SCHEDULING IN NETWORKS WITH LIMITED BUFFERS

by

Mahmoud Elhaddad

M.S. in Computer Science, North Carolina State University, 2001

Submitted to the Graduate Faculty of

Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2010

UNIVERSITY OF PITTSBURGH

FACULTY OF ARTS AND SCIENCES

This dissertation was presented

by

Mahmoud Elhaddad

It was defended on

May 6, 2010

and approved by

R. Melhem, PhD, Professor

D. Mossé, PhD, Professor

K. Pruhs, PhD, Professor

P. Krishnamurthy, PhD, Associate Professor

Dissertation Director: R. Melhem, PhD, Professor

SCHEDULING IN NETWORKS WITH LIMITED BUFFERS

Mahmoud Elhaddad, PhD

University of Pittsburgh, 2010

In networks with limited buffer capacity, packet loss can occur at a link even when the average packet arrival rate is low compared to the links speed. To offer strong loss-rate guarantees, ISPs may need to adopt stringent routing constraints to limit the load at the network links and the routing path length. However, to simultaneously maximize revenue, ISPs should be interested in scheduling algorithms that lead to the least stringent routing constraints. This work attempts to address the ISPs needs as follows. First, by proposing an algorithm that performs well (in terms of routing constraints) on networks of output queued (OQ) routers (that is, ideal routers), and second, by bounding the extra switch fabric speed and buffer capacity required for the emulation of these algorithms in combined input-output queued (CIOQ) routers.

The first part of the thesis studies the problem of minimizing the maximum session loss rate in networks of OQ routers. It introduces the Rolling Priority algorithm, a local online scheduling algorithm that offers superior loss guarantees compared to FCFS/Drop Tail and FCFS/Random Drop. Rolling Priority has the following properties: (1) it does not favor any sessions over others at any link, (2) ensures a proportion of packets from each session are subject to a negligibly small loss probability at every link along the sessions path, and (3) maximizes the proportion of packets subject to negligible loss probability.

The second part of the thesis studies the emulation of OQ routers using CIOQ. The OQ routers are equipped with a buffer of capacity B packets at every output. For the family of work-conserving scheduling algorithms, we find that whereas every greedy CIOQ policy is valid for the emulation of every OQ algorithm at speedup B, no CIOQ policy is valid at

speedup $s < \sqrt[3]{B-2}$ when preemption is allowed. We also find that CCF, a well-studied CIOQ policy, is not valid at any speedup $s < B$. We then introduce a CIOQ policy CEH, that is valid at speedup $s \geq \sqrt{2(B-1)}$. Under CEH, the buffer occupancy at any input never exceeds $1 + \lfloor \frac{B-1}{s-1} \rfloor$.

TABLE OF CONTENTS

1.0 INTRODUCTION 1

1.1 Minimization of the Maximum Session Loss Rate 2

1.2 Exact Emulation of OQ Routers Using CIOQ 3

1.3 Organization of the Dissertation 5

2.0 PROBLEM DEFINITION AND PRELIMINARIES 6

2.1 Problem Definition 6

2.1.1 Problem Parameters 6

2.1.2 Input Specification 7

2.1.3 The Arrival Process 9

2.1.4 The Performance Metric 9

2.2 Summary of Results 10

2.3 Related Research 12

3.0 THE ROLLING PRIORITY ALGORITHM 15

3.1 Specification of the Algorithm 15

3.1.1 Service and Drop Policies 16

3.1.2 Phase Randomization 17

3.2 Properties of Rolling Priority 18

3.3 Performance Under Heavy Traffic 19

3.4 Routing Tradeoffs 20

4.0 CONCLUDING REMARKS 24

5.0 PROBLEM DEFINITION AND PRELIMINARIES 26

5.1 The OQ Emulation Problem 27

5.2 Summary of Results 29
5.3 Related Work 31
5.4 Organization 32
6.0 SWITCH MODELS 33
6.1 OQ Algorithms 33
6.2 CIOQ Policies 35
6.3 Families of OQ Algorithms 37
7.0 OQ EMULATION OF NON-PREEMPTIVE SCHEDULING ALGORITHMS 40
7.1 The Speedup — Buffer Capacity Trade-off 40
7.2 The Critical Cells First CIOQ Policy 42
7.2.1 OQ Emulation using CCF and G-CCF 45
8.0 OQ EMULATION WITH PREEMPTION ALLOWED 49
8.1 The Speedup Lower Bound 49
8.2 CCF Is Not Better Than The Worst Greedy Policy 53
9.0 THE CCF-EAF HYBRID CIOQ POLICY 57
9.1 The CEH CIOQ Policy 58
9.2 Performance of CEH 60
10.0 CONCLUDING REMARKS 62
BIBLIOGRAPHY 64

LIST OF TABLES

1 Summary of OQ Emulation Results 62

LIST OF FIGURES

1 Output-Queued Router Architecture 8
2 Tail-Buffered Links 8
3 Network Sessions 8
4 Session Priorities Under the Rolling Priority Algorithm 17
5 Session Epochs Under the Rolling Priority Algorithm 17
6 FCFS/RD versus RP-n Under Heavy Load 21
7 Routing Tradeoffs at $B = 5$ 22
8 Routing Tradeoffs at $B = 7$ 23
9 Routing Tradeoffs at $B = 10$ 23
10 Output Queued and Combined Input-Output Queued Architectures 34
11 Illustration of the Proof of Theorem 8.1 51
12 Illustration of the Proof of Theorem 8.2 55

1.0 INTRODUCTION

Scheduling in packet routing networks can be described as allocating communication link resources (i.e., transmission and buffering resources) to packets over time. Until recently, the performance of scheduling algorithms in packet networks has mostly been studied in terms of packet delay and stability (boundedness of backlog). These studies, for example [26, 8, 45, 28, 12, 22, 27, 6], have led to valuable insights into the behavior of scheduling algorithms, such as FCFS and Generalized Processor Sharing (GPS). However, in investigating delay and stability, the packet network is modeled as a queuing network where communication links are represented by servers with infinite waiting room, which limits the practical value of the resulting algorithmic guarantees.

Delay and stability guarantees lead to bounds on buffer occupancy that can be leveraged in dimensioning buffer capacities at the router ports to prevent, or at least bound, packet loss. These buffer occupancy bounds are often dependent on network parameters, such as link capacities and the network diameter, which are impractical to track in today's large decentralized networks. More importantly, relying on such bounds for buffer dimensioning ignores the technological constraints on buffer capacity, which have recently risen due to increasing link speeds [2, 17], and the drive toward constructing photonic packet switches with integrated optical packet buffers [33, 9, 5].

Under stringent buffer capacity constraints, the packet loss rate (ratio of dropped packets to those offered to the network) becomes the primary metric in the evaluation of scheduling algorithms. This dissertation looks at two fundamental scheduling problems related to minimizing the loss rate. To date, this area of research has remained largely unexplored, with only few known results.

The first part of this dissertation studies the problem of minimizing the maximum session loss rate in networks with *output-queuing* routers (packet switches). The second part investigates the extra speed and buffering capacity required in *Combined input-output queuing* (CIOQ) routers — a scalable and widely adopted router architecture — so that the results obtained in the first part carry over to networks of CIOQ routers.

1.1 MINIMIZATION OF THE MAXIMUM SESSION LOSS RATE

Recent research has shown that TCP-NewReno flows traversing a single work-conserving link having a small buffer are able to withstand high loss rate and achieve good link utilization, under assumptions that limit the contribution of each flow to the total link load [17]. However, several questions regarding the performance of networks with small router buffers remain open. This work is motivated by one question that is critical to the utility of such networks: What statistical guarantees on the packet loss rate experienced by user flows (or aggregates thereof) can be supported by a network with small router buffers, without imposing severe restrictions on the link utilization or the routing path length?

Given the load at the network links and the link buffer capacities, the loss rate along a network path is determined by three factors: (1) the packet arrival process, (2) the packet size distribution, and (3) the scheduling algorithm (i.e., the service discipline and the drop policy) used at the links. The effect of variability in the arrival process and the benefit of limiting burstiness by regulating the arrival process have been well studied and understood [40, 36]. Similar queuing-theoretic results apply to the distribution of packet sizes; constant packet sizes are desirable when the objective is to minimize the rate of buffer overflow events at a link (the frequency of exceeding a certain buffer occupancy threshold). In contrast, there are only few known results concerning the performance of scheduling algorithms in networks with small or fixed-size buffers [1].

Motivated by the need for loss-rate guarantees in networks with small buffers, this work studies the problem of link scheduling to minimize the maximum session loss rate.[1] Al-

[1] A session is a traffic aggregate between two network routers, routed along a single network path.

gorithms that perform well on this metric enable network service providers to offer strong loss rate guarantees to their customers (e.g., as part of their service level agreement) while maintaining good utilization of the network capacity (hence, revenue).

In this part of the thesis, we introduce Rolling Priority, a local online work-conserving algorithm with the following properties: (1) The algorithm does not favor any session over others (in terms of packet loss rate) at any link, (2) it ensures that some packets from each session are subject to a small packet loss probability (much smaller than the average packet loss rate at the link) at every link along the session's path, and (3) it maximizes the proportion of packets from each session that are subject to such a small loss probability at every link. Intuitively, Rolling Priority performs well by minimizing the proportion of packets in each session that face a significant loss probability at any link. We show that this algorithm enables the network providers to offer strong loss rate guarantees while maintaining good utilization of the network capacity.

1.2 EXACT EMULATION OF OQ ROUTERS USING CIOQ

In the first part of this dissertation, as well as in the general packet scheduling literature, performance analysis of packet scheduling algorithms commonly assume that routers use Output Queuing (OQ) [1, 21, 22, 26, 8]: At each time step, all newly arriving packets are switched to their corresponding outputs where they are stored awaiting transmission. The simplicity of this model is attractive for analysis purposes since each router output is accurately viewed as a single-server queue controlled by an instance of the scheduling algorithm, independently of other router ports. However, a well-known practical limitation of OQ is that in a router with N ports, the switch fabric must have a *speedup* of N — it must transfer packets to the output at a speed N times the speed of the communication links. This limits scalability while the router sizes continue to grow beyond hundreds of ports.

To overcome the scalability problem, most packet routers use Combined-Input-Output Queuing (CIOQ) [10]: At each time step, up to s, $s \ll N$, packets can be switched from any input port to their corresponding outputs, and up to s packets can be switched to any

output port, so that the router's switch fabric may operate at a speedup of only s. The switch fabric is most commonly an unbuffered switching (cross-bar) matrix. Combined-Input-Output Queuing comes at the cost of maintaining additional packet buffers at the input ports (hence the name) and arbitrating access to the switch fabric among packets at the input ports. Switch-fabric arbitration may introduce dependence among packet arrivals at different outputs, thus complicating the analysis of scheduling algorithms in CIOQ routers.

A question that naturally arises is whether provable packet loss and delay guarantees provided by a scheduling algorithm in networks of OQ routers carry over to networks of CIOQ routers. This is the case if the CIOQ routers *emulate* the OQ routers: for any scheduling algorithm in a given class and any sequence of packet arrivals, replacing an OQ router with a CIOQ router does not change the sequence of dropped packets at the router, the order of packet departures from each output, or the departure times. Since packet loss and delay guarantees obtained by analyzing networks of OQ routers carry over to networks of OQ-emulating CIOQ routers, studying the minimum CIOQ switch speedup and buffer capacity required for OQ emulation is of practical and theoretical relevance. We refer to this problem as the *OQ emulation* problem.

In their seminal paper [10], Chuang, Goel, McKeown and Prabhakar studied a special case of the OQ emulation problem, where the output buffers in the OQ router, and both the input and output buffers in the CIOQ router are sufficiently large to prevent packet loss under every possible packet arrival sequence. Under this assumption, the drop policy of any scheduling algorithm is never exercised. Therefore, a CIOQ router emulates an OQ router if for every PIFO (Push-in-First-Out) service discipline and packet arrival sequence, the order of packet departures from each CIOQ router output and the departure times, are identical to the corresponding sequences for the OQ router. For this setting, the authors of [10] identified an arbitration policy that achieves OQ emulation at speedup 2, and showed that no arbitration policy can achieve OQ emulation at speedup $2 - 1/N$, where N is the number of input/output ports.

The second part of this dissertation studies the general OQ emulation problem (i.e., including the emulation of OQ loss behavior) when the buffer capacity at any router port cannot exceed some capacity $B > 1$. Specifically, it investigates the required buffer capacity

at the input ports of a CIOQ router and the required switch fabric speedup so that it can emulate an OQ router with B packet buffers at every output. In this setting, a particular concern is that buffer overflows at the input ports of a CIOQ router may lead to dropping packets that are not dropped by the OQ router.

The main findings in this part are as follows: Whereas a CIOQ router can emulate any non-preemptive OQ algorithm in the family above at a fabric speedup of 2, emulation of preemptive algorithms requires $\Omega(B^{\frac{1}{3}})$ speedup. We give a CIOQ policy for the emulation of any preemptive algorithm at $O(B^{\frac{1}{2}})$ speedup. This result suggests that the emulation preemptive OQ algorithms may be feasible only for OQ routers with small buffers.

1.3 ORGANIZATION OF THE DISSERTATION

The dissertation is organized into two self-contained but complementary parts, corresponding to the problems described in the previous sections. Whereas the loss rate minimization problem is defined on networks of OQ switches, the emulation problem involves scheduling packets within one CIOQ switch. As a result, each problem has its own network or switch model and related literature. This is reflected in the organization of the dissertation, with each of the parts including its own model, review of related research, and concluding remarks.

Part 1.3 (Ch. 2–4) covers the minimization of the maximum session loss rate in networks of OQ switches, and Part 4 (Ch. 5–10) covers the emulation of finite-buffered OQ switches using CIOQ.

2.0 PROBLEM DEFINITION AND PRELIMINARIES

In this part of the thesis, we study the minimization of the maximum session loss rate. We introduce and analyze the Rolling Priority algorithm, which is shown to offer better loss-rate guarantees compared to well-known algorithms such as FCFS/Drop Tail.

This chapter defines the scheduling problem by specifying the input to a scheduling algorithm and the objective function. It also reviews related work.

2.1 PROBLEM DEFINITION

2.1.1 Problem Parameters

For any scheduling algorithm, the worst-case input that determines the performance of the algorithm depends on the range of values that may be assumed by the network and traffic parameters. In this work, an instance of the scheduling problem is characterized by a triplet of parameters (B, N, H) where B is the link buffer capacity, N is the maximum number of sessions that may traverse a link, and H is an upper bound on the path length of every session. This work seeks to find an algorithm that performs well on all parameters in the *small-buffer* regime where $B << N$. The performance objective is formally stated in Section 2.1.4.

