ANALYSIS OF A PACKET SWITCH WITH MEMORIES RUNNING SLOWER THAN THE LINE RATE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Sundar Iyer

May 2000

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

Nick McKeown
(Principal Advisor)

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

Balaji Prabhakar
(Co-Advisor)

Abstract

This work is motivated by the desire to build a very high speed packet-switch with extremely high line-rates. In this work, we consider building a packet-switch from multiple, lower speed packet-switches operating independently and in parallel. In particular, we consider a (perhaps obvious) parallel packet switch (PPS) architecture in which arriving traffic is demultiplexed over k identical, lower speed packet-switches, switched to the correct output port, then recombined (multiplexed) before departing from the system. Essentially, the packet-switch performs packet-by-packet load-balancing, or “inverse-multiplexing” over multiple independent packet-switches. Each lower-speed packet switch, operates at a fraction of the line-rate, R; for example, if each packet-switch operates at rate R/k no memory buffers are required to operate at the full line-rate of the system. Ideally, a PPS would share the benefits of an output-queued switch; i.e. the delay of individual packets could be precisely controlled, allowing the provision of guaranteed qualities of service.

We ask the question: Is it possible for a PPS to precisely emulate the behavior of an output-queued packet-switch with the same capacity and with the same number of ports? In chapter 3, we prove that it is theoretically possible for a PPS to emulate a FCFS output-queued packet-switch if each layer operates at a rate of approximately $2R/k$. This simple result is analogous to Clos’ theorem for a three-stage circuit switch to be strictly non-blocking. We further show that the PPS can emulate any QoS queueing discipline if each layer operates at a rate of approximately $3R/k$. In chapter 4, we show that it is possible to mimic a multicast FIFO OQ switch with each layer operating at a rate of $(2\sqrt{N}+1)R/k$, where N is the total number of ports in the PPS.

However, our result appears to require a centralized scheduling algorithm with unreasonable processing and communication complexity. And so we consider a distributed approach which maintains work conservance in chapter 5. Finally, we relax the requirements on a PPS, and show in chapter 6, that it is possible to mimic a FIFO OQ switch within a specific delay bound with no speedup. The speedup can be as low as one if we use a fixed sized buffer in the inputs and outputs of a PPS.

Acknowledgments

I would like to thank my advisor Nick McKeown for the guidance and immense support both academically and otherwise, extended to me throughout this project. I am especially grateful for his perseverance in directing my efforts in a single path. He gave me the latitude to find something to work on, and helped me remain focussed on my work.

A part of this thesis work is a joint effort with Amr Awadallah. I would like to thank Amr for the many interesting discussions we had, and also specifically for being a mentor to me during the initial phases of this project. I also benefited from his meticulous and systematic approach.

I would also like to thank my co-advisor Balaji Prabhakar with whom I had many discussions on the parallel packet switch as well as switching algorithms in general. I am very grateful especially for the many conversations which were done in an informal yet challenging vein.

This work benefited greatly from various inputs given by Pankaj Gupta. Pankaj also helped verify some of the proofs in this report.

Lastly, I would like to thank all members of the research group with whom it was a great pleasure interacting.

Contents

CHAPTER 1

Introduction

1 Introduction

Wavelength division multiplexing (WDM) is making available long-haul fiber-optic links with very high capacity. WDM makes this possible by allowing a single fiber to contain multiple separate channels; today each channel typically operates at OC48c (2.5Gb/s), OC192c (10Gb/s) and in some systems at OC768c (40Gb/s). The packets or cells carried on each WDM channel are switched, or routed, by packet-switches (e.g. ATM switches, Frame Relay switches and IP routers) that process and then switch packets between different channels. It would be desirable to process packets in the optical domain, without conversion to electronic form. However, all packet-switches need buffering (by definition), and it is not economically feasible today to store packets optically. And so packet-switches will continue to use electronic buffer memories for some time to come.

But at the data rates anticipated for individual WDM channels, we may not be able to buffer packets at the speed at which they arrive and depart. For example, if a memory is 512-bits wide,[1] and buffers packets for a 160Gb/s WDM channel, the memory needs to perform a read or write operation every 1.6ns. This is impractical today; and unfortunately the time to perform a random access into an affordable memory is decreasing only slowly with time.

[1]. 512-bits, or 64-bytes is about the maximum that is practical or efficient because of ATM cell size and average packet sizes.

2 Motivation

It is the overall goal of this work to design a high capacity packet-switch (e.g. multiple terabits/second) that: (1) Supports individual line-rates in excess of the speeds of available electronic memory, and (2) Is capable of supporting the same qualities of service as an output-queued switch. These two goals cannot be realized alone by a conventional output-queued (OQ) switch; this is because OQ switches require buffer memory that operates at N times the line-rate, where N is the number of ports of the switch. This certainly doesn't meet our goal of memory running *slower* than any individual line-rate.

Likewise, we can't use the other widely used techniques for reducing memory bandwidth; namely, input-queued (IQ) and combined input and output queued (CIOQ) switches. In an input-queued switch each memory operates at the same speed as the external line-rate. While an improvement over OQ switches neither of our goals are met: (1) An IQ switch does not meet our requirement to use memories slower than the external line-rate, and (2) IQ switches are unable to provide the same QoS guarantees as an OQ switch. Because of the limited QoS capabilities of IQ switches, CIOQ switches were studied and recently, it was shown that a variety of qualities of service are possible in a CIOQ switch in which the memory operates at *twice* the line-rate [1]. Obviously, this doesn't meet our goal for memory speed.

We would like an architecture that overcomes these limitations, yet is practical. To this end, we consider here a parallel packet-switch (PPS) architecture comprised of multiple identical lower-speed packet-switches operating independently and in parallel. An incoming stream of packets is spread, packet-by-packet, by a demultiplexor across the slower packet-switches, then recombined by a multiplexor at the output. As seen by an arriving packet, a PPS is a single-stage packet-switch; all of the buffering is contained in the slower packet-switches, and hence our first goal is met because no buffers in a PPS need run as fast as the external line-rate. The demultiplexor selects an internal packet-switch (or "*layer*") and sends the arriving packet to that layer, where it is queued awaiting its departure time. When the packet's departure time arrives, the multiplexor requests that the packet be removed from its queue, and places the packet on the outgoing line.

It is an important characteristic of a PPS that the multiplexor and demultiplexor functions contain little or no buffering. However, they must make intelligent decisions; and as we shall see, the precise nature of the demultiplexing ("spreading") and multiplexing functions are key to the operation of the PPS.

3 Problem Statement

We are interested in the question: Can we select the demultiplexing and multiplexing functions so that a PPS can precisely emulate, or mimic, the behavior of an output-queued switch? Two different switches are said to *mimic* each other, if under identical inputs, identical packets depart from each switch at the same time

[2]. We shall later relax this condition of mimicking an output queued switch. If it is possible for a PPS to mimic an output-queued switch, it will be possible to control delay of individual packets and therefore provide QoS. In this work it is proved that a PPS can precisely emulate an output-queued switch which provides delay guarantees. Furthermore, each layer of the PPS may consist of a single CIOQ switch with memories operating slower than the rate of the external line.

We are not aware of any literature on the PPS architecture, but the architecture itself is not novel; “load-balancing” and “inverse-multiplexing” systems [3][4] have been around for some time, and this architecture is a simple extension of these ideas. There is similar work, which studied inverse ATM multiplexing and how to use sequence numbers to re-synchronize cells sent through parallel switches or links [5][6][7][8]. However, we are not aware of any analytical studies of the PPS architecture. As we shall see, there is an interesting analogy between the (buffered) PPS architecture and the (unbuffered) Clos Network [9].

4 Terminology

The following terminology is used throughout this report.

Cell: A fixed-length packet; not necessarily equal in length to a 53-byte ATM cell. For the purposes of this paper, although packets arriving to the switch may have variable length, we will assume that they are processed and buffered internally as fixed length “cells”. This is common practice in high performance routers; variable length packets are segmented into cells as they arrive, carried across the switch as cells, and reassembled back into packets before they depart.

Time slot: Refers to the time taken to transmit or receive a fixed length cell at a link rate of R.

Output-Queued (OQ) Switch: A switch in which arriving packets are placed immediately in queues at the output, where they contend with packets destined to the same output waiting their turn to depart. The departure order may simply be first-come first-served (FCFS) in which case we call it a FCFS-OQ switch. Other service disciplines, such as WFQ [10], GPS [11], Virtual Clock [12], and DRR [13] are widely used to provide QoS guarantees. A characteristic of an OQ switch is that the buffer memory must be able to accept (write) N new cells per time-slot where N is the number of ports, and read one cell per cell time. Hence, the memory must operate at $N + 1$ times the line-rate.

Input-Queued (IQ) Switch: A switch in which arriving cells are queued at the input, where they contend with packets waiting to go to any output. The service discipline is determined by a switch scheduler, or arbiter, which removes at most one cell per time-slot from each input, delivers it across a (usually non-blocking) switch-fabric and onto the outgoing line. A characteristic of an IQ switch is that the buffer memory need only write and read one new cell per time-slot. Hence, the memory must operate at twice the line-rate.

Work-conserving: A system is said to be work-conserving if its outputs never idle unnecessarily. In the context of a packet-switch, this means that an output never idles when there is a cell in the buffers of the packet-switch destined to that output. A consequence of a system being work-conserving is that the throughput is maximized, and the average latency of cells is minimized. An input-queued switch is not, in general, work-conserving because a cell can be held at an input queue while its output idles. An output-queued switch is work-conserving, and so a necessary, but not sufficient, condition for a switch to mimic output-queueing is that it be work-conserving.

PIFO Queues: A "Push-In First-Out" queue ordering is defined according to the following rules:

1. Arriving cells are placed at (or, "push-in" to) an arbitrary location in the queue,
2. The relative ordering of cells in the queue does not change once cells are in the queue, i.e. cells in the queue cannot switch places, and
3. Cells may be selected to depart from the queue only from the head of line.

PIFO queues are quite general and can be used to implement QoS scheduling disciplines such as WFQ, GPS and strict priorities.

5 Thesis Outline

Chapter 2 introduces the architecture of the parallel packet switch. A simple traffic pattern is described which proves that a PPS without speedup cannot mimic an OQ switch. In chapter 3 it is proved that a speedup of two suffices to mimic a FCFS-OQ switch. We also consider the problem of supporting qualities of service on a PPS and show that a speedup of three suffices for any PIFO queueing strategy. In chapter 4, multicast traffic patterns are considered. It is shown that the PPS with a speedup of $2\sqrt{N}+1$ can mimic a FCFS-OQ switch with multicast traffic. Since the methods introduced above required a centralized scheduler, we consider a distributed approach in chapter 5. We prove that a PPS can be work conserving using a distributed algorithm using a speedup of $\sqrt{N}$. In the next chapter we consider a more practical solution and show how a distributed algorithm can mimic an OQ switch within a specific delay bound. We use buffering to further reduce the speedup required in the PPS. In chapter 7, we conclude with remarks on the feasibility of the PPS and the approach.

CHAPTER 2

The Architecture of the Parallel Packet Switch

1 Introduction

In this chapter we focus on the specific type of PPS illustrated in Figure 2.1 in which the center-stage switches are OQ. The figure shows a 4×4 PPS, with each port operating at rate R. Each port is connected to all three output-queued switches (we will refer to the center-stage switches as “layers”). When a cell arrives at an input port, the demultiplexor selects a layer to send the cell to; the demultiplexor makes its choice of layer using a policy that we will describe later. Since the cells from each external input, of line rate R**,** are spread (“demultiplexed”) over k links, each input link must run at a speed of at least R/k, otherwise buffering (operating at the line-rate) would be required in the demultiplexor. Each of the layers receive cells from the N input ports, then switches each cell to its output port. During times of congestion, cells are stored in the output-queues of the center-stage, waiting for the line to the multiplexor to become available. When the line is available, the multiplexor selects a cell from among the corresponding k output queues in each layer. Since each multiplexor receives cells from k output queues, the queues must operate at a speed of at least R/k to keep the external line busy.

Externally the switch appears as an $N \times N$ switch with each port operating at rate R. Note that neither the muliplexor nor the demultiplexor contain any memory, and that they are the only components running at rate R.

