal on Sel. Areas of Comm.*, vol. 15, no. 5, pp. 855–866, 1997.

[91] A. Prakash, S. Sharif, A. Aziz, “An $O(\log^2 N)$ parallel algorithm for output queueing,” in *Proc. IEEE Infocom ‘02*, New York, Jun 2002.

[92] K. Pun and M. Hamdi, “Dispatching Schemes for Clos-Network Switches,” *Computer Networks*, vol. 44, no. 5, pp. 667–679, Apr 2004.

[93] R. Rojas-Cessa, E. Oki, H. J. Chao, “Maximum Weight Matching Dispatching Scheme in Buffered Clos-Network Packet Switches,” in *Proc. IEEE ICC ‘04*, Paris, France, Jun 2004.

[94] E. Rosen, A. Viswanathan, R. Callon, “Multiprotocol Label Switching Architecture,” *Internet Engg. Task Force* RFC 3031, Jan 2001.

[95] K. W. Ross, “Multi-service Loss Models for Broadband Telecommunication Networks,” Springer-Verlag, London, 1995.

[96] S. Sarkar, "Optimum Scheduling and Memory Management in Input-queued Switches with Finite Buffer Space," *IEEE Trans. Information Theory*, vol. 50, no. 12, pp. 3197–3220, Dec 2004.

[97] D. Shah, "Maximal Matching Scheduling is good enough," in Proc. IEEE Globecom '03, San Francisco, CA, Dec 2003.

[98] D. Shah and M. Kopikare, "Delay Bounds for Approximate Maximum Weight Matching Algorithms for Input Queued Switches," in *Proc. IEEE Infocom '02*, vol. 21, no. 1, pp. 1024–1031, New York, Jun 2002.

[99] M. Shreedhar and G. Varghese, "Efficient Fair Queueing using Deficit Round Robin," in *Proc. ACM Sigcomm '95*, pp. 231–242, Cambridge, MA, Sep 1995.

[100] A. Smiljanic, "Flexible Bandwidth Allocation in High-capacity Packet Switches," *IEEE/ACM Trans. Networking*, vol. 10, no. 2, pp. 287–293, Apr 2002.

[101] A. Smiljanic, "Performance of Load Balancing Algorithms in Clos Packet Switches," in *Proc. Workshop on High Performance Switching and Routing 2004*, Phoenix, AZ, Apr 2004.

[102] S. Stanley, "Switch-Fabric Chipsets," *Light Reading* survey, Document 47959, Mar 2004.

[103] D. C. Stephens and H. Zhang, "Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture," in *Proc. IEEE Infocom '98*, San Francisco, Mar 1998.

[104] D. Stiliadis and A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," *IEEE/ACM Trans. Networking*, vol. 6, no. 5, pp. 611–624, Oct 1998.

[105] D. Stiliadis and A. Varma, "A General Methodology for Designing Efficient Traffic Scheduling and Shaping Algorithms," in *Proc. IEEE Infocom '97*, pp. 326–335, Kobe, Japan, Apr 1997.

[106] D. Stiliadis and A. Varma, "Design and Analysis of Frame-based Fair Queueing: A new traffic scheduling algorithm for packet switched networks," in *Proc. ACM Sigmetrics '96*, pp. 104–115, Philadelphia, PA, May 1996.

[107] I. Stoica and H. Zhang, "Exact emulation of an Output Queueing switch by a Combined Input Output queueing switch," in *Proc. 6th Intl. Workshop on QoS*, pp. 218–224, Napa, CA, May 1998.

[108] B. Suter, T. V. Lakshman, D. Stiliadis, A. K. Choudhury, "Design Considerations for supporting TCP with Per-flow Queueing," in *Proc. IEEE Infocom '98*, San Francisco, CA, Mar 1998.

[109] Y. Tamir and G. Frazier, "High Performance multi-queue buffers for VLSI communication switches," in *Proc. 15th Annual Symp. on Computer Arch.*, pp. 343–354, Jun 1988.

[110] F. A. Tobagi, "Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks," in *Proc. of IEEE*, vol. 78, no. 1, pp. 133–167, Jan 1990.

[111] B. Towles and W. J. Dally, "Guaranteed Scheduling for Switches with Configuration Overhead," in *Proc. IEEE Infocom '02*, New York, Jun 2002.

[112] J. Turner, "An Optimal Nonblocking Multicast Virtual Circuit Switch," in *Proc. IEEE Infocom '94*, pp. 298–305, Toronto, Canada, Jun 1994.

[113] W. Wang, L. Dong, W. Wolf, "A Distributed Switch Architecture with Dynamic Load-balancing and Parallel Input-Queued Crossbars for Terabit Switch Fabrics," in *Proc. IEEE Infocom '02*, New York, Jun 2002.

[114] T. Weller and B. Hajek, "Scheduling Nonuniform Traffic in a Packet-Switching System with Small Propagation Delay," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 813–823, Dec 1997.

[115] G. Wilfong, B. Mikkelsen, C. Doerr, M. Zirngibl, "WDM Cross-connect Architectures with reduced complexity," *J. Lightwave Technology*, pp. 1732–1741, Oct 1999.

[116] Y. S. Yeh, M. G. Hluchyj, A. S. Acampora, “The Knockout Switch: A simple modular architecture for High-performance Packet Switching,” *IEEE J. Selected Areas in Comm.*, vol. 5, no. 5, pp. 1273–1283, Oct 1987.

[117] L. Zhang, “Virtual Clock: A New Traffic Control Algorithm for Packet Switching,” *ACM Transactions on Computer Systems*, vol. 9, no. 2, pp. 101–124, May 1991.