For a given triplet $P = (B, N, H)$, $\mathcal{I}(P)$ denotes the set of inputs obeying the restrictions imposed by the parameters.

2.1.2 Input Specification

For a given $P = (B, N, H)$, an input $I \in \mathcal{I}(P)$ is a triplet $(\mathcal{N}, \mathcal{S}, \mathcal{J})$, where $\mathcal{N}$ is a network represented by its topology, $\mathcal{S}$ is a collection of permanent sessions in $\mathcal{N}$, and $\mathcal{J}$ is a sequence of timed packet injections into $\mathcal{N}$ by the sessions in $\mathcal{S}$. The inclusion of the network topology as part of the input is motivated by the interest in algorithms that perform well on any network. This work considers only the case where all packets are of uniform size and equal importance. It is known that scheduling problems in this setting are not trivial [1]. Moreover, since the variability in packet sizes generally increases the frequency of buffer overflow events [22], one would expect networks with small buffers to segment or pack incoming packets into fixed-size ones at the network's edge.

In a network $\mathcal{N} = (V, E)$, the set of vertices V represents the output-queued network routers, and the set of directed edges $E \subseteq V \times V$ represents the network links. A link $e = (u, v)$ is equipped at its tail, u, with a buffer of capacity $B \geq 0$ packets. Each router in $v \in V$ is equipped with a set of input and output ports. An input port is *internal* if it is connected to the head, v, of a link (u, v) in E. Otherwise it is called an *external* input port. Similarly, an output port is called internal if it is connected to the tail of a link in E, and called external otherwise. Each session is associated with a dedicated external input port where it injects packets at its source router and with a dedicated external output port at its destination router, where packets leave the network unless dropped at some upstream router. Figs 1–3 illustrate the elements of a network.

Each session in $\mathcal{S}$, identified by its source-destination node pair, is assigned a fixed path and continuously injects packets at its source node. It has a bandwidth demand specifying its average packet injection rate. A session's path length and bandwidth demand are disseminated only to routers along its path. Recall that N and H are upper bounds on the number of sessions traversing any given link and the path length of any session, respectively.

Time proceeds in discrete steps. Each time step is divided into a forwarding substep and a switching substep. During the forwarding substep, each link transmits a packet from those present in its buffer, if any. The packet becomes available at the input port connected

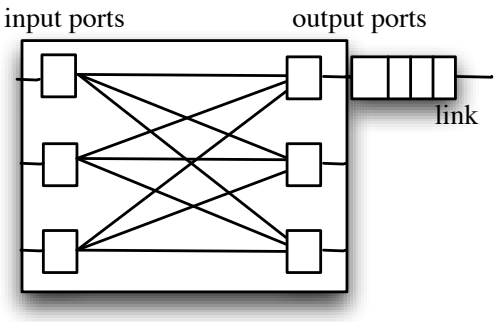


Figure 1: An output queued router (switch) has a fully-connected switch fabric. Packets arriving at the input ports are immediately switched to the corresponding output ports. Packets may be stored in the link buffers at the output pending transmission.

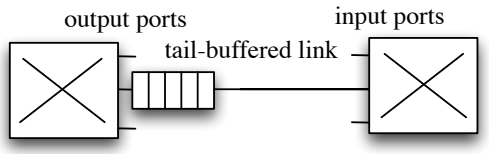


Figure 2: A link is a directed edge attached to an output port at its tail, and to an input port at its head. At its tail, a link is equipped with a buffer of capacity B. A router port is internal if it connects the router to another one through a link in E. It is external otherwise.

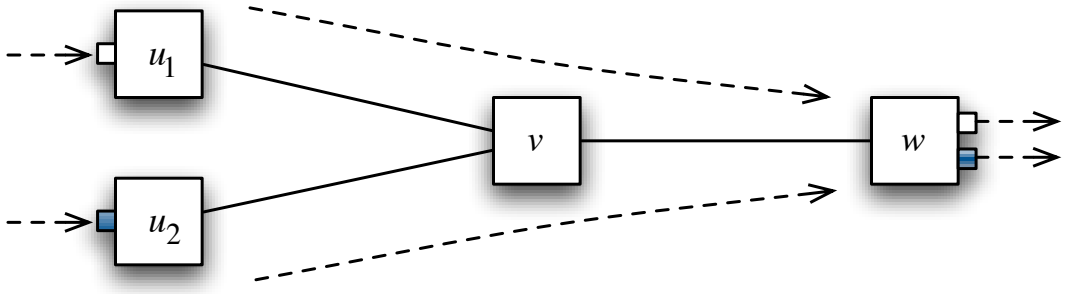


Figure 3: A session routed along a path (u_1, v, w) injects packets into a dedicated external input port at u_1, and its successfully delivered packets depart from the network through a dedicated external output port at w. The diagram also shows a session routed along (u_2, v, w). The two sessions share link (v, w).

to the head of the link after a propagation delay of an integral number of time steps. During the switching substep, each router moves all the packets available at its input ports to the appropriate output ports according on their respective paths. If, at the tail of any link, the number of packets requiring storage exceeds the buffer capacity, B, the scheduling algorithm must drop the excess packets.

The sequence of packet injections, $\mathcal{J}$, is a finite set $\mathcal{J} \subset \mathcal{S} \times \mathbb{Z}$. A session can inject at most one packet every time step into its external port. That is, for any t: $(s,t),(s',t) \in \mathcal{J}$ implies $s \neq s'$.

2.1.3 The Arrival Process

Traffic injection by each session is assumed to be a counting process with independent increments [38].[1] The number of arrivals in disjoint time intervals of the same length are iid random variables. Observe that the Poisson process is a special case of the above, where the number of arrivals in any time interval follows the exponential distribution. The injection rate of every session is also assumed to be an integral multiple of $\frac{1}{T_0}$, for some $T_0 \in \mathbb{Z}^+$.

The Rolling Priority algorithm, introduced in the next chapter, is designed to support a loss-rate guarantee over consecutive intervals of length T_0.

2.1.4 The Performance Metric

For a set of parameters $P = (B, N, H)$, let $I \in \mathcal{I}(P), I = (\mathcal{N}, \mathcal{S}, \mathcal{J})$, and let Σ be the set of all sequences of n disjoint intervals of length T_0 steps ($n > 0$). Further, let $A(I)$ denote the cost of algorithm A on input I. $A(I)$ is defined as:

$$A(I) \triangleq \max_{s \in \mathcal{S}} \max_{\sigma \in \Sigma} \frac{1}{n} \sum_{i=1}^{n} \mathbf{E}\big[X^A_{\sigma,i}(I)\big], \tag{2.1}$$

where $X^A_{\sigma,i}(I)$ is a random variable representing the fraction of packets dropped by algorithm A among those injected during the ith interval of σ. Obviously, the distribution of $X_{\sigma,i}$

[1] A stochastic process $\{\Gamma(t) : t \geq 0\}$ is a counting process if $\Gamma(t) \geq 0$ for every t, and for any $s > t$, $\Gamma(s-t) = \Gamma(s) - \Gamma(t) \geq 0$.

depends on the intensity of packets arrivals and the scheduling decisions made by A. $A(I)$ is the maximum of this expectation over all sequences in Σ – across all sessions. We refer to a session and sequence of intervals that determine the cost of A on input I as a *critical session* and a *critical sequence*, respectively. The sequence size n can be any positive integer. The effect of the sequence size on performance is explored in Ch. 3.

The cost of an algorithm A on P is its cost on the *worst* input in $\mathcal{I}(P)$. For the worst input to be well defined, $\mathcal{I}(P)$ is made finite by requiring that the number of routers in a network and the length of the packet injection sequence be bounded above by arbitrarily large constants defined in terms of B, N, and H.

Remarks. The cost of an algorithm A on an input I is the maximum expectation of the loss rate among all same-session sequences of packet injections. Using the maximum expectation as a metric reflects the interest in capturing the *average* performance of the algorithm on a critical sequence of intervals under the worst input.

Characterizing the performance of each algorithm based on its worst possible input is a natural choice when loss-rate guarantees are sought. The small-buffer regime described above is also a natural choice since, for any algorithm, increasing the number of sessions that can share a link relative to the buffer capacity permits inputs with higher packet loss rate.

2.2 SUMMARY OF RESULTS

In evaluating the worst-case performance of an algorithm, the input can be viewed as being generated by an adversary, whose goal is to maximize the loss rate of some session. This work assumes that the adversary fully specifies the input, including the complete sequence of packet injections, prior to injecting the first packet and without revealing it to the algorithm. Thus, if the algorithm is randomized, the adversary may not adjust the input based on the randomized decisions taken by the algorithm.

This work quantifies the loss guarantees offered by an algorithm when the sequence of packet injection from any given session obeys the statistical restrictions imposed by the

stochastic process of Section 2.1.3. The restrictions on the injection process limit the packet injection rate as well as the burstiness of arrivals. A network operator would naturally impose such statistical restrictions to be able to offer strong loss guarantees, while accepting as much revenue-generating traffic as possible. The arrival process assumed here is a generalization of the Poisson process. The guarantees are evaluated using analytical modeling and simulation. These results, along with accurate analytical models for characterizing the performance of FCFS/RD (Random Drop) and *Rolling Priority* (introduced below) as a function of the problem parameters, (B, N, H), have been published in [14, 15, 13].

For the general arrival process defined above, we introduce the Rolling Priority algorithm, which possesses the desired characteristics outlined below, and is designed to support a loss-rate guarantee over consecutive intervals of length T_0 steps:

- The algorithm assigns to each packet a rank drawn uniformly and independently at random from $[1, N]$, and the rank remains fixed throughout the packet's sojourn in the network.
- At each link, the algorithm transmits the packet with highest rank in every time step.
- In case of buffer overflow at an link, the algorithm drops packets with the lowest rank among those in the link's buffer and the new arrivals.

As we shall see in the next chapter, under Rolling Priority, the rank assigned to a packet may vary from one link to the next, but remains within a narrow range.

The performance of Rolling Priority is compared to FCFS/RD (non-preemptive random drop) analytically and using simulation. In FCFS/RD, if the number of packets arriving simultaneously at a link exceeds the available buffer space by e packets, then e randomly selected packets among the new arrivals are dropped. FCFS/RD is used as a representative of algorithms that do not favor individual sessions or packets over others at any link, and do not give any preferential treatment to packets that already consumed upstream resources.

The evaluation is carried in two different settings. First, in a heavy-traffic setting, which may be induced by link failures and the ensuing rerouting of traffic, leading to heavy load at the surviving links. Under such conditions, a network operator may guarantee a minimum throughput (the complement of the loss rate) to each session for the purpose of delivering

critical traffic. Second, in the normal operation setting, where the operator may seek to accept and route as many sessions as possible within constraints on routing path length, H and the maximum load at the link, which is represented by N when the sessions have identical bandwidth demand. The constraints are chosen to guarantee a constant bound on the loss rate of every session.

Under heavy Poisson traffic, Rolling Priority is shown analytically to support throughput guarantees that are nearly insensitive to the path length parameter, H. For example, it guarantees the delivery of nearly 67% of the packets injected by each session for paths up to 50 hops in length at 90% link utilization when the buffer capacity is 5 packets. This is in contrast to FCFS/RD which provides only 20% throughput guarantee under the same conditions. At 10 hops, Rolling Priority provides a throughput guarantee that is 10% higher than FCFS/RD (corresponding to a 44% improvements in the guaranteed loss rate bound). The performance gap grows in favor of Rolling Priority with increasing the limit on the path length, H.

In the normal-operation setting, simulation with periodic session traffic shows that for a given upper bound on the session loss rate, the Rolling Priority algorithm provides better load-path length trade-offs compared to FCFS/RD. For instance, at $B = 5$ and 60% link utilization, Rolling Priority increases the maximum path length at which the network can support a loss rate guarantee of 0.02 by 40% (from 15 to 21 hops). This can improve the network provisioning cost by reducing the link density (the ratio of links to routers) required to support anticipated traffic.

2.3 RELATED RESEARCH

As mentioned in motivating this work, the performance of scheduling algorithms in packet networks has mostly been investigated in terms of packet delay and stability (i.e., boundedness of backlog) guarantees, for example [26, 8, 45, 28, 12, 22, 27, 6]. However, the resulting buffer occupancy bounds fail to account for the technological constraints on buffer capacity, which have recently risen due to increasing link speeds [2], and the drive towards construct-

ing photonic packet switches with integrated optical packet buffers [33, 9, 5]. In networks with finite buffering capacity, the packet loss rate and throughput are the primary metrics for the evaluation of scheduling algorithms. Surprisingly, to date this area of research has remained largely unexplored, with only few known results.

Maximization of the total network throughput (the number of successfully delivered packets during a bounded interval) has been studied within the competitive analysis framework [1, 21, 4, 37]. In [1], Aiello et al. show that whereas every work-conserving algorithm is competitive on DAGs,[2] on general networks, Nearest-To-Go (NTG), Longest-In-System (LIS) and Furthest-From-Origin (FFO) are competitive, but FCFS/Drop Tail and Furthest-To-Go (FTG) are not.

The problem studied here is related to the network throughput problem as follows. The Rolling Priority algorithm is competitive for the network throughput problem on general networks, as it guarantees (in expectation) the delivery of a non-zero fraction of packets injected by each session. NTG, LIS and FFO on the other hand, may starve sessions traversing multiple hops. The author is not aware of any prior research on per-session throughput problems within the competitive analysis framework.

The work by Reisslein et al. [35] provides a bufferless-multiplexing framework for supporting statistical delay guarantees in multihop networks. Using traffic regulation at the ingress and bufferless multiplexing at the core, they transform the problem of providing ingress–egress delay guarantees into one of providing loss guarantees. The loss bounds are obtained using an approximate fluid-multiplexer model. The fluid model may severely underestimate the loss probability in packet multiplexers (links) because of the assumption that flows can have a fixed peak transmission rate (smaller than the link capacity) across all time scales. Under this assumption, a bufferless fluid multiplexer can simultaneously serve multiple flows without incurring "fluid" loss. Note that on the other hand, a packet multiplexer (a link) can only serve one packet (thus one flow) at a time at time scales smaller than the packet transmission time. The scheduling order (service discipline) is trivial in the bufferless multiplexing model. For the drop policy, the authors assume that if at any instant

[2] An algorithm is competitive if it has a bounded competitive ratio relative to an optimal offline algorithm. That is, the competitive ratio does not grow asymptotically with the length of the packet injection sequence.

the sum of flow rates exceeds the capacity of the link, fluid loss is shared proportionally among flows. As we shall see, this is not readily satisfied by scheduling algorithms.