We can compare the memory-bandwidth requirements of an $N \times N$ parallel packet-switch with those for an OQ switch with the same aggregate bandwidth. In an OQ switch, the memory bandwidth must be at least $(N+1)R$, and in a PPS at least $(N+1)R/k$. But we can reduce the memory bandwidth further using a

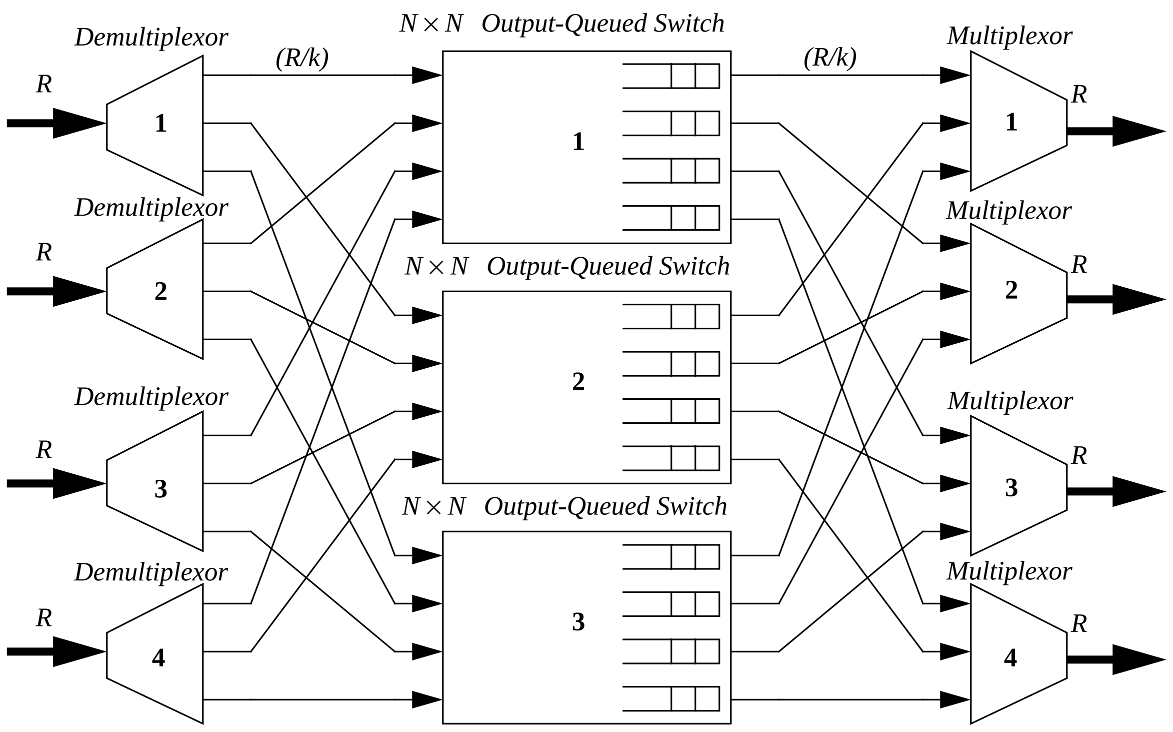


Figure 2.1: The architecture of a Parallel Packet Switch based on output-queued switches.

CIOQ switch. From [1], we know that an OQ switch can be mimicked precisely by a CIOQ switch operating at a speedup of two. So, we can replace each of the output-queued switches in the PPS with a CIOQ switch, without any change in operation. The memory bandwidth in the PPS is reduced to $3R/k$, (one read operation and two write operations per cell time) which is independent of N, and may be reduced arbitrarily by increasing k, the number of layers.

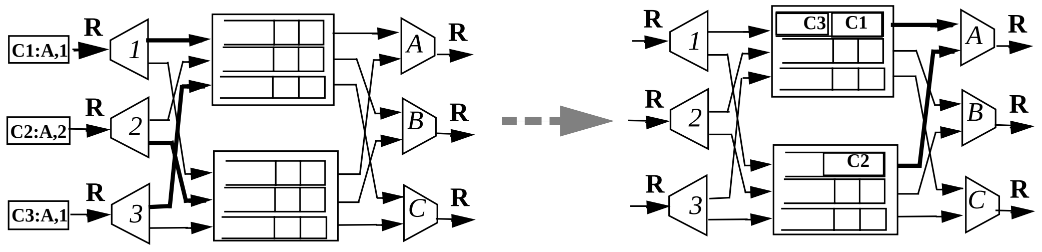
2 The Need for Speedup

It is tempting to assume that, because each layer is output queued, it is possible for the PPS described above to perform identically to an OQ switch. This is actually not the case unless we use speedup. To see why, we demonstrate that without speedup a PPS is not work-conserving, and hence cannot mimic an OQ switch.

Consider the PPS in Figure 2.2 with three ports and two layers ($N = 3$ and $k = 2$). The external lines operate at rate R, and the internal lines at rate $R/2$.

Assume that the switch is empty at time $t = 0$, and that three cells arrive, one to each input port, and all

(a) Cells arriving at time slot 0 and being sent to the middle stage switches



(b) Cells arriving at time slot 1 and being sent to the middle stage switches

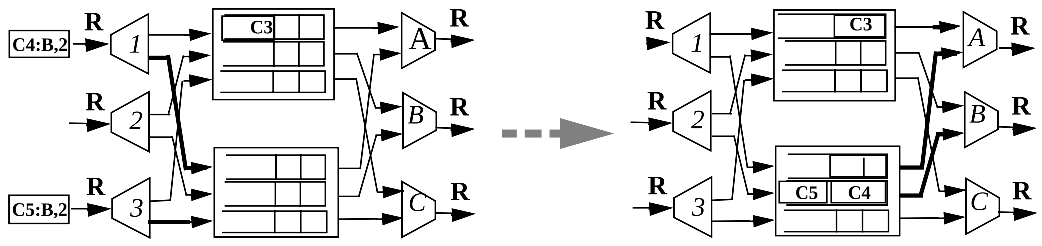


Figure 2.2: A $3x3$ PPS with an arrival pattern that makes it non work-conserving. The notation Ci :A, m denotes a cell nun i , destined to output port A , and sent to layer m .

destined to output port A . At least two of these inputs will choose the same layer. Let inputs 1 and 3 both choose layer 1 and send cells $C1$ and $C3$ to layer 1 in the first time slot. This is shown in Figure 2.2a. Input port 2 sends cell $C2$ to layer 2. These cells are shown in the output queues of the internal switches and await departure. In the second time slot, both the input ports which sent cells to the same layer in the first time slot, receive cells destined to output port B . As shown in the figure, cells $C4$ and $C5$ arrive at input ports 1 and 3 and they both must be sent to layer 2; this is because the internal line rate between the demultiplexor and each layer is only $R/2$, limiting a cell to be sent over this link only once every other time-slot. Now the problem becomes apparent: cells $C4$ and $C5$ are in the same layer, and they are the only cells in the system destined for output port B at time slot 1 . These two cells cannot be sent back-to-back in consecutive time-slots, because the link between the layer and the multiplexor operates only at rate $R/2$. So, cell $C4$ will be sent, followed by an idle time-slot at output port B , and the system is no longer work-conserving. Hence, the system cannot mimic an OQ switch.

Definition 2.1: ***Concentration:*** *Concentration is a term we will use to describe the situation when a disproportionately large number of cells destined to the same output are concentrated on a small number of the internal layers.*

Concentration is undesirable as it leads to unnecessary idling because of the limited line-rate between each layer and the multiplexor. One way to alleviate the effect of concentration is to use faster internal links. In general, we will use internal links that operate at a rate $S(R/k)$, where S is the *speedup* of the internal link.

For example, in our counter-example above, the problem could be eliminated by running the internal links at a rate of R instead of $R/2$ (i.e. a speedup of two). This solves the problem because the external output port can now read the cells back-to-back from layer two. But this appears to defeat the purpose of operating the internal layers slower than the external line rate. Fortunately, we will see in the next chapter that the speedup required to eliminate the problem of concentration is independent of the arriving traffic, R, N and is almost independent of k. In particular, we find that with a speedup of $2k/(k+2) \approx 2$, for large *k*, the PPS is work-conserving and can precisely mimic a FCFS-OQ switch.

3 Link Constraints

The operation of a PPS is limited by two constraints. We call these the Input Link Constraint and the Output Link Constraint as defined below.

Definition 2.2: ***Input Link Constraint-*** *An external input port is constrained to send a cell to a specific layer at most once every $\lceil k/S \rceil$ time slots. This is because the internal input links operate S/k times slower than the external input links. We call this constraint the input link constraint, or ILC.*

Definition 2.3: ***Allowable Input Link Set-*** *The ILC gives rise to the allowable input link set, AIL(i,n), which is the set of layers to which external input port i can start sending a cell in time slot n. This is the set of layers that external input i has not started sending any cells within the last $\lceil k/S \rceil - 1$ time slots. Note that $|AIL(i,n)| \le k, \forall (i,n)$.*

$AIL(i,n)$ evolves over time, with at most one new layer being added to, and at most one layer being deleted from the set in each time slot. If external input i starts sending a cell to layer l at time slot n, then layer l is removed from $AIL(i,n)$. The layer is added back to the set when it becomes free at time $n + \lceil k/S \rceil$.

Definition 2.4: ***Output Link Constraint-*** *In a similar manner to the ILC, a layer is constrained to send a cell to an external output port at most once every $\lceil k/S \rceil$ time slots. This is because the internal output links operate S/k times slower than the external output links. Hence, in every time slot an external output port may not be able to receive cells from certain layers. This constraint is called the output link constraint, or OLC.*

Definition 2.5: ***Departure Time***- *When a cell arrives, the demultiplexor selects a departure time for the cell. A cell arriving to input i at time slot n and destined to output j is assigned the departure time $DT(n, i, j)$. The departure time could, for example, be the first time that output j is free and able to send the cell. As we shall see later, other definitions are possible.*

Definition 2.6: ***Available Output Link Set***- *The OLC gives rise to the available output link set $AOL(j, DT(n, i, j))$, which is the set of layers that can send a cell to external output j at time slot $DT(n, i, j)$ in the future. $AOL(j, DT(n, i, j))$ is the set of layers that have not started sending any cells to external output j in the last $\lceil k/S \rceil - 1$ time slots before time slot $DT(n, i, j)$. Note that, since there are a total of k layers, $|AOL(j, DT(n, i, j))| \le k, \forall (j, DT(n, i, j))$.*

Like $AIL(i, n)$, $AOL(j, DT(n, i, j))$ can increase or decrease by at most one layer per time slot; i.e. if a layer l starts to send a cell to output j at time slot $DT(n, i, j)$, the layer is deleted from $AOL(j, DT(n, i, j))$ and then will be added to the set again when the layer becomes free at time $DT(n, i, j) + \lceil k/S \rceil$. However, whenever a layer is deleted from the set, the index $DT(n, i, j)$ is incremented. Because in a single time slot up to N cells may arrive at the PPS for the same external output, $AOL(j, DT(n, i, j))$ may change up to N times per time slot. This is because $AOL(j, DT(n, i, j))$ represents the layers available for use at some time $DT(n, i, j)$ in the future. As each arriving cell is sent to a layer, a link to its external output is reserved for some time in the future. So effectively, $AOL(j, DT(n, i, j))$ indicates the schedule of future departures for output j, and at any instant, $Max(DT(n, i, j)) + 1, \forall (n, i)$ indicates the first time in the future that output j will be free.

CHAPTER 3

Mimicking an Output Queued Switch

1 Introduction

In this chapter we shall prove the sufficiency conditions for a PPS to mimic an OQ switch. Initially we shall limit ourselves to mimicking an OQ switch with a FCFS queueing discipline. We shall then extend these results to consider a broad class of queueing disciplines.

2 Mimicking a FCFS-OQ Switch

2.1 Lower bounds on the size of the link constraint sets

The following two lemmas will be used shortly to demonstrate the conditions under which a PPS can mimic a FCFS-OQ switch.