Finally, a note regarding Active Queue Management (AQM) schemes [18]. These schemes, most notably Random Early Detection (of congestion) (RED) [19] attempt to prevent loss synchronization and fairly apportion loss among TCP flows sharing a common bottleneck by voluntarily (probabilistically) dropping packets without buffer overflow. RED has been evaluated on a single bottleneck with small buffer and was shown to perform poorly in this setting [32]. The reason however is shared among all AQM schemes, which are designed to detect the onset of congestion (overload) by observing the buffer occupancy using a moving average over a long time interval, rather than observing the instantaneous queue length. These schemes are too slow to react when the buffer capacity is small such that loss occurs without persistent overload.

3.0 THE ROLLING PRIORITY ALGORITHM

This chapter introduces the Rolling Priority algorithm which possesses the following characteristics: (1) it does not favor any session over others at any link, (2) it favors individual packets over others but treats each packet consistently throughout its sojourn in the network (favorably or otherwise), and (3) it results in low probability of contention between favored packets. Rolling Priority is designed to support loss-rate guarantees over a sequence of consecutive intervals of length T_0 under the arrival process specified in Section 6. In the following sections, the algorithm's performance is compared to that of FCFS/RD under heavy and moderate network load.

An overview of the results is presented in Section 2.2, along with the motivation for the choice of load settings, and the choice of FCFS/RD as the baseline algorithm.

3.1 SPECIFICATION OF THE ALGORITHM

The Rolling Priority (RP) algorithm is based on the concept of session epochs. From the perspective of a session, time is divided into disjoint *epochs* of $T_e = nT_0$ time steps (n is a parameter of the algorithm hence the epoch length is the same for all sessions). The boundaries of a particular session epoch are shifted from one link to the next along the path of the session by the link's propagation delay. The epoch boundaries for different sessions are not synchronized.

The algorithm was originally proposed for Optical Burst Switching (OBS) networks [15, 14], which are based on advance reservation of transmission opportunities. As described in

this chapter, RP can also be adopted in packet networks without any changes. However, the origins of the algorithm affect some design decisions. In particular, RP does not assign a fixed rank to each packet, instead, it guarantees (under some mild conditions) that the rank of a packet remains within a narrow range throughout its sojourn in the network.[1,2]

3.1.1 Service and Drop Policies

At every time step, RP gives scheduling priority (service and drop priority) to sessions sharing the link in the order of earliest-starting current epoch, where the current epoch of a session at a given time step is the unique session's epoch spanning that step. The session(s) with earliest-starting current epoch have the highest priority. Figure 4 illustrates the assignment of priority at different time steps at a link shared by three sessions a, b and c. At every time step in the interval $[t_1, t_2)$, the current epoch for session b started earlier than the current epochs of sessions a and c. As a result the priority of session b is highest within this interval. The highest priority session during $[t_2, t_3)$ is c, and it is a during $[t_3, t_4)$. The cycle repeats with the start of a new epoch of session a. The cyclic priorities can be enforced using a circular queue as shown in the figure. At the beginning of the time step, if the number of packets available at the link (those already in the buffer and those offered by the router's input interfaces) exceeds the buffer capacity B, the excess packets are dropped. RP drops packets from the least priority sessions so that the B packets with highest session priority remain. During the remainder of the time step, RP serves a packet from the highest priority session with backlog, if any.

[1] In OBS, each session periodically requests transmission opportunities at particular time steps during a future epoch. This setting prevents the algorithm from effectively using assigned ranks or priorities to packets since transmission requests are granted in the order they are received and a granted request cannot be revoked.

[2] It is assumed that a packet injected in a given session epoch remains within the epoch boundaries throughout its sojourn in the network.

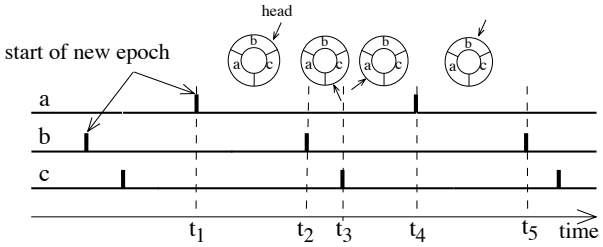


Figure 4: The priorities of three sessions a, b and c at a link. A circular queue is used to enforce the cyclic priorities. The `head` pointer indicates the session with highest priority at any given time.

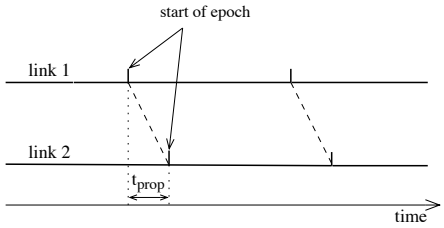


Figure 5: The start of a session epoch at two consecutive links (link interfaces) differs by the propagation delay of the upstream link (t_{prop}).

3.1.2 Phase Randomization

To ensure high priority packets are subject to a small loss probability at every link, RP uses randomization to avoid contention among a large number of high priority sessions at any link. Furthermore, RP loosely aligns the start of session epochs across the links it traverses so that a packet that is given high priority at a link is likely to have high priority at all links along the path. Both randomization and epoch alignment are part of session initialization that we now describe.

Each session has an associated *phase* variable ϕ. Suppose the session is initialized at time t_0. The ingress router of the session chooses the value of the phase uniformly at random from the interval $[0, T_e)$ so that the session starts a new epoch at time $t + \phi + iT_e$, $i = 0, 1, 2, \ldots$. The phase of the session is communicated to downstream links in the form of a one-time initialization packet, `init`, sent from the ingress at time $t_0 + \phi$. The reception time of the `init` packet at a given link specifies the session's epoch start times at that link. For instance, if an `init` packet for a particular session is received at the link at time t, then a new epoch for the session at that link starts at times $t + iT_e, i \geq 0$. Because RP's service and drop policies rely on the knowledge of session epoch boundaries, the `init` packets are always given higher scheduling priority than all data packets so that they are almost never

dropped.[3]

Figure 5 shows a timing diagram with two links in tandem along the path of a session. The session's `init` packet does not experience any queuing delay. In this case, the start of a new session epoch at the upstream link precedes the start of a new epoch at the downstream link by exactly the propagation delay of the upstream link.

3.2 PROPERTIES OF ROLLING PRIORITY

Let RP-n denote the algorithm RP with epoch duration $T = nT_0$ for some $n \geq 1$. We view each epoch as being composed of n consecutive subepochs, numbered $1, \ldots, n$, of length T_0 slots.

Consider an epoch e of a session s at a link l along its path. If the number of sessions sharing the link is N_l, then the number of sessions (with packets) of higher scheduling priority during the kth subepoch, $k = 1, \ldots, n$, is approximately a binomial random variable with success probability $\frac{n-k+1}{n}$, and expectation $N_l \frac{n-k+1}{n}$. This follows from phase randomization and applies to every epoch of any session s, at any link along its path.

Since the behavior above applies to all sessions, RP-n does not favor any session over others at any link. Clearly, RP-n assigns different scheduling priority to different packets, but every packet receives consistent treatment at every link — in terms of the proportion of higher priority traffic it contends with — as long as each packets remains within (or close) to the subepoch where it was injected.

Note also that as the number of subepochs, n, increases it becomes unlikely that many sessions start new epochs within the same subepoch. That is, contention among high-priority packets becomes unlikely.

In [15], we showed that for a set of parameters (B, N, H), the worst-case input for the

[3]Initialization packets are dropped only when there are too many `init` packets at a given link, but such packets are rare since sessions are traffic aggregates that are supposed to persist for long time (i.e., weeks or months). Recovery mechanisms from the loss or corruption of `init` packets is not part of the scheduling algorithm but should be provided, for example by having link interfaces notify the ingress routers of packets belonging to uninitialized sessions before dropping them.

Rolling Priority algorithm is one where a session s traverses exactly H hops and at each hop it contends with exactly $N-1$ sessions. The path of each of the background sessions shares only one link with that of session s. Moreover, background sessions do not face any contention upstream of the link shared with s.

3.3 PERFORMANCE UNDER HEAVY TRAFFIC

In this section, we compare the performance of FCFS/RD and RP-n. We find that under light load, FCFS/RD performs nearly as well as RP-n, But that the difference in loss rate bounds grows quickly with load.

Consider a session s routed along a path of length h link, competing at each link with $N-1$ one-hop sessions. Ignoring the effect of load thinning affecting s at upstream links, let the loss rate at every link along the path be β. Given the stationarity and independent increments properties of the arrival process, under FCFS/RD the individual packets are indistinguishable. That is, the loss probability of each packet at any link is also β. Thus the expected loss rate for session s averaged over any sequence of packets under FCFS/RD is given by:

$$\begin{aligned} M_s^{\text{FCFS/RD}} &= 1-(1-\beta)^h \\ &\approx 1-e^{-h\beta}, \qquad \text{for large } h. \end{aligned} \tag{3.1}$$

Under RP-n, consider an epoch of session s and let its subepochs be numbered 1 through n. By the properties of RP-n above, the expected scheduling priority for the session's packets during subepoch i linearly improves with increasing i. Suppose that there exists $q \in (0,1]$ such that the priority for all subepochs beyond subepoch $\lceil nq \rceil$ is high enough that the loss rate is much smaller than β. Specifically, let β_j denote the loss rate during the jth subepoch, and suppose $\frac{\beta}{\beta_j} \geq \alpha^p$ for $j : \lceil nq \rceil \leq j \leq n$ and for constants $\alpha > 0, p > 1$. Then under

RP-n,

$$M_c^{\text{RP-}n} < q + (1-q)\left[1 - \left(1 - \frac{\beta}{\alpha^p}\right)^h\right]$$
$$\approx q + (1-q)(1 - \sqrt[\alpha^p]{e^{-h\beta}}). \tag{3.2}$$

Comparing (3.1) and (3.2), we find that under moderate-to-heavy load, (e.g., $\beta > 0.01$), $M_c^{\text{FCFS/RD}}$ quickly approaches 1 as h increases. In contrast $M_c^{\text{RP-}n}$ grows slowly with h due to the α^p root. Furthermore, when the range of h is such that $h\beta \ll \alpha^p$, $M_c^{\text{RP-}n}$ saturates at q.

The difference in bounds between RP-n and FCFS/RD depends on how fast the loss rate drops across subepochs under RP-n. As an example consider the above network when traffic arrival correspond to a Poisson process. For simplicity we model each link as an M/M/1/B queue so that at load ρ the loss probability is approximately ρ^{B+1} where B is the buffer size. Under FCFS/RD $\beta = \rho^{B+1}$ and $M_c^{\text{FCFS/RD}} = 1 - (1-\rho^{B+1})^h$. Under RP-$n$, the traffic load that s contends with decreases linearly throughout the epoch. Thus for subepoch i, we have $\rho_i \approx \rho(1 - \frac{i}{n})$ and $\frac{\beta}{\beta_i} \approx \frac{\rho^{B+1}}{\rho^{B+1}(1-\frac{i}{n})^{B+1}} = (\frac{n}{n-i})^{B+1}$, which is on the form α^p.

The plot in Figure 6 compares the performance of FCFS/RD and RP-n at different values of n when the load at the links is 90%, and $B = 5$. At $n = 10$, RP guarantees the delivery of nearly 67% of the packets injected by each session for paths up to 50 hops in length indicating that beyond the third subepoch ($q = 1/3$), the loss rate at every link is negligible. This is in contrast to FCFS/RD which provides only 20% throughput guarantee. The RP curve for $n = 2$ highlights the role played by the size of the loss-rate averaging interval. The loss rate curve saturates around 0.5 indicating that loss rate during the second subepoch is negligible at every link.

3.4 ROUTING TRADEOFFS

This section presents a simulation-based comparison of the performance of RP-n and FCFS/RD under light to moderate load. This corresponds to the normal operation setting where the

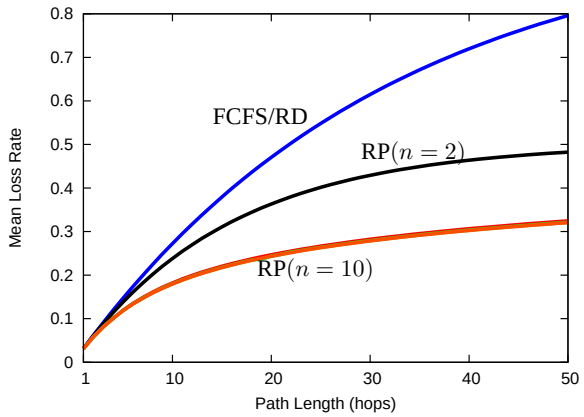


Figure 6: Numerical comparison of FCFS/RD versus RP-n under heavy load.

network operator seeks to support a bounded loss rate to all sessions by limiting the load on each link and the maximum session path length.

The performance of each algorithm is characterized by the loss rate of a *foreground* session routed along a path of h links, competing at each link with $N-1$ one-hop *background* sessions. Packet injection by each session is periodic with period size $T = 100$ steps. During any T consecutive time steps, a session is equally likely to inject a packet at any step. If a session injects a packet at time t, it also injects a packet at $t+T, t+2T, \ldots$. The load at a link is the rate of packet injection by all sessions traversing the link over T consecutive steps. Periodic traffic is chosen here to mimic traffic shaping at the ingress points of the buffer-limited network. In RP-n, each period represents a subepoch, that is $T_0 = T = 100$ steps.

In the simulation experiments, each session has a unit bandwidth demand. That is, every session injects only 1 packet per period. Under the assumption of independent arrival processes, this is shown in [14] to maximize the one-hop loss rate (average over all sessions). Allowing sessions with higher bandwidth demand reduces contention (the probability of buffer overflow in a given time step) at a given load because a session cannot inject more than one packet during a time step.

Figures 7, 8, 9 show the routing tradeoffs for both algorithms at $B = 5, 7, 10$ packets, respectively. In each plot, a contour represents a tradeoff between the routing path length

of the foreground session and the load at the links (number of session traversing a link) to achieve a desired constant loss rate. These tradeoffs are obtained by estimating the average loss rate for the foreground session at each path length and link utilization. The loss rate estimates are obtained by repeatedly running each experiment until the length of the 95-percentile confidence interval is at most 10% of the point estimate.