Lemma 3.1: *The size of the available input link set,* $|AIL(i, n)| \geq k - \lceil k/S \rceil + 1$*, for all* $i, n \geq 0$*; where* S *is the speedup on the internal input links.*

Consider external input port i. The only layers that i cannot send a cell to are those which were used in the last $\lceil k/S \rceil - 1$ time slots. (The layer which was used $\lceil k/S \rceil$ time slots ago is now free to be used again). $|AIL(i, n)|$ is minimized when a cell arrives to the external input port in each of the previous $\lceil k/S \rceil - 1$ time slots, hence $|AIL(i, n)| \geq k - (\lceil k/S \rceil - 1) = k - \lceil k/S \rceil + 1$. ❐

Lemma 3.2: *The size of the available output link set,* $|AOL(j, DT(n, i, j))| \geq k - \lceil k/S \rceil + 1$*, for all* $i, j, n \geq 0$.

The proof is similar to Lemma 3.1. We consider an external output port which reads cells from the internal switches instead of an external input port which writes cells to the internal switches. ❐

Theorem 3.1:*(Sufficiency) If a PPS guarantees that each arriving cell is allocated to a layer* l*, such that* $l \in AIL(i, n)$ *and* $l \in AOL(j, DT(n, i, j))$*, (i.e. if it meets both the ILC and the OLC) then the switch is work-conserving.*

Proof: Consider a cell C that arrives to external input port i at time slot n and destined for output port j. The demultiplexor chooses a layer l that meets both the ILC and the OLC; i.e. $l \in \{AIL(i, n) \cap AOL(j, DT(n, i, j))\}$, where $DT(n, i, j)$ is the index of $AOL(.)$ and represents the first time that external output j is free in the future at time slot n. Since the ILC is met, C is sent to layer l immediately without buffering. C is placed in output queue j of layer l where it awaits its turn to depart. In fact, the departure time of the cell $DT(n, i, j)$ has already been picked when it arrived at time n. C is removed from its queue at $DT(n, i, j)$ and sent to external output port j. The reason that C departs at $DT(n, i, j)$ is because, by definition of $AOL(j, DT(n, i, j))$, external port j was busy receiving cells from other layers up until the time C departs, and hence the output was never idle when there was a cell in the system destined to it. ❐

Theorem 3.2: *(Sufficiency) A speedup of* $S > 2k/(k+2)$ *is sufficient for a PPS to meet both the input and output link constraints for every cell.*

For the ILC and OLC to be met, it suffices to show that there will always exist a layer l such that $l \in \{AIL(i, n) \cap AOL(j, DT(n, i, j))\}$, i.e. that $AIL(i, n) \cap (AOL(j, DT(n, i, j))) \neq \varnothing$, which must be satisfied if $|AIL(i, n)| + |AOL(j, DT(n, i, j))| > k$. From Lemma 3.1 and Lemma 3.2 we know that $|AIL(i, n)| + |AOL(j, DT(n, i, j))| > k$ if $S > 2k/(k+2)$.[1] ❐

Corollary 3.1: *A PPS can be work conserving, if* $S > 2k/(k+2)$.

Having shown that a PPS can be work-conserving, we now show that with the same speedup, the switch can mimic a FCFS-OQ switch.

Theorem 3.3: *(Sufficiency) A PPS can mimic a FCFS-OQ switch with a speedup of* $S > 2k/(k+2)$.

Proof: Consider a PPS with a speedup of $S > 2k/(k+2)$ which, for each arriving cell, selects a layer that meets both the ILC and the OLC. A cell destined to output j and arriving at time slot n is scheduled to depart at time slot $DT(n, i, j)$, which is the index of $AOL(j, DT(n, i, j))$. By definition of $AOL(j, DT(n, i, j))$, $DT(n, i, j)$ is the first time in the future that output j is idle, and so it is also the time

[1] In these proofs we consider only integral values of speedup.

that the cell would depart in a FCFS-OQ switch. Therefore, each cell departs at the same time that it would in a FCFS-OQ switch, and hence the PPS mimics its behavior. ❒

A detailed description of the "centralized parallel packet switch algorithm" (CPA), suggested by this proof appears in Appendix 1. It is interesting to note that the above theorem is in some ways analogous to the requirements for a 3-stage Clos network to be strictly non-blocking. In fact, the proof is quite similar. Yet the two systems are clearly different — a PPS buffers cells as they traverse the switch, while a Clos network is a bufferless fabric used mostly in circuit-switches.

3 Providing Quality of Service Guarantees

3.1 Introduction

Quality of Service (QoS) is an important requirement in switches. There exist a number of QoS disciplines. The simplest amongst them is the FCFS queueing service, which has been described in the previous section. Other queueing disciplines include strict priorities, GPS, WFQ etc.

Enforcing a QoS discipline might involve (amongst other things) determining both the correct order and time in which cells destined to an output should depart. Once the correct sequence of departure of cells has been determined, the cells must actually be switched out at their respective departure times. In a switching environment, there are a number of places where this decision can be made. Specifically this can be determined at the inputs, the outputs or by a scheduler within the switch itself.

In this chapter we make the assumption that this sequence of departure for each output is determined by a scheduler within the switch. This decision is then conveyed to the inputs. The PPS then has the onus of supporting the specific order of departure as dictated by the scheduler and the queueing discipline. Also, we shall only consider the class of queueing disciplines which conform to a PIFO order.

Definition 3.1: ***PIFO Queues****: A "Push-In First-Out" queue ordering is defined according to the following rules:*

1. Arriving cells are placed at (or, "push-in" to) an arbitrary location in the queue,
2. The relative ordering of cells in the queue does not change once cells are in the queue, i.e. cells in the queue cannot switch places, and
3. Cells may be selected to depart from the queue only from the head of line.

PIFO queues are quite general and can be used to implement QoS scheduling disciplines such as WFQ, GPS and strict priorities. In the next section we prove that a speedup of three is sufficient to mimic an OQ switch with any PIFO queueing discipline.

4 Mimicking a PIFO OQ Switch

We now extend our results to find the speedup required for a PPS to provide the same QoS guarantees as an OQ switch.

Theorem 3.4: *(Sufficiency) A PPS can exactly mimic any OQ switch with a PIFO queueing discipline with a speedup of* $S > 3k/(k+3)$.

Proof: As shown in Definition 3.1 a PIFO queueing policy can insert a cell anywhere in its queue but it can not change the relative ordering of cells once they are in the queue. Consider a cell C that arrives to external input port i at time slot n and destined for output port j. The demultiplexor will determine the time that each arriving cell must depart, $DT(n, i, j)$ to meet its delay guarantee. The decision made by the demultiplexor at input i amounts to selecting a layer so that the cell may depart on time. Notice that this is very similar to the previous section in which cells departed in FCFS order requiring only that a cell depart the first time that its output is free after the cell arrives. The difference here is that $DT(n, i, j)$ may be selected to be ahead of cells already scheduled to depart from output j. The demultiplexor's choice of layer l to send an arriving cell C must meet three constraints:

1. The link connecting layer l to output j must be free at $DT(n, i, j)$. i.e. $l \in \{AOL(j, DT(n, i, j))\}$.

2. All the other cells destined to output j after C must also find a link available. In other words, if the demultiplexor picks layer l for cell C, it needs to ensure that no other cell requires the link from l to output j within the next $(\lceil k/S \rceil - 1)$ time slots. Since the cells following cell C have already been sent to specific layers, it is necessary that the layer l being chosen be distinct from the layers which the next $\lceil k/S \rceil - 1$ cells use. Formally we can write this constraint as $l \in \{AOL(j, DT(n, i, j) + \lceil k/S \rceil - 1)\}$

3. The link connecting the demultiplexor at input i to layer l must be free at time slot n. Hence $l \in \{AIL(i, n)\}$.

Thus layer l must meet all the following constraints i.e.

$$l \in \{AIL(i, n) \cap AOL(j, DT(n, i, j)) \cap AOL(j, DT(n, i, j) + \lceil k/S \rceil - 1)\}$$

For a layer l to exist,

$$AIL(i, n) \cap AOL(j, DT(n, i, j)) \cap AOL(j, DT(n, i, j) + \lceil k/S \rceil - 1) \neq \emptyset$$

This will be satisfied when,

$$|AIL(i, n)| + |AOL(j, DT(n, i, j))| + |AOL(j, DT(n, i, j) + \lceil k/S \rceil - 1)| > 2k$$

From Lemma [1] and [2] we know that,

$$|AIL(i, n)| + |AOL(j, DT(n, i, j))| + |AOL(j, DT(n, i, j) + \lceil k/S \rceil - 1)| > 2k$$

if,

$$S > 3k/(k+3)\ . \ \square$$

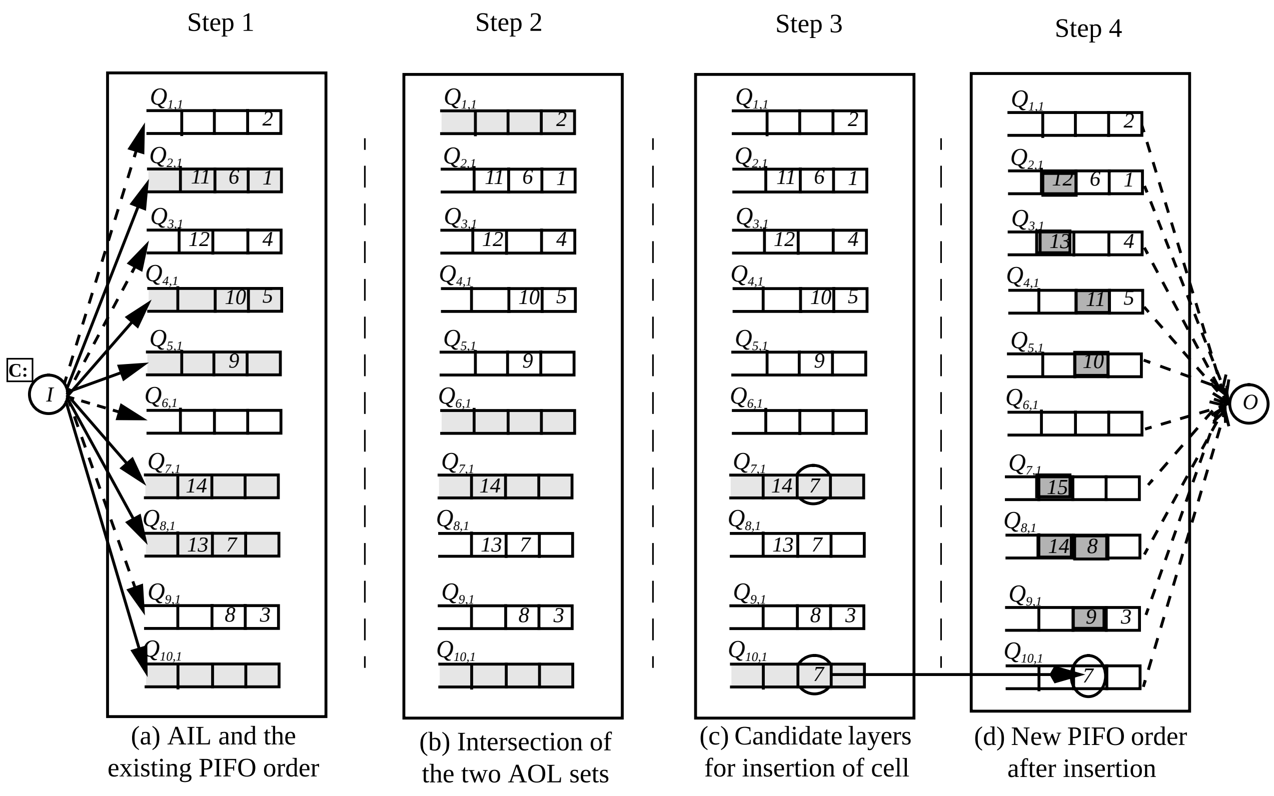

Step 1
Step 2
Step 3
Step 4
C:
I
O
Q1,1
Q2,1
Q3,1
Q4,1
Q5,1
Q6,1
Q7,1
Q8,1
Q9,1
Q10,1
(a) AIL and the existing PIFO order
(b) Intersection of the two AOL sets
(c) Candidate layers for insertion of cell
(d) New PIFO order after insertion

Figure 3.3: Insertion of cells in a PIFO order in a PPS with ten layers. $Q_{k,1}$ refers to output queue number one in the internal switch k. (a) The AIL constrains the use of layers $\{2, 4, 5, 7, 8, 10\}$. (b) The cell is to be inserted before cell number 7. The two AOLs constrain the use of layers $\{1, 6, 7, 10\}$ (c) The intersection constrains the use of layers {7,10}. (d) Layer 10 is chosen. The cell number 7 is inserted.