The figures show that the Rolling Priority algorithm improves the load-path length tradeoffs compared to FCFS/RD. For instance, at $B = 5$ and 60% link utlization (Figure 7), RP-n increases the maximum path length at which the network can support a loss rate guarantee of 0.02 by 40% (from 15 to 21 hops). This can translate to reduction in the network provisioning cost by reducing the link density (the ratio of links to routers) required to support anticipated traffic. Alternatively in the same plot, fixing the maximum path length and the desired loss bound (e.g., at 10 hops and 0.02 loss rate), one can see that RP-n improves the maximum allowed link utilization (hence the operating revenue for a given network) by 2% (from 63 to 65%). Similar observations can be for the other loss rate bounds and buffer sizes.

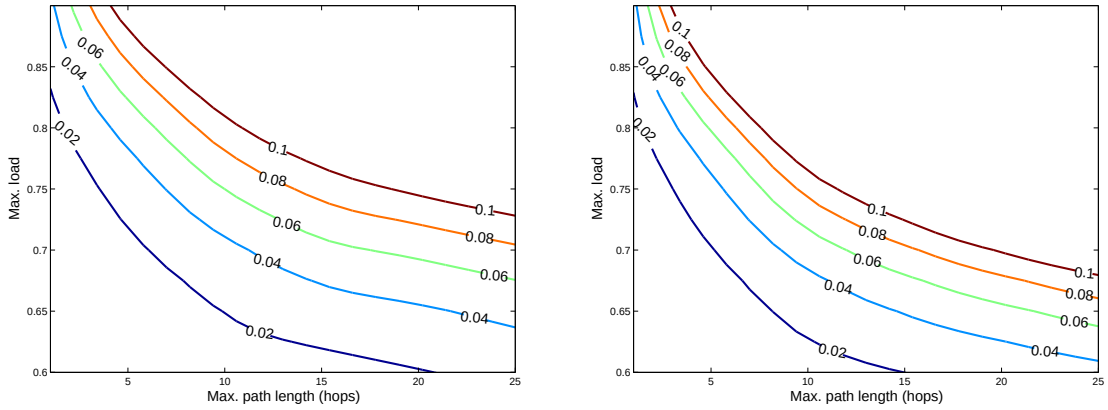


Figure 7: Observed tradeoff between load and path length to achieve a desired loss rate with periodic traffic and $B = 5$. Left: (a) RP-n ($n = 10$), and right: (b) FCFS/RD packet scheduling.

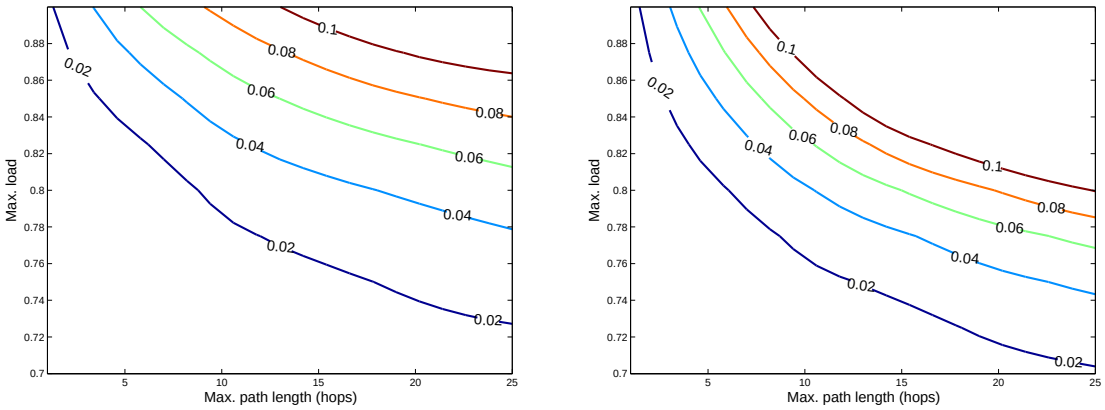


Figure 8: Observed tradeoff between load and path length to achieve a desired loss rate with periodic traffic and $B = 7$. Left: (a) RP-n ($n = 10$), and right: (b) FCFS/RD packet scheduling.

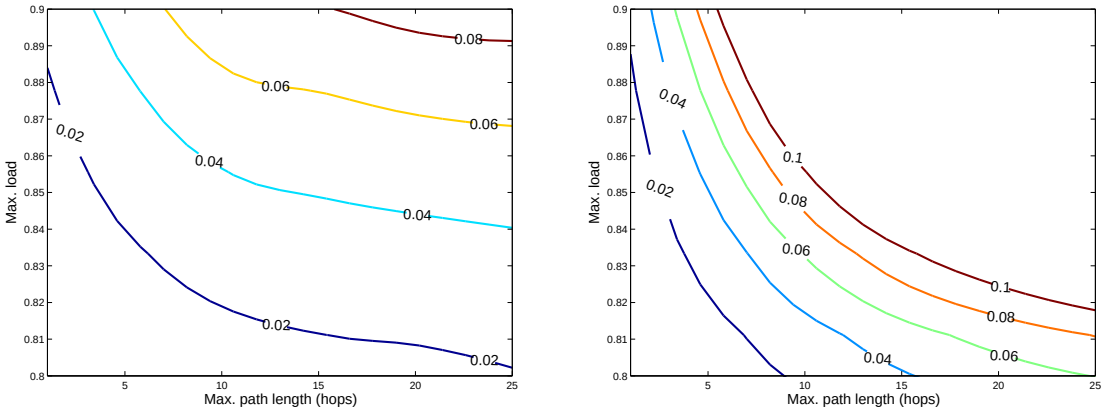


Figure 9: Observed tradeoff between load and path length to achieve a desired loss rate with periodic traffic and $B = 10$. Left: (a) RP-n ($n = 10$), and right: (b) FCFS/RD packet scheduling.

4.0 CONCLUDING REMARKS

This part of the dissertation considered the problem of packet scheduling to minimize the maximum session loss rate in networks of output-queued routers with limited buffering capacity, fixed-size packets, and unit-capacity links.

The problem and the corresponding algorithms are of practical interest to network operators seeking to provide strong loss-rate guarantees to customer sessions, especially in heavy-traffic conditions (e.g., those resulting from link failures). Analysis under heavy Poisson traffic showed that algorithms with the characteristics listed below, represented by Rolling Priority, support significantly stronger loss-rate guarantees with increasing maximum routing path length compared to FCFS/RD, which satisfies only item *(i)*.

(i) The algorithm does not favor any session over others (in terms of packet loss rate) at any link,

(ii) it ensures that some packets from each session are subject to a negligibly small packet loss probability, much smaller than the average packet loss rate at the link, at every link along the session's path, and

(iii) the proportion of packets from each session that are subject to a negligibly small loss probability at every link is as large as possible.

This work can be extended in several directions. The performance objective and algorithms are relevant to a wide class of queuing network applications. A similar objective can be defined for any queuing network with multiple commodities and limited storage capacity at the servers, and where the a server can only serve a limited numbers customers at any instant (as opposed to fluid flow models).[1] One possible extension is to allow jobs (packets)

[1] In a fluid flow model, any number of distinct commodities (flows) can by served simultaneously as long

of different sizes. Although, from a practical perspective, this will result in weaker performance guarantees, it becomes necessary in applications where grouping or fragmenting jobs into fixed-size ones is not possible.

It is immediately obvious that *(ii)* and *(iii)* above are also characteristics of algorithms that might perform well on network-throughput metrics (as opposed to session-loss or throughput metrics), while requiring that none of the sessions is starved. One metric that captures this objective is maximizing the log of the sum of session throughputs. A possible research avenue is to investigate how well Rolling Priority performs on the logarithmic-throughput objective.

A limitation of this work is that in practice, most routers do not follow the output-queuing architecture. The second part of this dissertation studies how the more-prevalent Combined Input-Output Queuing routers can be used to emulate scheduling algorithms running on output-queued routers to achieve the same performance guarantees.

as the sum of the flow rates does not exceed the server (e.g., link) capacity.

5.0 PROBLEM DEFINITION AND PRELIMINARIES

In most Internet switches (routers), each switch output is equipped with a packet buffer, and employs an output scheduling algorithm to resolve contention among packets attempting to access the attached link. A switch output can transmit one packet at a time from the buffer, and this packet then departs the switch. In addition to a service discipline that determines the packet transmission order, the output scheduling algorithm defines a drop policy (also known as the buffer management policy) to deal with buffer overflow events. The most commonly used algorithm is FIFO/Drop Tail where an incoming packet is dropped only if there is no space to store it in the appropriate output buffer, and packets in the buffer are served in FIFO order. A switch's inputs may also be equipped with buffers to hold the incoming packets until they can be delivered to the proper outputs, across the switch fabric. In this work, we consider the setting where packets arrive online, and all the links have equal speed (capacity). Each output can transmit one packet per time step, and there is at most one new arrival at each switch input per step.

Performance analysis of output scheduling algorithms in the above setting, for example [1, 21, 22, 26, 8], often assume that switches are of the Output Queuing (OQ) type. In an OQ switch, at each time step all newly arriving packets are switched to their respective outputs, where they are stored awaiting transmission. This switch architecture allows modeling packet networks as networks of queues where each switch output is accurately represented by a single-server queue controlled by an instance of the output scheduling algorithm, independently of the other switch ports. However, a well-known limitation of output queuing is that in a switch with N input/output ports, the switch must have an internal fabric speed that is N times the speed (capacity) of a link [10]: N packets destined to some output

may arrive at the same time step at different inputs. The switch fabric must then be able to simultaneously transfer the N packets to that output port (i.e., at N time the speed of the switch links). This limits the applicability of output queuing in current switches where scalability, in terms of link speed and the number of ports, is a primary design objective [23].

To avoid the fabric speed as a scalability bottleneck, most packet switches today use Combined Input-Output Queuing (CIOQ): At each time step, up to s ($s \ll N$) packets can be switched from any input port to their respective outputs, and up to s packets can be switched to any output port, so that the switch's fabric may operate at a speedup of only s relative to the link speed. CIOQ switches require packet buffers at the input ports, and a policy (the CIOQ policy) to arbitrate access to the switch fabric among packets stored at the inputs. Contention for access to the switch fabric among packets destined to different outputs complicates the analysis of scheduling algorithms in CIOQ switches.

A question that naturally arises is whether packet loss, throughput, and delay guarantees (including per-session guarantees) provided by any output scheduling algorithm in a network of OQ switches carry over to networks of CIOQ switches. This is indeed the case if replacing each OQ switch with a CIOQ switch does not change the sequence of packet departures from any of the outputs, which motivates the study of OQ switch emulation using CIOQ switches.

5.1 THE OQ EMULATION PROBLEM

OQ emulation is defined informally as follows: A CIOQ switch with N input/output ports emulates an OQ switch of the same size if for any output scheduling algorithm employed by the OQ switch (henceforth, OQ algorithm) and any sequence of packet arrivals, the sequence of packet departures from each CIOQ switch output is identical to the sequence of departures from the corresponding OQ output. The CIOQ switch can emulate the OQ switch if, given its fabric speedup, the CIOQ policy transfers incoming packets to their respective outputs through the fabric in time to meet their departure times from the emulated switch. If this is the case for every arrival sequence, irrespective of the switch size and the capacity of the

output buffers, we say that the CIOQ policy is *valid* for the emulation of the OQ algorithm. A CIOQ policy may be valid for the emulation of a given OQ algorithm under explicitly stated restrictions. In particular, it may be valid only in the *infinite-buffers setting*, in which the output buffers in the OQ switch (and the CIOQ switch) are considered to be of unlimited capacity. A CIOQ policy is valid for the emulation of a family of OQ algorithms if it is valid for the emulation of every algorithm in that family. A formal definition of validity is introduced in Section 6.2.

The *OQ emulation problem* was proposed by Chuang et al. [10], where the objective is to identify CIOQ policies that are valid, in the infinite-buffers setting, for the emulation of a family of OQ algorithms of practical interest, while imposing minimal requirements on the fabric speedup. In the OQ emulation problem, neither the CIOQ policy nor the emulated OQ algorithm has knowledge of future arrivals, and no statistical assumptions are made about the sequence of arrivals.

In the infinite-buffers setting, the drop policy is never exercised and, as such, the OQ algorithm can be defined by its service discipline. In that setting, Chuang et al. [10] introduced Critical Cells First (CCF),[1] a CIOQ policy that is valid at speedup 2 for the emulation of the family of Push-In-First-Out (PIFO) service disciplines, which includes many well-known disciplines such as FIFO (FCFS), Strict Priority, and Weighted Fair Queuing.[2] They also showed, using FIFO as an example, that no CIOQ policy is valid for the emulation of all PIFO service disciplines at speedup $\leq 2-1/N$. Similar results were obtained simultaneously and independently by Stoica and Zhang [42].

In this work, we investigate CIOQ policies for the emulation of OQ switches with fixed buffer capacity $B > 0$ at every output. Our interest in this setting is motivated by the emergence of technological constraints on buffer capacity in high-speed electronic and optical switches, which may limit B to a few dozen packets [17, 5].

Before summarizing our results we describe the framework within which OQ emulation is set [10, 24, 3]: To emulate a given OQ algorithm, the CIOQ switch maintains, at all time,

[1] The terms "packets" and "cells" are used interchangeably throughout the paper.

[2] In a PIFO service discipline, a packet arriving to an output queue can be inserted at any queue location. In each time step, the packet at the head of the queue, if any, departs from the switch.

complete information about the internal state of the OQ algorithm and the configuration (content) of the emulated switch buffers. This information is leveraged so that:

(i) The CIOQ policy can move the packets presently at the inputs to the output side in time for departure.

(ii) The output ports dequeue and transmit each packet that reaches its departure time.

At any time, a packet that is dropped by the emulated OQ algorithm is immediately discarded from the CIOQ buffer where it resides. To implement this framework, the CIOQ switch maintains a model of the OQ switch's output buffers, which is controlled by the OQ algorithm. In every time step, the CIOQ switch updates the model with any new arrivals and observes the algorithm's decisions. Note that this emulation framework applies to randomized as well as deterministic algorithms: Given an arrival sequence, the CIOQ switch emulates the sample path taken by the randomized algorithm.

5.2 SUMMARY OF RESULTS

We evaluate CIOQ policies in terms of the CIOQ speedup required for the emulation of work-conserving OQ algorithms, and the additional buffer capacity needed to prevent buffer overflow events at the CIOQ inputs.The CIOQ switch is assumed to have buffer capacity B at every output (the same output buffer capacity as the OQ switch). To find the buffer capacity needed at each input, we adopt a CIOQ switch model where the buffer capacity at the inputs is infinite, and bound the maximum buffer occupancy, over all arrival sequences, for the CIOQ policy under consideration. The bounds depend only on the switch parameters such as the speedup and the output buffer capacity.