Figure 3.3 shows an example of a PPS with $k = 10$ layers and $S = 3$. A new cell C arrives destined to output 1 and has to be inserted in the priority queue for output 1 which is maintained in a PIFO manner. Figure 3.3a shows that the cell is constrained by the AIL to be sent to layers $\{2, 4, 5, 7, 8, 10\}$. These layers are shown darkened in Figure 3.3a. It is decided that cell C must be inserted between $C6$ and $C7$. Figure 3.3b shows the intersection of the two AOL sets for this insertion. Cell C is constrained by the AOL to use layers $\{1, 6, 7, 10\}$. Figure 3.3c shows the candidate layers for insertion i.e. layers 7 and 10. Cell C is then inserted in layer 10.

CHAPTER 4

Multicasting in a Parallel Packet Switch

1 Introduction

In the previous chapter we have analyzed the working of a PPS with respect to unicast cells. In this chapter we introduce multicast traffic. Multicast cells are cells destined to one or more outputs. We prove here the conditions under which a PPS can mimic a multicast FCFS-OQ switch.

2 Terminology

Fanout: A multicast cell can be destined to one or more outputs. The number of outputs that a multicast cell C is destined to is called the fanout of the multicast cell. We denote the fanout of a multicast cell C by m_C.

Maximum Fanout: The maximum number of outputs which any multicast cell can be destined to is called the maximum fanout. For an input traffic pattern which contains broadcast cells the maximum fanout is N. In the case when the input traffic pattern is restricted from sending cells to more than m outputs, we will denote the maximum fanout by m, where $m \in \{1, \ldots, N\}$.

Copy Multicast: A multicast cell C, which has a fanout of m_C can be divided into m_C unicast cells each destined to one of the m_C outputs. In a copy multicast, each multicast cell is copied to create multiple unicast cells at the input of a packet switch, then each unicast cell is sent through the switch. The switch as such does not handle multicast cells.

Fanout Multicast: In fanout multicast, the switch takes care of sending each multicast cell to all of its respective outputs by internally copying the cells.

3 The Speedup Required for Copy and Fanout Multicast

In this section we shall show the conditions under which a PPS will mimic a FCFS-OQ switch and an OQ switch with a PIFO queueing discipline for multicast traffic patterns.

3.1 Copy multicast

Lemma 4.1: *(Sufficiency) A PPS, with a maximum fanout of* m*, can exactly mimic a FCFS-OQ switch with a speedup of* $S > m\lceil 2k/(k+2)\rceil$*, using copy multicasting.*

Proof: Since the maximum fanout of a multicast cell is m, each cell is divided into a maximum of m parts. Thus each input of the PPS can be considered to be operating at a line rate of mR. From Theorem 3.3, we have that a speedup of $S > 2k/(k+2)$ is sufficient for a PPS to mimic an FCFS-OQ switch which has inputs receiving unicast cells at a line rate of mR. Hence a speedup of $S > m\lceil 2k/(k+2)\rceil$ suffices to mimic an FCFS-OQ multicast switch with a maximum fanout of m.❒

Lemma 4.2: *(Sufficiency) A PPS, which has a maximum fanout of* m*, can exactly mimic any OQ switch with a PIFO queueing discipline with a speedup of* $S > m\lceil 3k/(k+3)\rceil$.

Proof: The proof is along the similar lines as above.❒

3.2 Fanout multicast

Lemma 4.3: *(Sufficiency) A PPS, which has a maximum fanout of* m*, can exactly mimic a FCFS-OQ switch with a speedup of* $S > (m+1)$ *, using fanout multicasting.*

Proof: Since the maximum fanout of a multicast cell is m, each multicast cell is destined to a maximum of m outputs. Consider a cell C that arrives at external input port i at time slot n and destined for output ports $\langle P_j\rangle$, where, $j \in \{1, \dots, m\}$. For the ILC and OLC to be met, it suffices to show that there will always exist a layer l such that it meets the ILC for input port i and meets the OLC for output ports $\langle P_j\rangle$, where, $j \in \{1, \dots, m\}$.

Thus layer l must meet all the above constraints i.e.

$$l \in \{AIL(i, n) \cap AOL(P_1, DT(n, i, P_1)) \cap$$
$$AOL(P_2, DT(n, i, P_2)) \cap \ldots (AOL(P_m, DT(n, i, P_m)))\}$$

This will be satisfied when,

$$|AIL(i, n)| + |AOL(P_1, DT(n, i, P_1))| + |AOL(P_2, DT(n, i, P_2))| +$$
$$\ldots + |AOL(P_m, DT(n, i, P_m))| > mk$$

From Lemma 3.1 and 3.2 we know that,

$$|AIL(i, n)| + |AOL(P_1, DT(n, i, P_1))| + |AOL(P_2, DT(n, i, P_2))| +$$
$$\ldots + |AOL(P_m, DT(n, i, P_m))| > mk$$

if,

$$S > (m + 1)\ .\ ❒$$

Lemma 4.4: *(Sufficiency) A PPS, with a maximum fanout of* m*, can exactly mimic an OQ switch with a PIFO queueing discipline, with a speedup of* $S > (2m + 1)$ *, using fanout multicasting.*

Proof: Since the maximum fanout of a multicast cell is m, each multicast cell is destined to a maximum of m outputs. Consider a cell C that arrives at external input port i at time slot n and destined for output ports $\langle P_j \rangle$, where, $j \in \{1, \ldots, m\}$. For the ILC and OLC to be met, it suffices to show that there will always exist a layer l such that

the layer l meets all the following constraints i.e.

$$l \in \{AIL(i, n) \cap$$
$$AOL(P_1, DT(n, i, P_1)) \cap AOL(P_1, DT(n, i, P_1) + \lceil k/S \rceil - 1) \cap$$
$$AOL(P_2, DT(n, i, P_2)) \cap AOL(P_2, DT(n, i, P_2 + \lceil k/S \rceil - 1)) \cap \ldots$$
$$AOL(P_2, DT(n, i, P_2)) \cap (AOL(P_m, DT(n, i, P_m + \lceil k/S \rceil - 1)))\}$$

This will be satisfied when,

$$|AIL(i, n)| +$$
$$|AOL(P_1, DT(n, i, P_1))| + |AOL(P_1, DT(n, i, P_1 + \lceil k/S \rceil - 1))| +$$
$$|AOL(P_2, DT(n, i, P_2))| + |AOL(P_2, DT(n, i, P_2 + \lceil k/S \rceil - 1))| + \ldots$$
$$|AOL(P_m, DT(n, i, P_m))| + |AOL(P_m, DT(n, i, P_m + \lceil k/S \rceil - 1))| > 2mk$$

From Lemma 3.1 and 3.2 we know that,

$$|AIL(i, n)| +$$
$$|AOL(P_1, DT(n, i, P_1))| + |AOL(P_1, DT(n, i, P_1 + \lceil k/S \rceil - 1))| +$$
$$|AOL(P_2, DT(n, i, P_2))| + |AOL(P_2, DT(n, i, P_2 + \lceil k/S \rceil - 1))| + \ldots$$
$$|AOL(P_m, DT(n, i, P_m))| + |AOL(P_m, DT(n, i, P_m + \lceil k/S \rceil - 1))| > 2mk$$

if, $S > (2m + 1)$. ❒

4 Devising an Optimal Strategy for Multicasting

We shall now reduce the speedup required for a PPS using a combination of both fanout and copy multicast. From the discussion in sections 3.1 and 3.2, we make the following observations.

1. The speedup required in the PPS using only copy multicasting increases in direct proportion to the number of copies made per cell. This additional speedup is required in order to transmit all the copies of the cell separately to the middle stage switches.

2. If the PPS is doing fanout multicasting, then a single cell can be sent to an internal layer which satisfies the OLC of all its constituent outputs. There is no additional speedup required to physically transmit the cell to the middle stage switch. However one requires a higher speedup just in order to be assured of finding such a middle stage switch which satisfies the multiple constraints.

3. Thus, copy multicast does not use the copying capability of the middle stage switches, whereas fanout multicast does not use utilize the speedup available on the links to the middle stage switches.

4. Hence, one could envisage that there exists an optimum strategy where, both the copying capability of the middle stage switches is utilized and the speedup of the links to the middle stage switches is utilized.

4.1 The bounded copy strategy

In this section, we introduce a method called bounded copy, which improves upon both copy multicasting and fanout multicasting. We shall bound the number of copies that can be made from a multicast cell.

Let us define q as the maximum number of copies that are made from a given multicast cell. Since, the maximum fanout of any given multicast cell is m we make each of these copies to be destined to a maximum of $\lceil m/q \rceil$ output ports.

4.2 Lower bounds on the size of the link constraint sets

Lemma 4.5: *The size of the available input link set, $|AIL(i, n)| \geq k - (\lceil k/S \rceil - 1) q$, for all $i, n \geq 0$; in a PPS using a bounded copy strategy, where S is the speedup on the internal input links.*

Consider external input port i. The only layers that i cannot send a cell to are those which were used in the last $\lceil k/S \rceil - 1$ time slots. (The layer which was used $\lceil k/S \rceil$ time slots ago is now free to be used again). $|AIL(i, n)|$ is minimized when a cell arrives to the external input port in each of the previous $\lceil k/S \rceil - 1$ time slots. Since a maximum of q links are used in every time slot, $|AIL(i, n)| \geq k - (\lceil k/S \rceil - 1) q$. ❐

Theorem 4.1:*(Sufficiency) A PPS, which has a maximum fanout of m, can exactly mimic a FCFS-OQ switch with a speedup of $S > 2\sqrt{m} + 1$.*

Proof: Consider a cell C that arrives at external input port i at time slot n and destined for output ports $\langle P_j \rangle$, where, $j \in \{1, \ldots, m\}$. This cell is divided into a maximum of q copies. Each copy is destined to distinct output ports $\langle P_j \rangle$, where, $j \in \{1, \ldots, \lceil m/q \rceil\}$. For the ILC and OLC to be met, it suffices to show that there will always exist a layer l such that the layer l meets all the following constraints i.e

$$l \in \{AIL(i, n) \cap AOL\,(P_1, DT(n, i, P_1)) \cap \\ AOL\,(P_2, DT(n, i, P_2)) \cap \ldots (AOL\,(P_{\lceil m/q \rceil}, DT(n, i, P_{\lceil m/q \rceil}))\,)\,\}$$

This will be satisfied when,

$$|AIL(i, n)| + |AOL(P_1, DT(n, i, P_1))| + |AOL(P_2, DT(n, i, P_2))| + \\ \ldots + |AOL(P_{\lceil m/q \rceil}, DT(n, i, P_{\lceil m/q \rceil}))| > (\lceil m/q \rceil)\,k$$

From Lemma 3.1 and 3.2 we know that,

$$|AIL(i, n)| + |AOL(P_1, DT(n, i, P_1))| + |AOL(P_2, DT(n, i, P_2))| + \\ \ldots + |AOL(P_{\lceil m/q \rceil}, DT(n, i, P_{\lceil m/q \rceil}))| > (\lceil m/q \rceil)\,k$$

if,

$$k - (\,(\lceil k/S \rceil - 1)\,q) + (\lceil m/q \rceil)\,(k - (\lceil k/S \rceil - 1)\,) > (\lceil m/q \rceil)\,k\,.$$

if,

$$k - (\,(\lceil k/S \rceil - 1)\,q) - (\lceil m/q \rceil)\,(\lceil k/S \rceil - 1) > 0$$

if,

$$(\lceil k/S \rceil - 1)\,(q + \lceil m/q \rceil) < k\,.$$

Now the minimum value of the speedup is got when $F(q) = (q + \lceil m/q \rceil)$ is minimized. But $F(q) = (q + \lceil m/q \rceil) < (q + m/q) + 1$. The minimum value is obtained when $q = \sqrt{m}$. Then the minimum speedup required is $S > 2\sqrt{m} + 1$.[1]❒

Theorem 4.2:*(Sufficiency) A PPS, which has a maximum fanout of m, can exactly mimic an OQ switch with a PIFO queueing discipline, with a speedup of $S > 2\sqrt{2m} + 2$.*

Proof: The proof is along the similar lines as above.