A CIOQ policy is said to be (s, b)-valid for the emulation of a given OQ algorithm if it is valid for the emulation of the algorithm at speedup s and, at that speedup, the buffer occupancy at any CIOQ input does not exceed b. For the family of work-conserving OQ algorithms, we find that whereas any *greedy* CIOQ policy is valid for the emulation of any algorithm at speedup B, no CIOQ policy is valid for the emulation of all algorithms at speedup

$s < \sqrt[3]{B-2}$, when *preemption* is allowed.[3,4]. We also show, using FIFO/Drop Front [44,25] as example, that CCF is not valid for the emulation of preemptive PIFO algorithms at any speedup $s < B$. We then introduce a greedy CIOQ policy, CEH, that is valid for the emulation of all work-conserving OQ algorithms at speedup $s \geq \left\lfloor \sqrt{2(B-1)} \right\rfloor$. Under CEH, the buffer occupancy at any input never exceeds $1 + \left\lfloor \frac{B-1}{s-1} \right\rfloor$. Beside ensuring that packets meet their departure time from the emulated OQ switch, CEH transfers packets destined to the same output in their order of arrival, whenever possible. This prevents the buildup of excessively large queues at the inputs.

For the family of non-preemptive OQ algorithms, we characterize a trade-off between the CIOQ speedup and the input buffer occupancy. Specifically, we show that for any greedy policy that is valid at speedup $s > 2$, the input buffer occupancy cannot exceed $1 + \left\lceil \frac{B-1}{s-2} \right\rceil$. We also show that a greedy variant of the CCF policy is $(2, B)$-valid for the emulation of non-preemptive OQ algorithms with PIFO service disciplines.

Although FIFO/Drop Tail is the most well-known algorithm, many algorithms of practical and theoretical interest use preemptive drop policies. In addition to Drop Front, preemptive policies include Nearest-To-Go [1], which resolves contention in favor of the packets with nearest destination, Strict Priority, and Random Drop, which chooses the packets to drop at random among those in the buffer.

The reason that there is no CIOQ policy capable of OQ emulation at constant CIOQ speedup is that if preemption is allowed all packets buffered at some CIOQ input port may immediately become needed at the corresponding outputs for departure. Thus, the CIOQ speedup must be at least equal to the maximum input buffer occupancy (over all possible arrival sequences). Although we obtain the lower bound using FIFO/Drop Front, similar examples can be constructed for OQ algorithms using the above-mentioned preemptive drop policies.

In addition to shedding light on the effect of preemption on the CIOQ resources required for OQ emulation, the $\sqrt[3]{B-2}$ lower bound, and the speedup required by CEH suggest that

[3] A greedy policy is one that transfers a maximal set of packets from the inputs to the outputs in every time step.

[4] An algorithm is non-preemptive if it may drop packets only by rejecting them upon arrival to an OQ switch's buffer. It is preemptive otherwise

the emulation of any OQ algorithm may be feasible in high-speed and all optical switches with limited buffer capacity [17, 5]. Finally, it is also worth noting that our results continue to hold in the case where the link capacities are not identical. In this case, the CIOQ speedup is the ratio of the fabric speed to the speed of the fastest link.

5.3 RELATED WORK

Whereas OQ emulation in the infinite-buffers setting has been studied extensively, only few studies investigated the emulation of OQ switch with finite buffers. A simulation-based study in [5], suggests that under light traffic conditions, a CIOQ switch with speedup 2 and an input buffer capacity of 2 packets exhibits a loss behavior similar to that of an OQ switch with small output buffers employing the FIFO/Drop Tail scheduling algorithm. This motivated our investigation of whether a similar result can be obtained for any OQ algorithm and under all traffic patterns.

Kesselman and Rosén [24] showed that CCF is $(2, 2B)$-valid for the emulation of the FIFO/Drop Tail algorithm. It is straightforward to show that this result applies to all OQ algorithms combining a PIFO service discipline and a non-preemptive drop policy. The greedy variant of CCF we describe here improves the maximum input buffer occupancy to B at the same computational complexity. Such savings could be of practical significance in all-optical switches.

Attiya, Hay, and Keslassy [3] proposed CIOQ policies for a relaxed version of the emulation problem: For any arrival sequence, each packet that successfully departs the OQ switch must depart the CIOQ switch within a bounded delay. They introduce a frame-based CIOQ policy that observes the packets departing from the OQ switch in each time frame, and transfers them from the input to the output in the following frame. The proposed CIOQ policy guarantees a relative packet delay and maximum buffer occupancy at most twice the output buffer capacity ($2B$), at speedup 2. Remarkably, the result holds for any OQ algorithms, even for those with preemptive drop policies. The reason is that even if all packets buffered at some CIOQ input depart simultaneously from the emulated OQ switch,

the CIOQ policy can spread their transfer to the output side over a time frame duration (B time steps) without violating the relative delay guarantee. Although the throughput of a CIOQ switch using the frame-based policy is identical to the throughput of the emulated OQ switch and the relative packet delay is small, exact guarantees (e.g., throughput) obtained for a multihop network of OQ switches do not carry over to networks of CIOQ switches because of permitted delay. Composing approximate bounds over multiple hops leads to loose bounds, the quality of which depends on the number of hops [26]. As a result, in this work we choose to investigate the cost of exact OQ emulation.

Finally, we should note that Minkenberg [34] studied the emulation of OQ switches with finite buffers, and reported a result that appears to contradict the results in this paper and in [24]. The result states that no CIOQ policy that does not starve some input queue can be work-conserving at any speedup $< N$ (the size of the switch). Hence, no policy can emulate an OQ switch employing a work-conserving scheduling algorithm. The result is obtained by constructing an example where the number of packets present in the CIOQ switch and destined to the same output can exceed the output buffer capacity. This is in contrast to the framework considered here and in [24, 3], where the CIOQ switch immediately discards any packet that is dropped by the OQ algorithm.

5.4 ORGANIZATION

In the next chapter, we introduce some notation and definitions used throughout this part of the thesis. In Chapter 7 we present the results pertaining to the emulation of non-preemptive OQ algorithms. Results pertaining to the emulation of preemptive algorithms are presented in Chapter 8, leading to the specification and analysis of the CEH CIOQ policy in the following chapter.

6.0 SWITCH MODELS

Consider an OQ switch with N input/output ports equipped with buffer capacity $B \geq 1$ packets at every output, and a CIOQ switch of the same size and output buffer capacity. Our goal is to identify CIOQ policies for the emulation of the OQ switch. In this section, we give a precise characterization of such policies and introduce notation and definitions used in the following chapters.

A switch's input and output ports are labeled $I_1, \ldots, I_N$ and $O_1, \ldots, O_N$, respectively. Given the foreseen technological limitations on buffer capacity and the demand for switch scalability, we limit our study to the case $N \gg B$. Time proceeds in discrete steps indexed by the natural numbers. A time step is divided into three phases: the arrival, switching, and departure phases, in that order. During the arrival phase, arriving packets are received at the input ports (at most one per port), whereas in the switching phase, the switch may transfer packets from the input side to the output side across its fabric. Finally, in the departure phase each output port can transmit one packet along the attached link.

A sequence of packet arrivals σ is a non-empty finite set of triplets $\langle I, \tau, p \rangle$, each representing the arrival of a packet p at input I and time step τ.

6.1 OQ ALGORITHMS

As shown in Figure 10(a), in an OQ switch, the fabric provides a dedicated point-to-point channel between each input and output. This enables the switch to simultaneously transfer up to N packets to each output port. Given that at most N packets arrive during a time step,

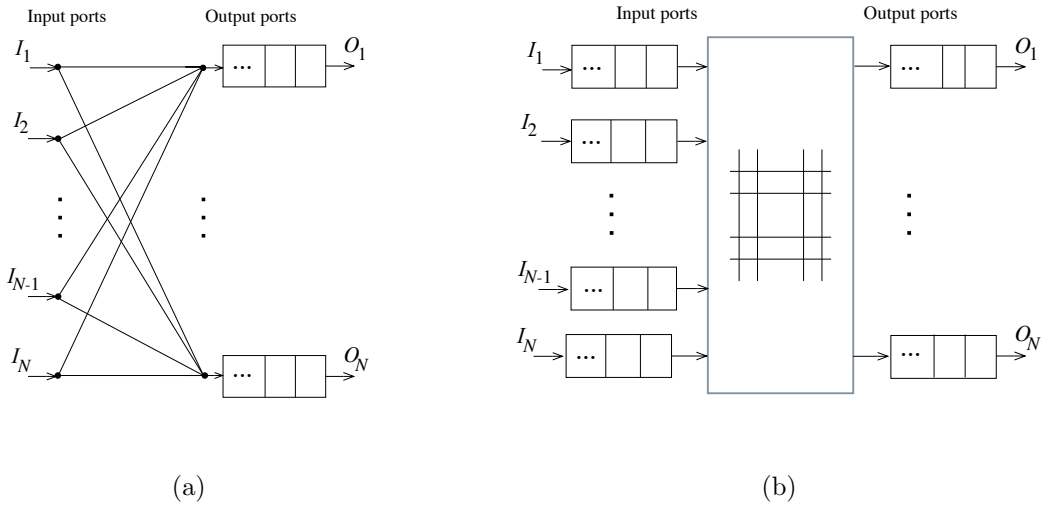


Figure 10: Switch architectures: (a) Output Queuing, and (b) Combined Input-Output Queuing.

all packets are transferred to their respective outputs in the switching phase immediately following their arrival.

At the output ports, each packet received from the input side is stored in the output buffer awaiting departure, or is dropped if no buffer space is available to store it. The output scheduling algorithm decides the departure order of packets in the buffer, and which packets are dropped in the case of overflow. For brevity, an output scheduling algorithm employed in an OQ switch is henceforth called an *OQ algorithm*.

Each output port in the OQ switch independently executes a copy of the OQ algorithm. Let σ be the arrival sequence. At any time, the *configuration of an output buffer* is the set of packets stored in the output's buffer. At the start of the departure phase of each time step t, the algorithm takes the current output configuration, and the history of packet arrivals and packet drops up to t as input, and decides which packets to drop, if any, and which packet to transmit during the departure phase. These decisions, along with any additional information (e.g., packets' queue positions in the case of FIFO-based algorithm), is called the *state of the OQ algorithm* at time t. Note that the OQ algorithm does not necessarily

arrange the packets in the buffer into a queue. It may, for example, randomly choose a packet to transmit in each step.

The sequence of packet departures given arrival sequence σ is represented by a set D_σ. Each element in the set is a triplet $\langle O, \tau, p \rangle$ denoting the departure of packet p from port O at time τ.

Within the OQ emulation framework described in Chapter 5.1, the CIOQ switch "simulates" a complete step (all three phases) of the OQ switch at the start of each CIOQ switching phase. This allows the CIOQ to keep track of the OQ algorithm's decisions. The CIOQ switch emulates the OQ switch if for every arrival sequence σ, the sequence of departures from the CIOQ switch ports is the same as the departure sequence from the emulated OQ switch, that is D_σ. This is the case if and only if, given the CIOQ speedup, the CIOQ policy transfers each packet from the input to its output in time for departure.

6.2 CIOQ POLICIES

Suppose σ is the arrival sequence at the CIOQ switch (Figure 10(a)). At the start of the switching phase of every time step t, the CIOQ policy maps the current *input configuration* (the set of packets stored at the inputs) and the current state of the OQ algorithm at each of the emulated OQ outputs to a subset of the packets available at the input ports. Packets in this subset are moved to the outputs across the CIOQ fabric during the switching phase. The choice of the packets in to move to the output in a given time step is deterministic and is subject to the speedup constraint: Given a fabric speedup $s \geq 1$, the policy must choose the packets to transfer so that at most s packets are moved from each input, and at most s packets are moved to each output in a given step.

A CIOQ policy that enables the CIOQ to emulate a given OQ algorithm is called a *valid* policy for the emulation of the algorithm.

Definition 6.1 (Valid CIOQ Policy)**.** *A CIOQ policy is valid for (the emulation of) a given OQ algorithm if, for any switch size N, output buffer capacity $B \geq 1$, and for every arrival*

sequence, it transfers the packets through the CIOQ fabric so that for every time step t, any packet that would depart from the emulated OQ switch during t is transferred to the corresponding CIOQ output before t's departure phase. A CIOQ policy is valid for a family of OQ algorithms if it is valid for every algorithm in that family.

A CIOQ policy may be valid for the emulation of an algorithm only under some restrictions. For example, only in the infinite-buffers setting where the output buffer capacity is considered unlimited.

For a given OQ algorithm, switch parameters, and arrival sequence, a valid policy is said to *meet the OQ departure time* of every packet. Valid policies for the emulation of a particular OQ algorithm (or a family thereof) may differ in the buffer capacity requirements at the CIOQ inputs and the required CIOQ speedup. A CIOQ policy that is valid at speedup s, and for which the input buffer occupancy does not exceed b under any arrival sequence, is called an (s, b)-valid CIOQ policy. It is easy to see that an (s, b)-valid policy is also (s', b')-valid for all (s', b') where $s' \geq s$ and $b' \geq b$, if at speedup s' it transfers at each time step a super-set of the packets it would transfer at speedup s.

We focus our attention on CIOQ policies that are *greedy*. A greedy policy transfers a maximal set of packets to the output in every time step. As a result, for every non-greedy CIOQ policy π and CIOQ speedup s, one can define a greedy policy π', that, at every time step transfers a super-set of the packets transferred by π. Obviously, if π is valid (for the emulation of some OQ algorithm) at speedup s, then π' is also valid at the same speedup.

The following definitions lead to a formal characterization of greedy policies, and are used in subsequent chapters:

Definition 6.2 (Input Blocking)**.** *A packet p at a CIOQ input port I is input blocked during a time step t if, during t's switching phase, the CIOQ policy transfers s packets from I to the output side, and these packets do not include p.*

Definition 6.3 (Output Blocking)**.** *A packet buffered at some input port and destined to output port O is output blocked during time step t if, during t's switching phase the CIOQ*

policy transfers s packets to output O, and these packets do not include p.

Definition 6.4 (Greedy CIOQ Policy)**.** *A CIOQ policy is greedy if at every time step, every packet buffered at an input port is either transferred to the output, is input blocked, or is output blocked.*

6.3 FAMILIES OF OQ ALGORITHMS

The objective of the OQ emulation problem is to identify CIOQ policies that are valid for the emulation of all OQ algorithms, at minimum CIOQ speedup and input buffer capacity requirements. Toward this end, we seek upper and lower bounds on the resource requirements of greedy CIOQ policies for the emulation of families of work-conserving algorithms.

Because, in the OQ switch, an output buffer can accept at most B new packets in a time step, a speedup of B is sufficient for the emulation of all work-conserving algorithms.