Corollary 4.1:*(Sufficiency) A PPS, can exactly mimic a multicast FCFS-OQ switch with a speedup of $S > 2\sqrt{N} + 1$ and a multicast OQ switch with a PIFO queueing discipline with a speedup of $S > 2\sqrt{2N} + 2$.*

Proof: In any multicast traffic pattern, the maximum fanout of a cell can be N. This occurs when a broadcast cell is encountered. We set the maximum fanout $m = N$ in Theorem 4.1 and Theorem 4.2 to obtain the above statement.❒

[1] In all of these proofs we find a conservative bound on the speedup. The exact bound can be lower by solving for the specific case.

CHAPTER 5

A Distributed Approach in a Parallel Packet Switch

1 Introduction

1.1 Communication complexity

In chapter 3, we saw an algorithm for the demultiplexor to choose the layer that each arriving cell is sent to. Unfortunately, the algorithm is not very practical because each of the N input multiplexors must make their decision one at a time. Cells arriving at different ports and destined to the same output must use $AOL(j, DT(n, i, j))$ in their calculation; and when an input has selected a layer to send a cell to, the set corresponding to the cell's output will change. As a result, N sequential decisions are required per time slot. We conclude that the centralized algorithm does not scale well with the number of external input ports because of the $O(N)$ sequential decisions to be made and is not practical for large N.

So instead, we now describe a distributed algorithm, called the "Distributed PPS Algorithm" (DPA) which scales better with the number of external ports, and is therefore more practical. But DPA comes with two costs: First, we must abandon the goal of the PPS mimicking an output queued switch, in exchange for just achieving work conservation. Second, and as we will see, DPA requires additional speedup.

2 DPA: A Distributed PPS Algorithm

Perhaps the most obvious distributed algorithm would be one in which each input selects a layer independently based on its own private copy of the $AOL(j, DT(n, i, j))$ for each output j. Of course, the problem with this approach is that several external inputs may choose the same layer for cells destined to the same external output, causing concentration and a loss of work-conservation (as described in Chapter 2.).

DPA attempts to overcome the need for N sequential decisions by partitioning the N external inputs into k equal-sized groups $\langle G_l \rangle\big|_{l=1}^{k}$, where $|G_l| = \lceil N/k \rceil$, $\forall\ l \in \{1, \ldots, k\}$, except maybe for $l = k$.

We will index each group member by $G_l[i]$, where $i \in \{1, \ldots, \lceil N/k \rceil\}$. There are N output schedulers, one for each external output and we will denote them by $S(j)$, where $j \in \{1, \ldots, N\}$. Each output scheduler, $S(j)$, keeps track of which layers are currently available; i.e. it maintains $AOL(j, DT(n, i, j))$.

DPA proceeds in $\lceil N/k \rceil$ iterations, $I_1, \ldots, I_{\lceil N/k \rceil}$, as follows. In the i^{th} iteration, the i^{th} member of each input group (if it has a newly arriving cell), determines the layer that it will use to send the cell to its external output; the k inputs $G_l[i]\ \ \forall\, l \in \{1, \ldots, k\}$ make requests for layers to the output schedulers, which respond by granting a layer to each input that places a request. The request-grant process in the i^{th} iteration has the following three phases:

1. **Broadcast phase**: Each output scheduler, $S(j)$ broadcasts $AOL(j, DT(n, i, j))$ to all inputs.
2. **Request phase**: All of the k inputs in $G_l[i]$, $\forall\ l \in \{1, \ldots, k\}$ may, if they have a new cell, request the output scheduler for a new layer. Consider, for example, an input $a \in G_l[i]$ which has a newly arrived cell for output b. Input a selects *two* layers that meet the ILC and OLC, (i.e. two layers from $AIL(a, n) \cap AOL(b, DT(n, a, b))$) and sends requests for both layers to output scheduler $S(b)$. We will call the two requests the primary request and the duplicate request. We will see later how and why the input selects the two layers to request, and why the intersection is guaranteed to contain at least two layers.
3. **Grant phase**: For each of the layers that requested it, every output scheduler grants one layer from among the two that were requested. The output scheduler updates its set of available links $AOL(j, DT(n, i, j))$ before the next iteration begins. When the scheduler grants to an input, it also assigns a “window” number to the granted cell indicating a time window in which the cell will depart. The window number is used by the output scheduler to determine the order in which to deliver cells onto the outgoing line. Each window is of length k time slots and so at most k cells are assigned to the same window. Notice that in a given iteration at most k inputs may request the same output; and so the cells scheduled in one iteration will either all be scheduled to depart in the same window, or in two consecutive windows (in the case when their departure times straddle two windows).

The algorithm then repeats the three-phases for each of the $\lceil N/k \rceil$ iterations. To complete the description of the algorithm, we need to determine how each input selects two layers from $AIL(a, n) \cap AOL(b, DT(n, a, b))$, and then show that the intersection always contains at least two layers.

Definition 5.1: *The Duplication Function-*

Suppose that an external input port from group g selects some layer $l \in \{AIL(i, n) \cap AOL(j, DT(n, i, j))\}$. Then in addition to layer l, it also selects layer $\hat{l} = Dup(g, l) = (g + l) \bmod k$, where k is the number of layers in the PPS. We call $Dup(g, l)$ the duplication function.

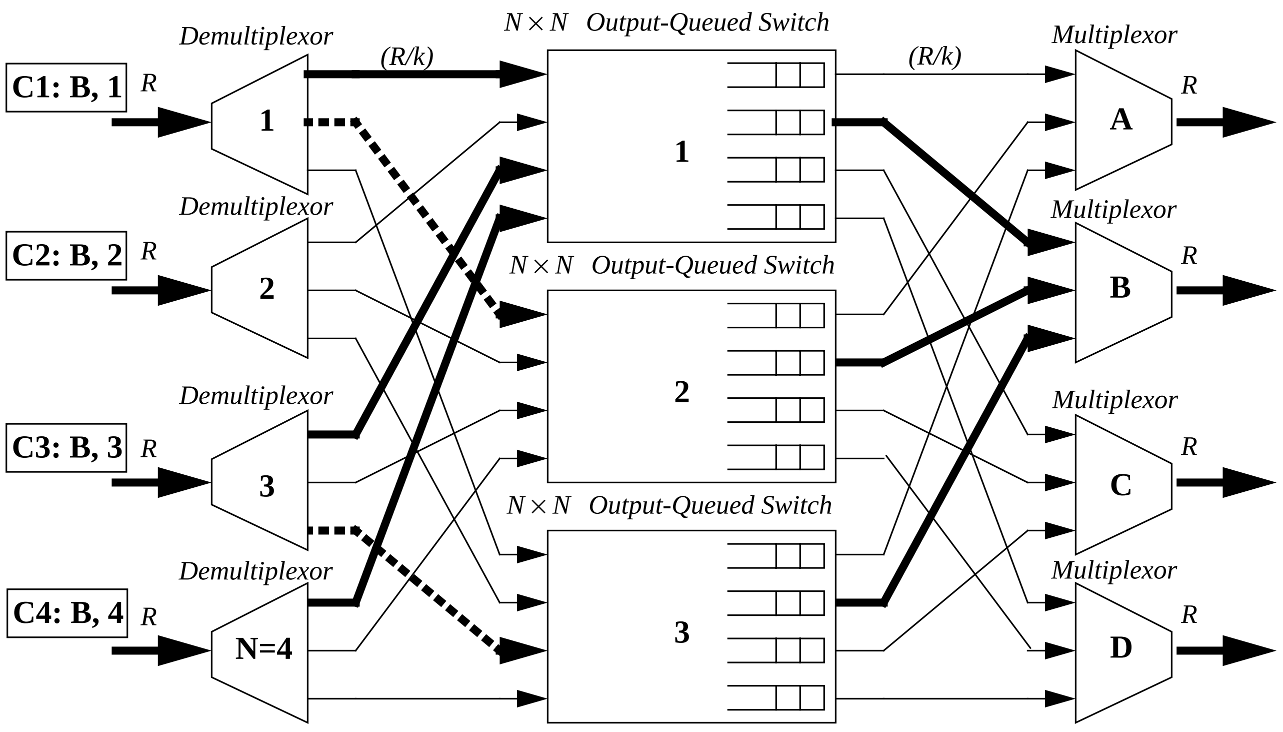


Figure 5.1: An example of duplicate requests in a 4×4 PPS, with newly arriving cells at each input all destined to output B. The thick lines indicate primary requests, and the dotted lines indicates duplicate requests.

The duplication function is chosen with the following objective: when an output scheduler receives multiple requests from different inputs in the same iteration, we would like it to be able to grant a different layer to each input, so as to prevent concentration.[1] With the duplication function defined above, as the example in Figure 5.1 shows, if all the inputs request the same layer, all the corresponding duplicate requests will be for different layers. In the example, $(G_1 = \{Input\ 1, Input\ 2\})$, $G_2 = \{Input\ 3\}$ and $G_3 = \{Input\ 4\}$ (in general when N/k is not an integer, some groups will contain an extra input; in this case G_1 does), and all four inputs have a newly arriving cell for output *B*. In the first iteration, one external input from each group participates in a request phase; in the example, inputs 1, 3, and 4 all choose layer $l = 1$, and duplicate layers $\hat{l} = 2, 3, 1$ respectively. The output scheduler $S(B)$ can now avoid concentration in this iteration by granting a different layer to each requesting input; e.g. it could grant layers $2, 4$ and 1 to inputs $1, 3$ and 4 respectively.

[1] There are presumably a variety of duplication functions that would meet this criterion.

Remark: External input 4 does not request for a duplicate layer because when the external input group number is k, it can be seen from the duplication request function, that the primary and duplicate requests are both for the same layer. Hence, inputs belonging to group k send only one request.

2.1 The conditions under which DPA can request two layers

DPA requires that an input requests two layers for each arriving cell. We can see that this requires the intersection of $AIL(i, n)$ and $AOL(j, DT(n, i, j))$ to be larger than it was before — it must now contain at least two layers, instead of just one in the previous algorithm. Worse still, the intersection must contain both a primary layer and its duplicate, as defined by the duplication function. Now the size of the intersection $AOL(j, DT(n, i, j))$ is dictated by the number of available input and output links, which in turn is determined by the speedup S of the internal links. So here, we consider how big the $AOL(j, DT(n, i, j))$ set needs to be as a function of S. We will use this requirement later when we determine the speedup required for the PPS to be work-conserving.

Lemma 5.1: *If each input is to find a primary and duplicate layer to request, we require that* $|AOL(j, DT(n, i, j))| > \lfloor k/2 \rfloor + \lceil k/S \rceil$ *when the speedup of the internal links is* S.

Proof: Consider a cell arriving at input i and destined to output j. In order to have two layers to request, the demultiplexor at external input i needs a layer and its duplicate in the intersection set $AIL\,(i, n) \cap AOL(j, DT(n, i, j))$. If the intersection set contains $\lfloor k/2 \rfloor + 1$ or more layers, then it must contain at least one layer and its duplicate; i.e. if $|AIL(i, n)| + |AOL(j, DT(n, i, j))| > k + \lfloor k/2 \rfloor + 1$ then there are two layers in the intersection set that can be requested. Now, recall from Chapter 3 that $|AIL(i, n)| \ge k - \lceil k/S \rceil + 1$, which implies that $|AOL(j, DT(n, i, j))| > \lfloor k/2 \rfloor + \lceil k/S \rceil$. ❒

2.2 The speedup required for a PPS with DPA to be work conserving

As stated earlier, DPA does not allow a PPS to mimic a FCFS-OQ switch; so instead, we will show in this section that DPA enables a PPS to be work-conserving when there is sufficient speedup.

But before we can prove the theorem we require one more lemma.

Lemma 5.2: *The maximum number of inputs that are granted by the same layer in one iteration is* $\lceil \sqrt{k} \rceil$.

Proof: Please see Appendix 2.❒

We can now proceed to the main theorem.

Theorem 5.1:*(Sufficiency) DPA is work conserving, with a speedup of* $S \ge \lceil \sqrt{k} \rceil + 4$, *for large* k.

Fact 5.1: *(Necessary for PPS to operate)* Consider the cells in a PPS belonging to the same layer and destined to the same output. For the PPS to be work conserving we require that no more than S such cells be scheduled to depart from the same layer in a given window of k time slots.

Fact 5.2: *(Sufficiency for the PPS to be work conserving)* Consider one iteration of DPA. Assume that at the beginning of an iteration every set $AOL(j, DT(n, i, j))$ consists only of layers with fewer than $S - \lceil \sqrt{k} \rceil$ cells scheduled to depart in the present window. Then from Lemma 5.2 we know that at the end of the iteration, there can be no more than $S - \lceil \sqrt{k} \rceil + \lceil \sqrt{k} \rceil = S$ cells destined to the same output from the same layer. DPA will then satisfy Fact 5.1 and the PPS will be work conserving.

From Lemma 5.1 we know that $|AOL(j, DT(n, i, j))| > \lfloor k/2 \rfloor + \lceil k/S \rceil$ for all $j, DT(n, i, j) \geq 0$. Now, consider one external output, j, which is currently scheduling cells from departure window W_j. In general, different layers will contain different numbers of cells destined to this output during window W_j. We will consider those layers with the smallest number. In particular, we will consider the $\lfloor k/2 \rfloor + \lceil k/S \rceil$ layers which have the smallest number of cells destined to output j during W_j.

Reductio-Ad-Absurdum: **(Assumption)** Assume that some of these layers do not meet the constraints as described in Fact 2; i.e. there exist at least $k - (\lfloor k/2 \rfloor + \lceil k/S \rceil) = \lceil k/2 \rceil - \lceil k/S \rceil$ layers which have more than $S - \lceil \sqrt{k} \rceil$ cells that need to depart in window W_j. Hence, the PPS contains at least $(\lceil k/2 \rceil - \lceil k/S \rceil)(S - \lceil \sqrt{k} \rceil)$ cells that are scheduled to depart in window W_j. But, this number is greater than or equal to k if $S \geq \lceil \sqrt{k} \rceil + 3$ and $k > 16$ or when $S \geq \lceil \sqrt{k} \rceil + 4$ and $k > 2$ **(contradiction)**.

Therefore the assumption made in the proof cannot be true when DPA has the required speedup as described above. In such a case, DPA will always meet the requirements as described in Fact 2 and will be remain work conserving. ❐

CHAPTER 6

Mimicking an OQ Switch within Delay Bounds

1 Introduction

In previous chapters, we showed that the parallel packet switch can mimic a FCFS-OQ switch receiving unicast traffic using a centralized approach called CPA. Also a PPS with DPA can be shown to be work conserving. However DPA required a very large speedup whereas CPA involved a very large communication overhead.

In this chapter we shall continue to consider a FCFS queueing discipline and restrict ourselves to unicast traffic. In order to have a practical solution, we shall further relax the requirements on a PPS such that we mimic a FCFS-OQ switch *within a specific delay bound*. We would also like to get rid of communication complexity and high speedup in a PPS. Hence we ask the following questions.

1. Can the inputs and outputs be made to operate independently?
2. Can we further decrease the speedup required in the internal layers?

2 Terminology

Independent Demultiplexors: The demultiplexors in a PPS are said to be independent when they do not share any state or information amongst themselves i.e. they do not communicate with each other.

Local Departure Time: When a cell arrives, the demultiplexor selects a departure time for the cell. This departure time is selected assuming that the only cells arriving to the PPS were from that specific demultiplexor. Hence the departure time is selected locally without any information about the other demultiplexors. A cell arriving to input i at time slot n and destined to output j is assigned the local departure time $LDT(n, i, j)$.

Local AOL Set: The local available output link set $LAOL(j, LDT(n, i, j))$**,** is the set of layers that can send a cell to external output j at time slot $LDT(n, i, j)$ in the future assuming that the only cells which arrived at the PPS were cells from input i .Thus, $LAOL(j, LDT(n, i, j))$ is the set of layers that have not sent any of the last $\lceil k/S \rceil - 1$ cells to external output j at any time slot previous to $LDT(n, i, j)$.

We note that the local AOL set differs from the AOL set in that it is assumes that all cells in the PPS arrive from the local input only. Also the local AOL set does not contain any of the layers which sent the last $\lceil k/S \rceil - 1$ during any previous time slot, unlike the AOL set which discounted layers which sent these cells in the $\lceil k/S \rceil - 1$ timeslots previous to $DT(n, i, j)$.

Conflict-free Ordering: An ordering of cells on a given output j is said to be conflict-free on output j if output j can read these cells in the given order without violating the output link constraints.

3 Speedup Conditions on a Distributed Algorithm.

Lemma 6.1: *(Sufficiency) A speedup of* $2k/(k+2)$ *is sufficient for a PPS with independent demultiplexors to send cells from each input to each output in a conflict-free ordering*

By definition the local AOL set maintained by an input i for an output j forms a conflict-free ordering on output j. In order that the local AOL and the AIL be met, it suffices to show that there will always exist a layer l such that $l \in \{AIL(i, n) \cap LAOL(j, LDT(n, i, j))\}$, i.e. $AIL(i, n) \cap (LAOL(j, LDT(n, i, j))) \neq \varnothing$, which must be satisfied if $|AIL(i, n)| + |LAOL(j, LDT(n, i, j))| > k$. Also the local AOL set $LAOL(j, DT(n, i, j))$ has the same minimum bound as the AOL set. From Lemma 3.1 and 3.2 we know that $|AIL(i, n)| + |LAOL(j, LDT(n, i, j))| > k$ if $S > 2k/(k+2)$. ❐

3.1 Relaxing the conditions on mimicking an OQ switch

We have shown that it is possible for a PPS with independent demultiplexors to send cells in a conflict-free manner to each output j with a speedup of two. However for an output j to mimic an OQ switch we require that consecutive cells which leave the output of the OQ switch (irrespective of the inputs on which they arrive) should appear to the output in a conflict-free ordering.

Since this is not true in general, we shall relax the constraints on mimicking an OQ switch. By the very nature in which cells are sent to the middle stage switches there can be a concentration of cells for a given output in one middle stage switch. However the concentration is only temporary as cells from all inputs are distributed in a conflict-free manner to a specific output. However since temporary concentration may occur in one middle stage switch and may not occur in another, certain cells can reach the outputs earlier or later than others. This implies that these cells can get delayed and the departure cell order could be different from that of the shadow OQ switch. However we shall impose the following two requirements:

1. Firstly, we would like this delay to be bounded.
2. Secondly, we require that cells arriving at an input i destined to an output j be sent in the same order as that of the shadow OQ switch. Thus cells may be re-ordered on a given output, but not between an input-output pair.[1]

3.2 Mimicking an OQ switch within a specific delay bound

We now present the main idea behind a distributed algorithm which will mimic an FCFS-OQ switch within a specific delay bound. Each demultiplexor will request the middle stage switches for the next cell which should be dispatched from its output queue. However we know that consecutive cells which are meant to be dispatched by the demultiplexor may be queued in the same middle stage switch. But we know that since cells from each input to each output are maintained in a conflict-free ordering, there cannot be more than N consecutive cells for a given output which are queued in the same middle stage switch.

In general, a cell C which is already queued in a middle stage switch, might be delayed by not more than N internal time slots in reaching an external output. Also certain other cells which arrived from the same input from which cell C arrived and destined to the same output as cell C might have reached the external output ahead of it. Hence one will require a re-sequencing buffer at the external output in order to maintain cell order between an input and output.

We see that a distributed approach naturally requires the PPS to have a re-sequencing buffer at the multiplexors. We will now use a buffer at the demultiplexors also and further reduce the speedup required in a PPS. We shall then present an integrated scheme in the next chapter and prove the worst case delay bounds on any single cell.

4 Using Buffering in a PPS

In all previous chapters we have looked upon a PPS as a switch with buffers in the middle stage layers only. We now consider the PPS with a fixed sized buffer operating at the line-rate at the demultiplexors.

[1]. As such this is a requirement of many switches and routers. This is done because TCP performs poorly when cells get re-ordered.

4.1 Terminology

Buffer Degree: The degree d of a buffer is defined as the maximum number of cells (including any cells in transmission) which can be queued in a demultiplexor for each middle stage switch.

The buffer in the demultiplexors is organized as follows. Each demultiplexor allots a separate local buffer for each middle stage switch. Whenever a cell is to be sent to a middle stage switch l it is queued in the separate local buffer for l and is dispatched as soon as it reaches the head of the local buffer. The size of each of these local buffers is the same and the maximum number of cells it can hold is given by the buffer degree. We note that if the degree of a buffer is d, then it can hold a total of kd cells with a maximum of d cells being allowed for each middle stage switch.[2]

Buffered Allowable Input Link Set-. The buffered available input link set, $BAIL(i, n)$, is the set of layers to which external input port i can start sending a cell between time slot n and $n + dk$. This is the set of layers for which the number of cells queued for that layer at time slot n is less than the buffer degree.

Note that $|BAIL(i, n)| \leq k, \forall\,(i, n)$.

5 Reducing the Speedup with Buffering

Lemma 6.2: *The local available output link set $LAOL(j, LDT(n, i, j))$ changes in a round robin manner for a PPS with speedup one.*

By definition, when the speedup is one the local available output link set is the set of layers that have not sent any of the last $k - 1$ cells to external output j. Once an output receives $k - 1$ cells from a given input, there can be only one layer available in the local available output set. After that the local AOL set changes in a round robin manner.

Lemma 6.3: *The number of cells $D\,(i, l, T)$ sent by a demultiplexor i to middle stage switch l in any time interval T is bounded by*

$$D\,(i, l, T) \leq T \qquad if \qquad T \leq N$$

$$D\,(i, l, T) \leq \left\lceil \frac{T}{k} \right\rceil + N - 1 \qquad if \qquad T > N$$

[2] Till now we have only considered a PPS with zero buffer degree.

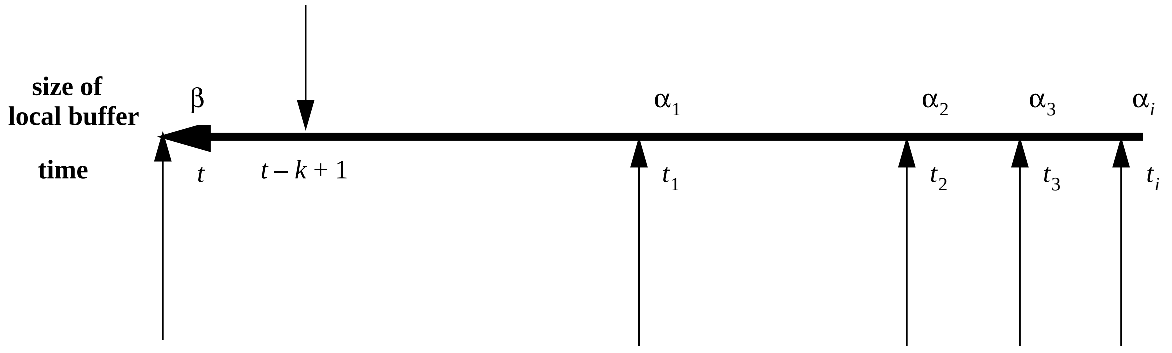


Figure 6.1: A sequence of arrivals to a buffer with degree β. The size of the local buffer for the demultiplexor i for middle stage switch l is shown.

We know from Lemma 6.1 that each local available output link set changes in a round robin manner. Hence for every k cells received by a demultiplexor for a specific output exactly one cell is sent to each middle stage layer. We can write $S(i, T) = \sum_{j=1}^{N} \bar{S}(i, j, T)$, where $\bar{S}(i, j, T)$ is the sum of the number cells sent by the demultiplexor i to output j in time interval T and $S(i, T)$ is the sum of the of the number of cells sent by the demultiplexor to all outputs in time interval T.

We have $D(i, l, T) \le \sum_{j=1}^{N} \left\lceil \frac{S(i, j, T)}{k} \right\rceil \le \left\lceil \sum_{j=1}^{N} \frac{S(i, j, T)}{k} \right\rceil + N - 1 = \left\lceil \frac{S(i, T)}{k} \right\rceil + N - 1 \le \left\lceil \frac{T}{k} \right\rceil + N - 1$.

The proof for $T \le N$ is obvious.

Theorem 6.1:*(Sufficiency) A PPS with independent demultiplexors and no speedup, can send cells from each input to each output in a conflict-free ordering with a buffer of degree N.*

Proof: We know by definition of the local AOL set that the cells from a given input are dispatched to a given output in a conflict-free manner. Hence, it remains to be shown that a conflict-free dispatch policy does not overflow the buffer at the demultiplexor.