Proposition 6.1. *Every greedy CIOQ policy is $(B, 1)$-valid for the emulation of all work-conserving OQ algorithms.*

Proof. The result follows by induction from the observation that the input buffers are empty prior to the earliest arrivals, and at each time step, if the CIOQ input buffers are empty at the start of the arrival phase, they are also empty at the end of switching phase. □

Such speedup requirement is feasible only when B is very small (e.g., up to 5), but would be prohibitive even in high-speed packet switches with limited buffering capacity.

To obtain lower bounds on the resource requirements of greedy CIOQ policies, we consider subsets of work-conserving algorithms that include well-known and widely-used ones.

Namely, the family of algorithms with *non-preemptive* drop policies (non-preemptive algorithms) and the family of algorithms with *PIFO* service disciplines (PIFO algorithms).

The drop policy of an OQ algorithm is non-preemptive if an incoming packet may be dropped upon arrival to the OQ switch, but may not be dropped once admitted to the output buffer. Otherwise, the drop policy is preemptive. Non-preemptive drop policies are collectively referred to as "Drop Tail." These policies differ in how the tie is broken when the number of arrivals destined to an output port in a given time step exceeds the space available in that output's buffer. Possible tie-breaking rules include randomly choosing the "victim" packets among those arrivals, and tie-breaking based on input port numbers, or based on information in the packets' headers [7].

A PIFO service discipline arranges the packets in the output buffer into a queue, where:

(P1) At each time step, the packet at the head of the output queue departs the OQ switch.

(P2) An arriving packet is inserted at some arbitrary position (defined by the service discipline) in the output queue.

(P3) For each pair of packets p, q in the output queue, if p precedes q relative to the head of the queue at some time t, then this order is preserved at every subsequent step where both packets remain in the buffer.

In the absence of further packet arrivals to the output port, the position of any packet in the queue determines the time it departs from the OQ switch. We refer to this as the *projected departure time* of the packet at time t. Note that a PIFO service discipline may be paired with any drop policy (premptive or non-preemptive).

In the next chapter we investigate the speedup and input buffer capacity required by greedy CIOQ policies for the emulation of non-preemptive OQ scheduling algorithms. Em-

ulation of OQ preemptive algorithms is considered in the following chapter.

7.0 OQ EMULATION OF NON-PREEMPTIVE SCHEDULING ALGORITHMS

In this chapter, we study the emulation of non-preemptive OQ scheduling algorithms. First, we characterize a trade-off between speedup and the maximum input buffer occupancy. The trade-off applies to all greedy CIOQ policies that are valid at speedup $s > 2$. Then, we describe a greedy variant of the CCF policy introduced in [10] and show that this variant is $(2, B)$-valid for the emulation of non-preemptive PIFO OQ algorithms.

7.1 THE SPEEDUP — BUFFER CAPACITY TRADE-OFF

Theorem 7.1. *Let π be a greedy CIOQ policy that is valid for the emulation of a non-preemptive OQ algorithm A at speedup $s > 2$, with buffer capacity B at every output port. Then, the buffer occupancy at each of the CIOQ switch's inputs does not exceed $1 + \lceil \frac{B-1}{s-2} \rceil$.*

Proof. To reach contradiction, suppose that there is a CIOQ input $I_i, i \in \{1, \ldots, N\}$, with buffer occupancy exceeding $1 + \lceil \frac{B-1}{s-2} \rceil$ at some time step. Let t be the earliest such step and consider the following claim:

Claim 7.1. *Let p be packet with the earliest arrival time among those in I_i's buffer just*

after the arrival phase of time step t, and let $t-x$ be p's arrival time. Then the greedy CIOQ policy had transferred at least $x+B+1$ packets to p's output port during the interval $[t-x,t)$.

Let $O_j, j \in \{1,\ldots,N\}$, be p's output port. By Claim 7.1 (proof below), neither p nor the first $x+B$ packets transferred to O_j during $[t-x,t)$ are dropped by the non-preemptive OQ algorithm. Otherwise, these packets would have been dropped by the CIOQ upon arrival. That is, without being buffered for a complete time step at the input (as in p's case) or being transferred to the output. Since the emulated OQ output serves at most x packets during $[t-x,t)$ and the arrival sequence is the same for both the CIOQ and the emulated OQ switch, the emulated OQ output corresponding to O_j would hold more than B packets at the beginning of time step t, which contradicts the fact that the output buffer capacity of the emulated OQ switch is B packets.

To prove the claim first observe that $x \geq \lceil (B-1)/(s-2) \rceil + 1$: since an input can receive at most one new arrival in each time step, I_i's input buffer content at t has to build up over at least $\lceil \frac{B-1}{s-2} \rceil + 2$ time steps starting with p's arrival and including t. The value of x exceeds $\lceil (B-1)/(s-2) \rceil + 1$ if there are arrivals after p that are transferred to the output before time t. Suppose the number of packets that arrive at I_i in $[t-x,t)$ and are transferred to the output before t is z. Then $x-z = \lceil \frac{B-1}{s-2} \rceil + 1$. Now, observe that I_i buffers at most $\lceil (B-1)/(s-2) \rceil$ packets at the beginning of step $t-x$ (prior to p's arrival). These pakets, in addition to the z packet described above, will be transferred to the output during $[t-x,t)$, resulting in at most $T_{\mathrm{IB}} = \big(\lceil \frac{B-1}{s-2} \rceil + z \big)/s$ input-blocked steps for p during $[t-x,t)$.

Let T_{OB} denote the number of steps where p is output blocked during $[t-x, t)$. Then

$$\begin{aligned} T_{\mathrm{OB}} &= x - T_{\mathrm{IB}} \\ &\geq x - \frac{1}{s}\left(\left\lceil \frac{B-1}{s-2} \right\rceil + z \right). \end{aligned}$$

Under any greedy CIOQ policy, the number of packets transferred to p's output during the steps where p is output blocked is sT_{OB}.

$$\begin{aligned} s \cdot T_{\mathrm{OB}} &\geq x + (s-1)x - \left\lceil \frac{B-1}{s-2} \right\rceil + z \\ &= x + (s-2)\left\lceil \frac{B-1}{s-2} \right\rceil + (s-1) + (s-2)z \\ &\geq x + (B-1) + (s-1) + (s-2)z \\ &> x + B, \end{aligned}$$

where the second step is obtained using $x - z = \left\lceil \frac{B-1}{s-2} \right\rceil + 1$, and the last step follows from the restriction $s > 2$ and the fact that $z \geq 0$. □

7.2 THE CRITICAL CELLS FIRST CIOQ POLICY

In this section, we review the CCF CIOQ policy of [10] and introduce its greedy variant, G-CCF. We show that G-CCF is $(2, B)$-valid for the emulation of non-preemptive PIFO algorithms. In contrast to this result, we show in the next chapter that G-CCF is not valid for OQ emulation at any speedup less than B when preemption is allowed.

CCF and G-CCF consist of two components: The management of input buffers, and the selection of packets to transfer to the output in every step. We begin by describing the buffer management component, which is common to both policies, then specify packet selection, starting with G-CCF.

Input Buffer Management: Under both CCF and G-CCF, the input buffer is organized as a queue that permits insertion of packets at arbitrary locations and the removal of packets at arbitrary locations. Consider an arbitrary packet p and let t be its arrival time. Further, let l be the **output cushion** of p, defined as the number of packets at p's output that have earlier projected departure time than p from the emulated OQ switch (as calculated after t's arrival phase). Packet p is inserted into the input queue at position $l+1$ (from the head of the queue). If the queue has less than l packets, the arriving packet is inserted at the end of the queue.

Packet Selection in G-CCF: To choose the set of packets to transfer to the output, in each time step G-CCF computes a *many-to-many pairwise-stable matching* (details below) of input ports to output ports. For this, G-CCF uses the Gale-Shapley Deferred Acceptance algorithm [20] (also know as the stable-marriage algorithm), as adapted by Roth to the many-to-many setting [39].

Given a CIOQ speedup $s \geq 1$, each port participates with a *quota* of s packets in the many-to-many matching. That is, up to s packets at each input port are transferred to the output side and up to s packets are transferred to an output port. Matching is based on the preferences of the inputs and outputs. The output preference is represented by a list of packets and the respective inputs arranged in increasing order of the projected OQ departure time. The input preference is a list of the packets queued at the input (and their respective outputs) arranged in the same order as the input queue. A port prefers to be matched with ports that appear earlier in its preference list. In the following pseudo-code, an outstanding request for a packet is a request that the corresponding input has not already rejected.

DEFERRED-ACCEPTANCE-ALGORITHM

while there are outputs with unfilled quota and outstanding requests

do

Each such output requests its preferred packets from the inputs to fulfill its quota

Each input grants the requests it prefers without exceeding its quota

Note that in the second step of the **while** loop, an input may cancel previous grants to accept more preferred requests.

Per the definition pairwise stability [41], a matching is pairwise-stable given the G-CCF preference lists if at every time step t, for every packet p buffered at some input at the beginning of the switching phase, either:

- p is transferred to the corresponding output during t,
- s packets with earlier projected OQ departure times are transferred to p's output during t, or
- s packets ahead of p in its input queue are transferred to their corresponding outputs during t.

It follows that G-CCF is a greedy CIOQ policy (cf. Definition 6.4).[1]

Packet Selection in CCF: CCF computes s one-to-one stable matchings in every time step by repeatedly invoking the (one-to-one) Deferred Acceptance algorithm [20]. The one-to-one algorithm uses the same input and output preference lists as G-CCF. Each output can request at most 1 packet, and each input can grant at most 1 packet in an iteration of the **while** loop.

Though the resulting matchings are individually stable, one can construct an example where the iterative matching procedure in CCF fails to transfer a maximal set of packets in

[1] A pairwise stable matching is guaranteed to exist at every time step. In general terms, a pairwise stable matching exists if every *agent* has substitutable preferences. That is, the agent continues to want to partner with an agent from the other side of the market even if another agent becomes unavailable [31, 39]. In our application, agents are ports on the input and output sides of the switch. Input Ports have substitutable preferences. If any packets buffered at the input are output blocked, it continues to want to transfer the remaining packets to their respective outputs. Similarly, output ports have substitutable preferences. If any packets requested by an output port are input blocked, it continues to seek the remaining packets on its preference list.

a given time step; thus showing that CCF is not a greedy policy. The reason is that there can be an invocation of the one-to-one Deferred Acceptance algorithm where all packets requests by an output port are rejected by the corresponding inputs (each input grants a better preferred request), while more than one of these packets are not input blocked in a later iteration.

In our application, the Deferred Acceptance algorithm is $O(B \cdot N)$ regardless of the input and output quotas. Thus, at a constant speedup both G-CCF and CCF have the same worst-case complexity.

7.2.1 OQ Emulation using CCF and G-CCF

Kesselman and Rosén proved that CCF is $(2, 2B)$-valid for the emulation of the FIFO/Drop Tail algorithm [24]. The result also holds for any non-preemptive PIFO algorithms. Here, we give a similar result for G-CCF that lowers the input buffer capacity capacity required to B packets.

Lemma 7.1. *G-CCF is valid at speedup* 2 *for the emulation of any non-preemptive PIFO OQ scheduling algorithm.*

Proof. The proof is similar to [10, Lemma 1]. Consider a packet p that is not dropped upon arrival by the OQ algorithm. At any time step during which p remains at the input, the *slackness* of p is obtained by subtracting the number of packets ahead of p in the input queue from p's output cushion. Observe that by the input-buffer management rule, the slackness is at least zero upon p's insertion into the input queue. The proof proceeds by showing that at speedup 2, the slack remains non-negative. Observe that by the definition of the output

cushion, whenever p reaches its departure time, its output cushion must be zero. As a result, if p's slackness is non-negative, the number of packets ahead of p in the input queue when it reaches its departure time must also be zero. As p would be at the top of both the input and output preference lists during that step, it would be transferred to the output in time for departure.

To see that the slackness remains non-negative, observe that at each time step where p is at the input, the many-to-many Deferred Acceptance algorithm increases p's output cushion by 2, or decreases the number of packets ahead of it in the input queue by 2. Since the output cushion may also decrease by one due to a departure, and the number of packet ahead of p at the input may increase by one due to a new arrival, p's slackness either increases or remains unchanged in every step it remains at the input. □

Now, we are ready to state our main result for G-CCF.

Theorem 7.2. *For any output buffer capacity $B > 0$, G-CCF is a $(2, B)$-valid CIOQ policy for the emulation of any non-preemptive PIFO OQ algorithms.*

Proof. By Lemma 7.1, G-CCF is valid at speedup 2 for the emulation of any non-preemptive PIFO algorithm. We show that at speedup 2, for any packet p, the number of packets ahead of p in the input queue never exceeds $B - 1$.

Suppose p arrives at some input I_i, $i \in \{1, \ldots, N\}$, at t. Let l be p's output cushion upon arrival. To avoid the trivial case, we assume p is not dropped by the emulated OQ algorithm, hence $l < B$. The packet is inserted by G-CCF into some position $\ell + 1$, where $\ell \leq l$.

Consider the sequence of consecutive time steps starting with p's arrival. Since G-CCF

is a greedy policy, in each step, p is either transferred to the output, is input blocked, or is output blocked. At any time step where p remains at the input, the number of packets ahead of p in the input queue ends up with an increase of one packet only if p is output blocked. That is, if its output cushion also ends up with a net increase of one packet at the end of the time step. Furthermore, p's output cushion ends up with a decrease of one packet only if p is input blocked. That is, if the number of packets ahead of it in the input queue ends up with a decrease of at least one packet (at most one packet is inserted into the input queue ahead of p, while exactly two packets ahead of p are transferred to the output in that step).

For each step where p is output blocked its output cushion increases by exactly one, and for each step where it is input blocked its output cushion decreases by at most one, and two packets ahead of it in the queue are transferred to the output. Thus, the number of packets ahead of p in the input queue increases by 1 only if its output cushion also increases by 1 (a step where p is output blocked). Furthermore, p's output cushion decreases by 1 only if the number of packets ahead of it in the input queue decreases by at least 1 (at most one packet is inserted into the input queue ahead of p at any step).

For any time step τ, let $\underline{\tau}$ denote τ's arrival phase and $\overline{\tau}$ denote τ's switching and departure phases. Suppose the number of packets ahead of p in the input buffer reaches B for the first time at the end of the arrival phase of step $t_B > t$. Divide $[\overline{t}, \underline{t_B}]$ into $B - \ell$ intervals $[\overline{t_\ell}, \underline{t_{\ell+1}}], [\overline{t_{\ell+1}}, \underline{t_{\ell+2}}], \ldots, [\overline{t_{B-1}}, \underline{t_B}]$, where $t_\ell = t$ and t_i is the earliest time step where the number of packets ahead of p in the input buffer reaches i at the end of the arrival phase.