Reductio-Ad-Absurdum: **(Assumption)** Assume that the statement of the above theorem is not true. Let a buffer of degree β not be sufficient. Let t be the first time that the buffer at input i overflows for some middle stage switch l. Let the buffer degree be β. Then the buffer has exactly β cells destined to middle stage switch l. Let t_1 be the time at which the first of these β cells came. This is shown in Figure 6.1. Let the number of cells in the buffer for middle stage switch l at time t_1 be α_1. Similarly let t_2 be the time at which the first of these α_1 cells came. Let the number of cells in the buffer for middle stage switch l at time t_2 be α_2 and so on. Finally let t_i be the time at which the number of cells in the buffer $\alpha_i = 0$. We then have the following two cases:

1. **Case One:** $t - t_i \le N$. If the time interval $T = t - t_i \le N$, then $D(i, l, T) \le T \le N$. Thus less than N cells have been sent to the buffer for middle stage l since the time t_i at which it was empty. Hence the buffer for output j cannot overflow if $\beta = N$. Thus setting the buffer degree N gives us a contradiction to the above assumption and hence a buffer of degree N is sufficient in this case.
2. **Case Two**: $t - t_i > N$. We know that the first amongst a total of $\sum \alpha_i$ cells which arrived at the buffer for middle stage l and left the buffer cam at time t_i. Also since the buffer at time t still contains β cells, the last of the $\sum \alpha_i$ could not have left before time $t - k + 1$. Since the buffer was never empty between time t_i and time $t - k + 1$ we have that at least $\sum \alpha_i = \lceil (t - k + 1 - t_i) / k \rceil \ge \lceil (t - t_i) / k \rceil - 1$ cells left the buffer for middle stage l. Now, we know that for time interval $t - t_i$, the number of cells which can arrive for a middle stage switch l, is bounded by $D(i, l, T) \le \lceil (t - t_i) / k \rceil + N - 1$. But we know that $D(i, l, T) = \beta + \sum \alpha_i$. Hence we have $\beta + \sum \alpha_i \le \lceil (t - t_i) / k \rceil + N - 1$. Again, we get a contradiction if we set $\beta = N$. Thus, a buffer degree N is sufficient in this case too.

Since the buffer never overflows, the PPS can ensure a conflict-free ordering for all cells sent from each input to each output. ❐

Theorem 6.2:*(Sufficiency) A PPS with independent multiplexors and speedup one, can read cells from each input to a specific output in a conflict-free ordering with a buffer degree N.*

Proof: The proof is by appealing to symmetry. We compare the independent demultiplexors to the independent multiplexors. In the former cells which arrive on a single input are destined to different outputs and amongst cells destined to a specific output, these are written in a round robin manner to the middle stage switches. The local buffers are used to buffer requests to middle stage switches temporarily.

In the latter, cells which are destined to a single output arrive from different inputs and amongst cells which belong to the same input, these are read in a round robin manner from the middle stage switches. The local buffers in the outputs are used to delay the cells from the middle stage switches temporarily so that they can be re-ordered.❐

Theorem 6.3:*(Sufficiency) A PPS with independent demultiplexors and multiplexors and a speedup of one having a buffer degree N in both the multiplexors and the demultiplexors can mimic a FCFS-OQ switch within a delay bound of $2N$ time slots.* ❐

Proof: The proof follows from Theorem 6.1 and 6.2. A cell faces a maximum delay of N time slots in each of the multiplexor and the demultiplexor. Hence the maximum total delay faced by a cell is $2N$ time slots.

CHAPTER 7

Conclusions

1 Introduction

While it is difficult to predict the growth of the Internet over the coming years, it seems certain that packet-switches will be required with: (1) increased switching capacity, (2) support for higher line-rates, and (3) support for differentiated qualities of service. All three of these requirements present challenges of their own. For example, higher capacity switches may require new architectures; higher line-rates may grow to exceed the capabilities of commercially available memories, making it impractical to buffer packets as they arrive; and the need for differentiated qualities of service may require performance comparable to output-queued switches. The difficulty of overcoming these challenges could be seen as an argument for circuit switching where switching is simple, buffers are not needed, and qualities of service are achieved through peak-provision of bandwidth.

While the use of circuit-switching may be appealing, we considered here an alternative — a parallel packet switch (PPS) which achieves high capacity by placing multiple packet switches in parallel, rather than in series with each other as is common in multistage switch designs. Hence, each packet that passes through the system encounters only a single stage of buffering; furthermore, and most interestingly, the system line-rates may operate in excess of the speed of the buffer memory. The main result of this work is that it theoretically possible to build such a PPS that exactly mimics an output-queued packet switch regardless of the nature of the arriving traffic. The mimicking can be maintained even when the system is providing guaranteed qualities of service. In short, and at first glance, it appears that all three of the challenges outlined above are overcome by the PPS system.

Unfortunately, as described above, the result is only a theoretical one. There are two hindrances to making the PPS practical. First, each layer of the PPS is an output-queued (OQ) switch itself, which implies that the speed of its buffer memory must grow linearly with the number of ports of the PPS. We *can* overcome this by replacing each output-queued switch with a Combined Input-Output Queued (CIOQ) switch that mimics its behavior using an internal speedup of two. Each memory in the system could now run at a rate independent of the number of ports of the PPS, and can be made arbitrarily small by increasing the number of packet-switches used. This solution, however, requires that the CIOQ switch be made practical — something for which work is in progress, but no solution is yet available.

The second hindrance is that the theoretical result assumes a centralized algorithm to determine which layer each arriving cell is sent to. We believe that the algorithm may take too long to run, as its running time grows linearly with the number of ports of the PPS. Although we have proposed here a distributed algorithm (DPA), it only goes some way towards overcoming this hindrance. First, DPA does not guarantee that the PPS will mimic an OQ switch; instead it only ensures that the PPS is work-conserving. Second, DPA requires each packet switch to run several times faster than before.

By allowing a small, finite buffer at each input and output (operating at the line rate), we find that it is possible for a PPS to mimic a FCFS-OQ switch within a specific delay bound. We think that this approach is the most promising since it does away with the running time complexity of CPA and the communication complexity of DPA. It can also function without speedup, albeit at the cost of some buffering at the line rate.

So, in summary, we think of this work as a first step towards building high-capacity switches that support guaranteed qualities of service in which memory bandwidth is not the bottleneck.

References

[1] S. Chuang, A. Goel, N. McKeown, B. Prabhakar, “Matching output queueing with a combined input/output-queued switch,” *IEEE J.Sel. Areas in Communications,* Vol. 17, no. 6, pp. 1030-1039, June 1999.

[2] B. Prabhakar and N. McKeown, “On the speedup required for combined input and output queued switching,” *Automatica,* Vol. 35, pp. 1909-1920, December 1999.

[3] P. Fredette, “The past, present, and future of inverse multiplexing,” *IEEE Communications,* pp. 42-46, April 1994.

[4] J. Duncanson, “Inverse multiplexing”, *IEEE Communications,* pp. 34-41, April 1994.

[5] J. Turner, “Design of a broadcast packet switching network,” *IEEE Trans. on Communications,* pp. 734-743, June 1988.

[6] H. Kim, A. Leon-Garcia, “A self-routing multistage switching network for broadband ISDN,” *IEEE J. Sel. Areas in Communications,* pp. 459-466, April 1990.

[7] I. Widjaja, A. Leon-Garcia, “The helical switch: A multipath ATM switch which preserves cell sequence,” *IEEE Trans. on Communications*, Vol. 42, no. 8, pp. 2618-2629, Aug 1994.

[8] F. Chiussi, D. Khotimsky, S. Krishnan, “Generalized inverse multiplexing of switched ATM connections,” in *Proc. IEEE Globecom ‘98 Conference. The Bridge to Global Integration*, Sydney, Australia, Nov. 1998.

[9] C. Clos, “A study of non-blocking switching networks,” *Bell Systems Technical Journal* 32, 1953.

[10] A. Demers, S. Keshav; S. Shenker, “Analysis and simulation of a fair queueing algorithm,” *J. of Internetworking: Research and Experience,* pp.3-26, 1990.

[11] A. Parekh and R. Gallager, “A generalized processor sharing approach to flow control in integrated services networks: the single node case,” *IEEE/ACM Transactions on Networking*, vol. 1, no. 3, pp. 344-357, June 1993.

[12] L. Zhang, “Virtual Clock: A new traffic control algorithm for packet switching networks,” *ACM Transactions on Computer Systems*, vol.9 no.2, pp.101-124, 1990.

[13] M. Shreedhar and G. Varghese, “Efficient fair queueing using deficit round robin,” in *Proc. ACM Sigcomm*, Sep 1995, pp. 231-242.

APPENDIX 1

A Centralized Algorithm for a PPS

1 Centralized PPS Algorithm

We now present an insert and dispatch scheme, called CPA (Centralized Parallel Packet Switch Algorithm) in order to emulate a FCFS-OQ switch based on the results obtained in Chapter 3.

1.1 Notation

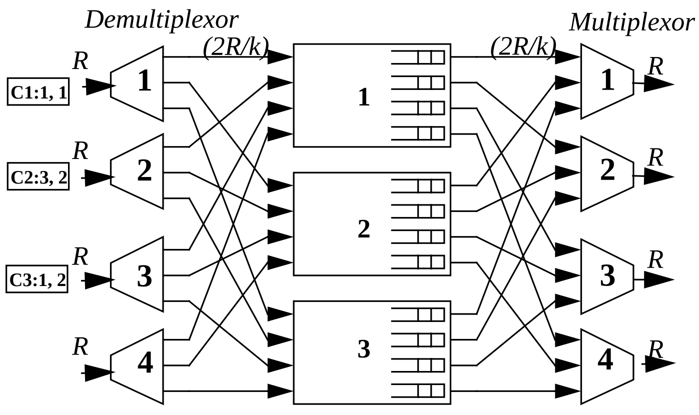
1. $HOL(j, l)$, denotes the head of line of cell at output port j of internal switch l.
2. $T(c, j)$, denotes the sequence number tagged to a cell c which is destined to output port j The sequence $T(c, j)$ denotes the FIFO order of all cells destined to output port j. These tags are unique for each cell destined to output port j.

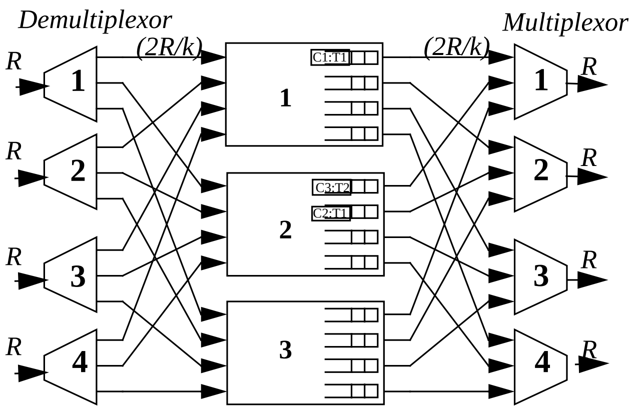
1.2 Steps in CPA

CPA consists of two parts which operate independently at each of the external input and the output ports. There is a centralized scheduler which maintains the allowable output link set for each output.

1. **De-multiplexor:** Each external input port maintains its allowable input link set. When a cell c arrives to external input port i destined to output port j at time slot n, the input port requests the centralized scheduler for a layer. A copy of $AIL(i, n)$, and as well as the destination port number j is sent. The centralized scheduler then computes a layer l, such that $l \in \{AIL(i, n) \cap AOL(j, DT(n, i, j))\}$, as described in Theorem 3.1. The centralized scheduler also returns a tagged sequence number $T(c, j)$ asso-

(a) Cells arriving at time slot 0 and being sent to the middle stage switches





Cell C1 chooses layer 1 arbitrarily from {1,2,3} ^ {1,2,3}

AOL(1,1) is updated to {1,2,3} - {1} = {2,3}

AIL(1,1) is updated to {1,2,3} - {1} = {2,3}

Cell C2 chooses layer 2 arbitrarily from {1,2,3} ^ {1,2,3}

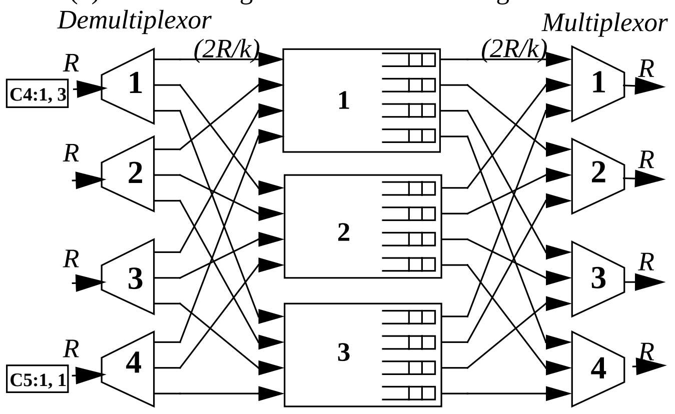
Cell C3 has a departure time d(0,3,1)=1

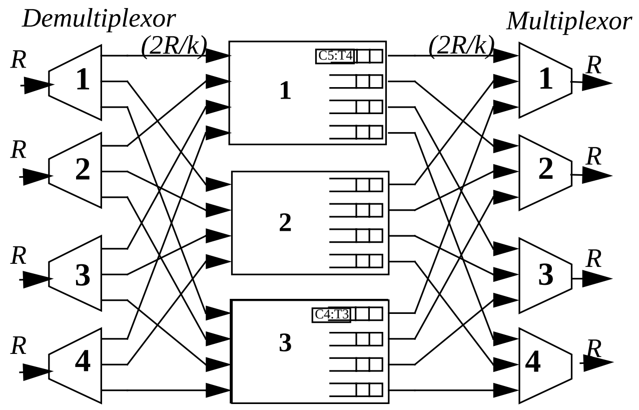
Cell C3 has to choose from AIL(3,0) ^ AOL(1,1)

Cell C3 chooses layer 2 from {1,2,3} ^ {2,3}

AOL(1,2) is updated to {2,3} - {2} + {1} = {1,3}

(b) Cells arriving at time slot 1 and being sent to the middle stage switches





Cell C4 has an expected departure time d(1,1,1) = 2

Cell C4 has to choose from AIL(1,1) ^ AOL(1,2)

Cell C4 chooses layer 3 from {2,3} ^ {1,3}

AOL(1,3) is updated to{1,3} - {3} + {2} = {1,2}

Cell C5 has an expected departure time d(4,1,1)=3

Cell C5 has to choose from AIL(4,1) ^ AOL(1,3)

Cell C5 chooses layer 1 from {1,2,3} ^ {1,2}

AOL(1,4) is updated to {1,2} - {1} + {3} = {2,3}

Figure A1.1: The CPA algorithm for a 4×4 PPS. The notation Cq:j, l denotes a cell numbered q, destined to output port j, and sent to layer 1. The notation Cq:Tp denotes the same cell numbered q, being tagged with a sequence number p.

ciated with the arriving cell c. The external input i tags the cell with the output port number j and sequence number $T(c, j)$ and transmits the cell to layer l. The values of $AIL(i, n)$, $AOL(j, DT(n, i, j))$, n and $DT(n, i, j)$ are updated.

1. **Multiplexor:** We assume that the cells at the output queues of the internal switches are maintained in sorted order in the ascending order of their sequence numbers. Each external output port j computes a layer l such that $T(HOL(j, l), j) = Min\,(\forall l)\,T(HOL(j, l), j)$. The output port chooses this layer to dispatch the next cell.

In the example shown in Figure A.1a at the input cells $C1$, $C2$, $C3$ arrive at a 4×4 PPS with $k = 3$ layers and $S = 2$ at time $t = 0$. These cells are sent to layers $1, 2, 2$ respectively. The AIL and the AOL is updated at the input after each cell is sent. Cells $C4, C5$ arrive at time slot $t = 1$ and are sent to layers $3, 1$ respectively.

At the output as shown in Figure A.1b, cells $C1, C3, C4, C5$ are tagged sequence numbers $1, 2, 3, 4$. This is the FIFO order in which they are sent to output 1 respectively. Cell $C2$ is tagged sequence number 1 as it is the first cell destined to output 3. The output ports choose the order of departure by calculating the minimum amongst all cells at the head of line.

1.3 Practical considerations

1. **De-multiplexor**: In CPA the maintenance of AIL is relatively straightforward since by definition the AIL is maintained locally by each input and it changes at most once every external time slot. However the maintenance of AOL requires significant computation as it might get updated N times in a single time slot. This also necessitates a large amount of communication between each input and the central scheduler which maintains the AOL for all outputs. Clearly, this is impractical for large N.

1. **Multiplexor**: Each external output chooses the correct order of cells by computing the minimum tagged sequence number among all cells at the head of line of its corresponding output queue. A maximum of k such computations will have to be performed in each time slot. We can reduce this time by performing a one time sort operation on the head of line cells and then maintaining these values in sorted order for each additional head of line cell which appears at the output.

There is also the problem of sequence numbers being exhausted and issues related to re-use of sequence numbers [8]. We do not address these issues in this report.

APPENDIX 2

Proof of Lemma 5.2

1 Some Preliminary Lemmas

Before proving Lemma 5.2, we will need the following simple lemma.

Lemma A2.1: *If x external inputs generate primary and duplicate requests in a single scheduling stage of DPA, they will make requests to at least $\lceil \sqrt{x} \rceil$ layers.*

Proof: Assume that in a single iteration of DPA, x inputs request a total of q layers. Each input picks a primary and a duplicate layer, which we denote by the tuple (a, b), where a and b are the primary and duplicate layers respectively. In this case, a and b can take on one of q different values. Now, the number of different 2-tuples that can be formed from a set of size q is $2 \times C(q, 2) = q(q-1)$. Also, at most one input (which belongs to the input group k) can generate a tuple of the form (a, a). Therefore, given that the duplication function ensures that each input generates a different tuple, there can be at most $q(q-1) + 1$ inputs that sent requests to these q layers i.e. $x \leq q(q-1) + 1$, which implies that $q \geq \lceil \sqrt{x} \rceil$ layers. ❐

2 Proof of Lemma 5.2

We can now proceed to prove the main lemma of this appendix.

Lemma A2.2: *In any one iteration, at most $\lceil \sqrt{k} \rceil$ inputs are granted to send to the same layer.*

Before we commence the proof, it is worth considering why the number of inputs granted to the same layer is less than k. After all, there are k groups of inputs, all of which can make requests in the same iteration. The reason is because of the duplicate function — the duplicate requests give the output schedulers more choice of which layers to use, and hence the opportunity to reduce concentration in any one layer.

Proof: We will start by considering $G(j, l)$, the number of inputs granted in one iteration to send cells destined to output j via layer $\hat{l}$. We'll denote the maximum $\hat{G} = Max \forall (j, l)\ G(j, l)$ and we will call any layer, $\hat{l}$, that maximizes $G(.,.)$ for some output a "Max-layer", and we will denote the set of all Max-layers by $\hat{L}$. In what follows, we will find a lower-bound on the number of Max-layers, $|\hat{L}|$ in any one iteration, then use this number to prove the lemma.

First, let us consider how knowing $|\hat{L}|$ proves the lemma. Suppose we know that there are at least $|\hat{L}|$ Max-layers. Since they contain $\hat{G}$ cells each, then given that at most k external inputs can send requests, we get that $|\hat{L}|\hat{G} < k$, or in other words $\hat{G} < \lceil k / |\hat{L}| \rceil$, which gives a bound on the maximum number of cells which are granted to a given layer.

Lower bound on $|\hat{L}|$, the number of Max-layers: Consider the case when there are a set of Max-layers, and that the addition of one more grant will cause one Max-layer to make one more grant. Consider one such Max-layer, l. By definition of a Max-layer, as a result of the grants that output scheduler $S(j)$ makes in one iteration, $\hat{G}$ different inputs will send cells to output j via layer l. Note that in order for layer l to be able to send all of the $\hat{G}$ cells that it receives (consecutively and in this one iteration) to output j in consecutive time slots it will require a speedup of at least $\hat{G}$ on the output links.

Fact A2.1: In addition to layer l, there are at least $\lceil (\hat{G}/2) \rceil - 1$ other Max-layers.

This fact is true because the $\hat{G}$ inputs (that were granted layer l) sent at least $\hat{G}$ primary requests and $\hat{G}$ duplicate requests. Of these $2\hat{G}$ requests that these inputs sent, at least one of the primary or the duplicate layer from each request is layer l. Therefore, at least $\hat{G}$ requests were for layer l and $\hat{G}$ were for a set of other layers, $\bar{L}$, where $l \notin \bar{L}$. (with one possible exception as described below) Now consider the $\hat{G}$ requests for layers in set $\bar{L}$. These $\hat{G}$ requests can be partitioned into two sets: a set $\bar{L}_p$ of primary requests and a set $\bar{L}_d$ of duplicate requests. Because of the way the duplicate function is defined, both $\bar{L}_p$ and $\bar{L}_d$ contain distinct layers, and so one of the two sets contains at least $\hat{G}/2$ layers. We need to refine this last statement in two ways: first, because $\hat{G}/2$ may be non-integral, $\hat{G}/2$ becomes $\lceil \hat{G}/2 \rceil$; and second, because one input group (the k^{th} input group) requests a primary layer, but no duplicate layer, $\lceil \hat{G}/2 \rceil$ becomes; $(\lceil (\hat{G}/2) \rceil - 1)$ i.e. there are at least $(\lceil (\hat{G}/2) \rceil - 1)$ distinct layers, other than l, that receive requests.

Furthermore, each of the layers in the set $\bar{L}$ must also grant to exactly $\hat{G}$ cells in this iteration. If they did not, i.e. some layer $l \in \bar{L}$, does not grant to $\hat{G}$ requests, then the output scheduler could exchange one of these grants with one to layer l (which sent a duplicate request to layer l). But now layer l would grant to only $\hat{G} - 1$ inputs which contradicts our assumption. Hence, all $(\lceil (\hat{G}/2) \rceil - 1)$ layers are Max-layers and each, by definition, grants to $\hat{G}$ different inputs.

Fact A2.2: In addition to layer l and the $(\lceil (\hat{G}/2) \rceil - 1)$ layers described above, there are least $(\lceil \hat{G}/\sqrt{2} \rceil - \lceil \hat{G}/2 \rceil)$ other Max-layers.

From the previous fact, we know that there are at least a total of $\lceil \hat{G}/2 \rceil \hat{G}$ inputs which have cells destined for output j in this iteration. Using Lemma 1, we know that these $\lceil \hat{G}/2 \rceil \hat{G}$ inputs request at least $\lceil \hat{G}/\sqrt{2} \rceil$ layers. Hence there must be at least another $(\lceil \hat{G}/\sqrt{2} \rceil - \lceil \hat{G}/2 \rceil)$ layers that contain either primary or duplicate requests from these inputs. But these layers must also be Max-layers, otherwise we could exchange the grants as before and lead to a contradiction.

Continuing on the same argument, we can show that there exist another $\langle \lceil \hat{G}/\sqrt[4]{2} \rceil - \lceil \hat{G}/(\sqrt{2}) \rceil \rangle$ Max-layers, and so on. The total number of Max-layers is given by:

$$1 + \left\{ \left\lceil \frac{\hat{G}}{2} \right\rceil - 1 \right\} + \left\{ \left\lceil \frac{\hat{G}}{\sqrt[2]{2}} \right\rceil - \left\lceil \frac{\hat{G}}{2} \right\rceil \right\} + \left\{ \left\lceil \frac{\hat{G}}{\sqrt[4]{2}} \right\rceil - \left\lceil \frac{\hat{G}}{\sqrt[2]{2}} \right\rceil \right\} + \left\{ \left\lceil \frac{\hat{G}}{\sqrt[8]{2}} \right\rceil - \left\lceil \frac{\hat{G}}{\sqrt[4]{2}} \right\rceil \right\} \ldots \rightarrow \hat{G}$$

Therefore, we know that there at least $\hat{G}$ Max-layers that each grant to $\hat{G}$ inputs, which means that at least $\hat{G}^2$ requests were made in this iteration. But we know that there can be at the most k external inputs which can send requests, thus we have $\hat{G}^2 < k$ or $\hat{G} < \sqrt{k}$.

This proves that a speedup of $\lceil \sqrt{k} \rceil$ is sufficient to guarantee that all requests can be granted in one scheduling time slot. ❒