Since the number of packets ahead of p increases by 1 in every interval $[\overline{t_i}, \underline{t_{i+1}}]$, the number of steps where p is output blocked exceeds the number of steps where it is input

blocked by at least 1 in every such interval. It follows that at the end of the arrival phase $\underline{t_{i+1}}$, the output cushion is at least one more than it was at the end of $\underline{t_i}$. Thus, p's output cushion reaches at least B at the end of the arrival phase of time step t_B.

Let the output backlog of packet p be the total number of packets present at the CIOQ switch and destined to the same output as p. p's output cushion is strictly smaller than its output backlog since the latter accounts for p itself. Thus p's output backlog at the end of $\underline{t_B}$ exceeds B, which contradicts the fact that the emulated OQ has output buffer capacity B. □

8.0 OQ EMULATION WITH PREEMPTION ALLOWED

In this chapter we show that no greedy CIOQ policy is valid for the emulation of all OQ algorithms at speedup $s \leq \sqrt[3]{B-2}$ when preemption is allowed, and that G-CCF is not valid at any speedup $s < B$ under the same conditions.

8.1 THE SPEEDUP LOWER BOUND

Theorem 8.1. *No greedy CIOQ policy is valid for the emulation of all PIFO scheduling algorithms at any speedup $s \leq \sqrt[3]{B-2}$ when preemption is allowed in the emulated OQ algorithm, and the output buffer capacity is B.*

Proof. The theorem holds trivially for $B < 3$. For any $B \geq 3$ and $s \leq \sqrt[3]{B-2}$, we construct an example where no greedy CIOQ policy can transfer all packets to their outputs on time for departure. The example uses FIFO/Drop Front as the emulated OQ scheduling algorithm. Under FIFO/Drop Front, whenever an overflow occurs, packets at the head of the emulated FIFO queue are preempted (dropped) to make room for the new arrivals. The new arrivals are then inserted at the tail of the FIFO queue.[1]

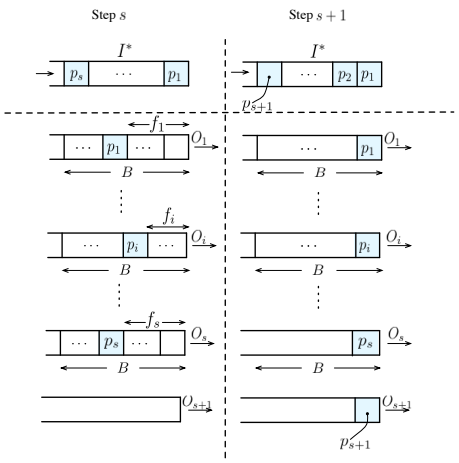
[1] In case the number of new arrivals exceeds B, only B arbitrarily chosen packets are accepted.

Starting with $t = 1$, we will specify a packet arrival sequence that leads to buffer occupancy s at some input port I^* after exactly s steps. In the following arrival phase, new arrivals will increase the buffer occupancy at I^* to $s+1$ packets (all having distinct outputs), and cause the Drop Front policy to preempt all packets ahead of those buffered at input I^* in their emulated FIFO output queues. Thus, all packets buffered at I^* become needed immediately for departure at their respective outputs. Since no more than s packets can be transferred simultaneously to the output side from the same input port, at least one of the packets among those buffered at I^* misses its OQ departure time, which completes the proof.

The arrival sequence is as follows (see Figure 11 for an illustration). At every step t in $[1, s]$, $B - 1$ packets destined to port O_t are injected into inputs I_1 through I_{B-1}. Under any CIOQ policy, in the switching phase of any step $t \in [1, s]$, at most st input ports are dequeued by the policy. This is because the inputs buffer packets destined to at most t different outputs and, given speedup s, each output can receive at most s packets in a time step. It follows that at the end of time step s, at most $s \sum_{t=1}^{s} t = \frac{s^3+s^2}{2} \leq s^3$ ports have been dequeued in one or more time steps. Conversely, among ports I_1 through I_{B-1}, at least $(B-1) - s^3$ ports are never dequeued in $[1, s]$. Given $s \leq \sqrt[3]{B-2}$, there is at least one such port. Let I^* be a port in $\{I_1, \ldots, I_{B-1}\}$ that was never dequeued during $[1, s]$. Then port I^* buffers exactly s packets at the end of step s. To keep the emulated output buffers full at every t in $[1, s]$, a packet destined to O_i is injected into port I_{B-1+i} for each output $O_i \in \{O_1, \ldots, O_t\}$. Observe that since FIFO/Drop Front is work-conserving, if all packets with OQ departure time in $[1, s]$ are transferred to the output in time for departure, the number of packets present in the switch and destined to output O_t is exactly B at the end

	O_1	O_2	$\dots$	O_t	$\dots$	O_s
I_1				1		
I_2				1		
$\vdots$				$\vdots$		
I_{B-1}				1		
I_B	1					
I_{B+1}		1				
$\vdots$			$\ddots$			
I_{B+t-1}				1		
$\vdots$						
I_N						

(a) Packets arrivals at time $t \leq s$. A packet arrival is indicated by 1 in the table position corresponding to the input port at which it arrives and its output destination.



(b) Configuration of CIOQ input I^* and the emulated output queues at the end of the arrival phase of steps s and $s+1$.

Figure 11: Illustration of the arrival sequence in the proof of Theorem 8.1.

of the arrival phase of every step in $[t, s]$, and $B-1$ packets at the end of the departure phase.

In the arrival phase of step $s+1$, packets are injected as follows. Let the s packets already buffered at I^* be denoted $p_1, \ldots, p_s$, in such a way that packet p_i is the packet destined to output O_i. Furthermore, for each $p_i \in \{p_1, \ldots, p_s\}$, let $f_i < B-1$ denote the number of packets ahead of p_i in its emulated FIFO output queue. For each $p_i \in \{p_1, \ldots p_s\}$, $f_i + 1$ packets destined to port O_i are injected into ports $I_{(i+1)B}$ through $I_{(i+1)B+f_i}$. In addition, one packet destined for output O_{s+1} is injected into port I^*. We will denote this packet by p_{s+1}.

Since at the end of (the departure phase of) step s, each of the emulated output buffers corresponding to outputs $O_i, i = 1, \ldots, s$ buffers $B-1$ packets, the f_i new arrivals will cause the Drop Front policy to preempt (drop) all the packets ahead of p_i in its emulated FIFO output queue. The new arrivals are then added to the tail of the emulated FIFO output queue. Thus, every packet $p_i, i = 1, \ldots, s$ buffered at I^* must be transferred to the output during step $s+1$ in order to meet its departure time. Packet p_{s+1} must also depart during step $s+1$ since it is the only packet present in the switch that is destined to output O_{s+1}. However, at most s packets can be transferred from port I^* to the output side during step $s+1$. Thus at least one packet misses its departure time. □

The example in the proof of Theorem 8.1 assumes $N \geq 2B^2$ input/output ports. It uses FIFO/Drop Front, which is a PIFO OQ scheduling algorithm. The Drop Front policy has been proposed for the objective of minimizing the queuing delays incurred by successfully delivered packets [44], but has also been shown to improve TCP throughput compared to Drop Tail [25].

8.2 CCF IS NOT BETTER THAN THE WORST GREEDY POLICY

Next we show that when preemption is allowed, G-CCF (hence CCF) is not valid for the emulation of all PIFO OQ algorithms at any $s < B$. To reach this result, we demonstrate using an example that G-CCF fails to emulate a variant of the FIFO/Drop Front OQ scheduling algorithm that recognizes two different classes of packets: a low-delay class, denoted as class L, and a bulk data transfer class denoted as class T. We refer to this variant as *2-class FIFO/Drop Front*. The proof exploits the fact that G-CCF favors packets with earlier projected OQ departure times in every time step, and the fact that "investing" in such packets may be futile if preemption is allowed.

In 2-class FIFO/Drop Front, each traffic class has a fixed allocation (a partition) of the emulated OQ buffer capacity. We specify the buffer allocations by a pair (B_L, B_T) where $B_L + B_T = B$. At any time, the number of class-L packets present in the buffer does not exceed B_L, and similarly for class-T. An incoming packet is inserted into the proper buffer partition based on its class. Each of the two partitions is a FIFO buffer, where Drop Front is used to resolve overflow events. In each time step, a class-T packet is served if and only if no class-L packets are present in the L-partition.[2]

Theorem 8.2. *If preemption is allowed, G-CCF is not valid for the emulation of PIFO OQ algorithms at any $s < B$.*

Proof. Consider a CIOQ switch employing the G-CCF policy to emulate the 2-class FIFO/Drop Front scheduling algorithm. The proof proceeds by specifying a sequence of packet arrivals that,

[2] As described, 2-class FIFO/Drop Front uses Complete Buffer Partitioning [29]. It is straightforward to specify a similar algorithm that allows each class to utilize any unused buffer capacity by the other, potentially extracting some statistical multiplexing gains [16]. The proof of Theorem 8.2 remains unaffected.

at speedup $s = B_L < B$, results in buffer occupancy $B_L \cdot B + 1$ at some CIOQ input port in as many steps. Without further packet arrivals, at least one packet buffered at the designated input is not transferred to the output in time for departure: Since the emulated algorithm is work-conserving, the $B_L B + 1$ packets must all depart the switch by the end of step $B_L B + B$. On the other hand, at most B_L packets can be moved from the designated input to the output side in a time step. Hence, at the end of step $B_L B + B$, exactly one packet remains buffered at the designated input, thus missing its departure time.

It remains to specify the arrival sequence (see Figure 12 for an illustration). At each time step t in $[1, B_L B]$, a T-packet destined to output O_t arrives at input I_1. Furthermore, for each packet p buffered at I_1 including the new arrival, $s = B_L$ L-packets, destined to the same output as p, are injected at s different input ports.[3] At every time step t in $[2, B_L B]$, in the emulated OQ switch, the newly arrived L-packets cause the scheduling algorithm to drop all the L-packets already in the output buffer of every output port in $\{O_1, \ldots, O_t\}$. On the other hand, in the CIOQ switch, G-CCF transfers the newly arrived L-packets to their respective outputs immediately upon arrival, where they replace the preempted packets. Given $s = B_L$, all T-packets are output blocked, thus remain buffered at the input.

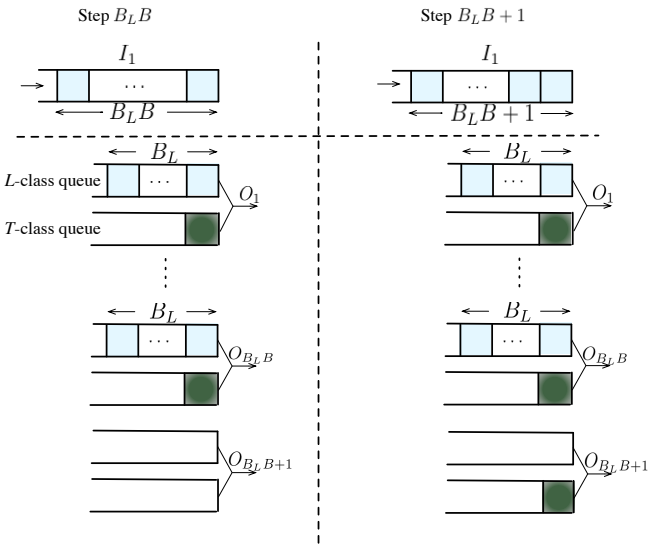
At the end of time step $B_L B$ there are as many packets buffered at each of the input ports I_1. In the following step, a T-packet arrives at port I_1 destined to output $O_{B_L B+1}$, thus raising the buffer occupancy at input $I_1$to $B_L B + 1$.

□

[3]The number of different switch ports used in this example is no more than $2B^3$. Since at $t = B_L B$ the buffer occupancy at port I_1 is at most $B_L B$, the number of different inputs used for packet injection is at most $B_L^2 B < B^3$. Each of the packets at I_1 is destined to a different output. Thus the total number of ports used by the arrival sequence up to step $B_L B$, including the outputs, does not exceed $B^3 + B_L B < 2B^3$.

	O_1	O_2	$\dots$	O_t	$\dots$	O_{B_LB}
I_1				T		
I_2	L					
$\vdots$	$\vdots$					
I_{B_L+1}	L					
I_{B_L+2}		L				
$\vdots$		$\vdots$				
I_{2B_L+1}		L				
$\vdots$			$\ddots$			
I_{tB_L+2}				L		
$\vdots$				$\vdots$		
$I_{(t+1)B_L+1}$				L		

(a) Packets arrivals at time $t \leq B_LB$. A packet arrival is indicated by T or L (depending on the packet's class) in the table position corresponding to the input port at which it arrives and its output destination.



(b) Configuration of CIOQ input I_1 and the emulated output queues at the end of the arrival phase of steps B_LB and B_LB+1.

Figure 12: Illustration of the arrival sequence in the proof of Theorem 8.2.

In constructing the example to show that G-CCF is not a valid policy at any speedup $s < B$, we exploited the fact that whenever possible, CCF transfers packets to each output in the order of their projected OQ departure times. As a consequence of preemption, some packets (the T-packets in our example) remain output blocked for an extended period of time, Thus allowing the occupancy of corresponding input buffers to build up. In the next chapter, we consider mitigating this buffer buildup at the inputs by ordering the output preference lists based on the time of packet arrival.

9.0 THE CCF-EAF HYBRID CIOQ POLICY

In constructing the example to show that CCF is not a valid policy at any speedup $s < B$ when preemption is allowed, we exploited the fact that, whenever possible, CCF transfers packets to each output in the order of their OQ departure times. As a result, some packets may remain output blocked for a relatively large number of steps, thus allowing the occupancy of corresponding input buffer to build up.

Early Arrivals First (EAF) is a CIOQ policy where every newly arrived packet is inserted at the head of the corresponding input queue. To choose the packets to transfer to the output in a given time step, EAF computes a many-to-many stable matching of input to output ports in the same way as G-CCF. However, unlike G-CCF, each output's preference list is a list of the packets buffered at the inputs and destined to that output, arranged in order of non-increasing arrival time, with ties broken based on port numbers.

As with G-CCF, a pairwise-stable matching always exists under EAF, and, assuming a CIOQ speedup $s > 1$, it is one where at every time step, for each packet p at the input, either:

- p is transferred to the corresponding output during t,
- s packets with earlier arrival times are transferred to p's output during t, or
- s packets ahead of p in its input queue are transferred to their corresponding outputs

during t.

It is easy to see that unlike CCF and G-CCF, EAF is not prone to input buffer buildup when preemption is allowed. But it is also obvious that EAF would fail to emulate a scheduling algorithm where a later arrival to the switch may have an earlier departure time; for example, OQ algorithms based on strict packet priorites or the Last-In-First-Out service discipline.

9.1 THE CEH CIOQ POLICY

Now we propose and investigate the performance of a greedy policy, CEH, which is a hybrid of CCF and EAF. Under CEH, new arrivals to the CIOQ switch are inserted at the head of the corresponding input buffer. Given CIOQ speedup $s \geq 2$, CEH chooses the packets to transfer from the input to the output by sequentially computing two pairwise stable matchings using the Deferred Acceptance algorithm (Section 7.2)

The first is a matching computed using the input and output CCF preference lists. In this matching, every output has a quota of 1 and every input has a quota of s. That is, whereas the number of packets participating in the stable matching destined to any given output does not exceed 1, an input may participate in the matching with up to s packets.

The quotas for the second matching are calculated as follows: Suppose some port P (an input or output port) participates in the first matching with $U(P)$ packets. Then, in the second matching its quota is $s - U(P)$ packets. The second matching is computed using the EAF preference lists described above.

The following lemma implies that CEH is a greedy policy. We use this result in the next section to obtain the speed required for OQ emulation using CEH.

Lemma 9.1. *At every time step t, for every packet p buffered at one of the inputs at the beginning of the arrival phase, either:*

(*i*) *p is transferred to the corresponding output, say O,*

(*ii*) *There exists a packet with earlier OQ departure time and $s-1$ packets with earlier arrival times that are transferred to output O, or*

(*iii*) *Exactly s packets ahead of p in its input queue are transferred to their corresponding outputs.*

Proof. Suppose a packet p buffered at some input I is not transferred to the output during a time step t. Then, in the first stable matching, either p does not participate in matching in favor of a packet with earlier OQ departure time, or in favor of s packets ahead of it in the input queue. Let $U(I) \leq s$ be the number of packets with which input I participates in the first stable matching. Furthermore, suppose p is destined to output O and let $U(O) \leq 1$ be defined similarly to $U(I)$.

In the second matching, ports I and O have quotas $s - U(I)$ and $s - U(O)$, respectively. Since the second matching is a stable matching where p does not participate, then either $s - U(O)$ packets destined to O with arrival times earlier than p participate in the matching, or $s - U(I)$ packets ahead of p in the input queue participate in the matching. The lemma follows from the definitions of $U(I)$ and $U(O)$. □

Now we present the main result concerning CEH.

9.2 PERFORMANCE OF CEH

Theorem 9.1. *At any speedup* $s \geq \max\{2, \left\lceil \sqrt{2(B-1)} \right\rceil\}$, *CEH is* $(s, 1 + \frac{B-1}{s-1})$*-valid for the emulation of all work-conserving OQ algorithms.*

Proof. Suppose $s \geq 2$ and consider a packet p that is not dropped by the OQ scheduling algorithm. Suppose the packet arrives at time t and departs the OQ switch at time $t' > t$.

Upon arrival, at most $B-1$ packets with earlier arrival times than p and destined to the same output are buffered at the CIOQ's input side. This is because at most B packets with a common destination can simultaneously exist in the CIOQ switch. Obviously, the number of packets buffered at the input side and have earlier arrival times than p does not increase in subsequent time steps.

At time t, p is at the head of the input queue. At any step $\tau \in [t, t')$, where p is not transferred to the output due of output blocking, at least $s-1$ packets at the input side with earlier arrival times than p participate in the second stable matching. These packets are then transferred to the output during τ. Consequently, since at the input side, there is at most $B-1$ packets with earlier arrival time than p at time t, the number of time steps in $[t, t')$ during which p is output blocked is at most $\lfloor \frac{B-1}{s-1} \rfloor$. In any step during which p is output blocked, the number of packets ahead of it in the input queue increases by at most one. Thus, the number of packets ahead of p in its input buffer is incremented by 1 at most $\lfloor \frac{B-1}{s-1} \rfloor$ times.

Suppose p is not transferred to the output until t'. During t', p cannot be output blocked since it has the earliest departure time among packets destined to its output. It follows that p is moved to the output if during t' it is not input blocked. This is the case if $s > \lfloor \frac{B-1}{s-1} \rfloor$.

That is, if $s \geq \left\lceil \sqrt{2(B-1)} \right\rceil$. Thus CEH is valid at any speedup $s \geq \left\lceil \sqrt{2(B-1)} \right\rceil$.

The buffer occupancy at any input port never exceeds $1 + \left\lfloor \frac{B-1}{s-1} \right\rfloor$ since the number of packets ahead of any packet in an input queue is incremented at most $\left\lfloor \frac{B-1}{s-1} \right\rfloor$ times, irrespective of whether it is eventually dropped or transferred to the output. □

Notice that the proof allows for newly arriving packets to preempt packets already in the switch buffers. It also doesn't make any restriction on the service discipline. In fact it allows the emulated OQ algorithm to rearrange the packet positions in the output queue at any time.

10.0 CONCLUDING REMARKS

This part of the dissertation investigated CIOQ policies for the emulation of finite-buffered OQ switches employing a work-conserving (OQ) scheduling algorithm. It showed that emulation of preemptive algorithms requires $\Omega(B^{\frac{1}{3}})$ speedup, and introduced a CIOQ policy for the emulation of any preemptive algorithm at $O(B^{\frac{1}{2}})$ speedup. This result suggests that the emulation of OQ routers with small buffers using CIOQ is practically feasible. The results are summarized in Table 1.

Table 1: Summary of OQ Emulation Results

	Speedup (s)		
	Lower Bound	Upper Bound	Required Input Buffer Capacity
PIFO/non-preemptive	2	2	$1 + \lceil \frac{B-1}{s-2} \rceil$ if $s > 2$; B if $s = 2$
PIFO/preemptive	$\sqrt[3]{B-2}$	$\sqrt{2(B-1)}$	$1 + \lfloor \frac{B-1}{s-1} \rfloor$

Closing the gap between the speedup lower bound and the upper bound due to CEH is an obvious direction for future research. Another direction is the study of OQ emulation in buffered crossbar switches (CICQ) [11, 43, 30]. The additional buffering capacity at each cross points in CICQ switches may reduce the speedup required for OQ emulation compared to CIOQ switches. Magill et al. [30] studied exact emulation of OQ switches with unlimited

buffers and fixed-size packets using CICQ switches; showing that the emulation of several specific OQ service disciplines is possible at speedup 2. Turner [43], investigated the emulation of OQ switches with unlimited buffers using asynchronous CICQ switches. Allowing packets of different sizes, he showed that the emulation of PIFO service disciplines, with bounded packet delay relative to the emulated OQ switch, is possible at speedup 2 with limited buffering at the crosspoints. It is interesting to investigate whether similar results hold for the emulation of OQ switches with finite buffers and preemptive scheduling algorithms.

BIBLIOGRAPHY

[1] W. Aiello, R. Ostrovesky, E. Kushilevitz, and A. Rosén. Dynamic routing on networks with fixed-size buffers. In *Symposium On Discrete Algorithms (SODA)*, 2003.

[2] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *ACM SIGCOMM '04*, August /September 2004.

[3] H. Attiya, D. Hay, and I. Keslassy. Packet-mode emulation of output-queued switches. In *ACM symposium on on parallel algorithms and architectures*, Jan 2006.

[4] Y. Azar and R. Zachut. Packet routing and information gathering in lines, rings and trees. *Proc. 13th Annual European Symp. on Algorithms (ESA)*, Dec 2005.

[5] N. Beheshti, Y. Ganjali, R. Rajaduray, D. Blumenthal, and N. McKeown. Buffer sizing in all-optical packet switches. In *Optical Fiber Communication*, 2006.

[6] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *J. ACM*, 48(1):13–38, 2001.

[7] B. Braden, D. Clark, and J. C. et al. RFC2309: Recommendations on queue management and congestion avoidance in the internet. *Internet RFCs*, Jan 1998.

[8] C.-S. Chang. *Performance Guarantees in Communication Networks*. Springer-Verlag, London, UK, 2000.

[9] C.-S. Chang, Y.-T. Chen, and D.-S. Lee. Constructions of optical FIFO queues. *IEEE/ACM Trans. Netw.*, 14(SI):2838–2843, 2006.

[10] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queuing with a combined input output queued switch. *IEEE Journal on Selected Areas in Communications*, 17(6):1030–1039, June 1999.

[11] S.-T. Chuang, S. Iyer, and N. Mckeown. Practical algorithms for performance guarantees in buffered crossbars. In *In IEEE INFOCOM*, 2005.

[12] J. A. Cobb, M. G. Gouda, and A. Elnahas. Time-shift scheduling: Fair scheduling of flows in high-speed networks. *IEEE/ACM Transactions on Networking*, 6(3):274–285, June 1998.

[13] M. Elhaddad, H. Iqbal, T. Znati, and R. Melhem. On minimizing the worst-case loss rate in packet-routing networks. Technical report, The University of Pittsburgh, 2007.

[14] M. Elhaddad, R. Melhem, and T. Znati. Analysis of a transmission scheduling algorithm for supporting bandwidth guarantees in bufferless networks. *ACM Sigmetrics Performance Evaluation Review*, December 2006.

[15] M. Elhaddad, R. Melhem, and T. Znati. Supporting loss guarantees in buffer-limited networks. *International Workshop on Quality of Service (IWQOS)*, June 2006.

[16] A. Elwalid, D. Mitra, and R. Wentworth. A new approach for allocating buffer and bandwidth to heterogeneous regulated traffic in an ATM node. *IEEE Journal of Selected Areas in Communications*, 13(6):1115–1127, August 1995.

[17] M. Enachescu, Y. Ganjali, A. Goel, N. McKewon, and T. Roughgarden. Routers with very small buffers. In *IEEE Infocom*, 2006.

[18] S. Floyd. Proposed modifications to RED, and other proposals for active queue management. [Online:] http://www.icir.org/floyd/red.html. (A list of AQM proposals).

[19] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, 1993.

[20] D. Gale and L. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, Jan 1962.

[21] E. Gordon and A. Rosén. Competitive weighted throughput analysis of greedy protocols on DAGs. In *PODC '05: Proceedings of the twenty-fourth annual ACM SIGACT-*

SIGOPS symposium on Principles of distributed computing, pages 227–236, New York, NY, USA, 2005. ACM Press.

[22] M. Harchol-Balter and D. Wolfe. Bounding delays in packet-routing networks. In *the 27th Annual ACM Symposium on Theory of Computing (STOC)*, May 1995.

[23] S. Iyer and N. McKeown. Analysis of the parallel packet switch architecture. *IEEE/ACM Transactions on Networking (TON*, 11(2), Apr 2003.

[24] A. Kesselman and A. Rosen. Scheduling policies for CIOQ switches. *Journal of Algorithms*, 60(1):60–83, Jul 2006.

[25] T. Lakshman, A. Neidhardt, and T. Ott. The drop from front strategy in TCP and in TCP over ATM. In *INFOCOM*, Jan 1996.

[26] J. Le Boudec and P. Thiran. *Network Calculus: A theory of deterministic queues for the Internet*. Number 2050 in LNCS. Springer Verlag, 2002.

[27] E. Leonardi, M. Mellia, M. A. Marsan, and F. Neri. Joint optimal scheduling and routing for maximum network throughput. 2005.

[28] C. Li and E. Knightly. Coordinated multihop scheduling: A framework for end-to-end services. *IEEE/ACM Trans. Netw.*, 10(6), December 2002.

[29] A. Lin and J. Silvester. Priority queueing strategies and buffer allocation protocols for traffic control at an atm integrated broadband switching system. *IEEE Journal on Selected Areas in Communications*, 9(9):1524–1536, Dec 1991.

[30] R. Magill, C. Rohrs, and R. Stevenson. Output-queued switch emulation by fabrics with limited memory. *IEEE Journal on Selected Areas in Communications*, 21(4):606–615, 2003.

[31] R. Martínez, J. Massó, A. Neme, and J. Oviedo. An algorithm to compute the full set of many-to-many stable matchings. *Mathematical Social Sciences*, Jan 2004.

[32] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy RED. In *Proc. of 7th. International Workshop on Quality of Service (IWQoS'99), London*, pages 260–262, June 1999.

[33] N. McKeown and D. Wischik. Hot Topic: Making router buffers much smaller. *SIGCOMM Comput. Commun. Rev.*, 35(3):73–74, 2005.

[34] C. Minkenberg. Work-conservingness of CIOQ packet switches with limited output buffers. *Communications Letters, IEEE*, 6(10):452– 454, 2002.

[35] M. Reisslein, K. W. Ross, and S. Rajagopal. A framework for guaranteeing statistical QoS. *IEEE/ACM Trans. Netw.*, 10(1):27–42, 2002.

[36] J. W. Roberts and J. T. Virtamo. The superposition of periodic cell arrival streams in an ATM multiplexer. *IEEE Trans. Commun.*, 39(2):298–303, Feb. 1991.

[37] A. Rosén and G. Scalosub. Rate vs. buffer size: greedy information gathering on the line. *Proceedings of the nineteenth annual ACM symposium on parallel architectures and algorithms*, Dec 2007.

[38] S. M. Ross. *Stochastic Processes*. John Wiley & Sons, Inc., second edition, 1996.

[39] A. Roth. Stability and polarization of interests in job matching. *Econometrica*, Jan 1984.

[40] V. Sivaraman, H. Elgindy, D. Moreland, and D. Ostry. Packet pacing in small buffer optical packet switched networks. *IEEE/ACM Transactions on Networking (TON*, 17(4), Aug 2009.

[41] M. Sotomayor. Three remarks on the many-to-many stable matching problem. *Mathematical Social Sciences*, Jan 1999.

[42] I. Stoica and H. Zhang. Exact emulation of an output queueing switch by a combined input output. In *International Workshop on Quality of Service*, 1998.

[43] J. S. Turner. Strong performance guarantees for asynchronous buffered crossbar scheduler. *IEEE/ACM Trans. Netw.*, 17(4):1017–1028, 2009.

[44] N. Yin, M. Hluchyj, and M. Mansfield. Implication of dropping packets from the front of a queue. *IEEE Trans. Communications*, Jan 1993.

[45] T. Znati and R. G. Melhem. Node delay assignment strategies to support end-to-end delay requirements in heterogeneous networks. *IEEE/ACM Trans. Netw.*, 12(5):879–892, 2004.