Design and Evaluation of the Combined Input and Crossbar Queued (CICQ) Switch

by

Kenji Yoshigoe

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Kenneth J. Christensen, Ph.D.
Tapas K. Das, Ph.D.
Miguel A. Labrador, Ph.D.
Rafael A. Perez, Ph.D.
Stephen W. Suen, Ph.D.

Date of Approval:
August 9, 2004

Keywords: Performance Evaluation, Packet switches, Variable-length packets, Stability, Scalability

Acknowledgements

I would like to express my gratitude to my advisor Dr. Kenneth J. Christensen for providing me tremendous opportunities and support. He has opened the door for me to pursue an academic career, and has mentored me to a great extent. I would not have been able to come this far, had I not had his valuable advice throughout my Ph.D. studies. I would like to thank my committee: Dr. Tapas K. Das, Dr. Miguel A. Labrador, Dr. Rafael A. Perez, and Dr. Stephen W. Suen. I would like to acknowledge the National Science Foundation and the Department of Computer Science and Engineering at the University of South Florida for financial support. And last, but not least, I would like to acknowledge my parents. I dedicate this to you since it would not have been possible without your unconditional support.

Table of Contents

List of Tables

List of Figures

Glossary of Acronyms

2DRR	Two-Dimensional Round Robin
ARPANET	Advanced Research Projects Agency Network
ATLAS	ATm multi-LAne backpressure Switch
ATM	Asynchronous Transfer Mode
BEL	Basic ELement
BMX	Bus Matrix Switch
CBR	Constant Bit Rate
CICQ	Combined Input and Crossbar Queue/Queued/Queueing
CIOQ	Combined Input and Output Queue/Queued/Queueing
CIXB	Combined Input-one-cell-CP Buffer crossbar
CLA	Carry Look-Ahead
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CP	CrossPoint
DRRM	Dual Round Robin Matching
DSP	Digital Signal Processor
EM	Extended Memory
EXH	Exhaustive
FCFS	First-Come First-Served
FCVC	Flow-Controlled Virtual Channels
FIFO	First-In First-Out
FIRM	FCFS In Round Robin
FC	Flow Control
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPS	Group-Pipeline Scheduler
HOL	Head Of Line

IBM	International Business Machines
IBP	Interrupted Bernoulli Process
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IQ	Input Queue/Queued/Queueing
ISDN	Integrated Services Digital Network
LAN	Local Area Network
LQF	Longest Queue First
MAC	Media Access Control
MPE	Masked Priority Encoder
MSSR	Multi-Stage Self-Routing
NEBS	Network Equipment Building Standard
OCF	Oldest Cell First
OQ	Output Queue/Queued/Queueing
ORR	Overlapped Round Robin
PCRRD	Pipeline-Based Concurrent Round-Robin Dispatching
PGPS	Packetized Generalized Processor Sharing
PIM	Parallel Iterative Matching
PPA	Ping-Pong Arbitration
PPE	Programmable Priority Encoder
PRRA	Parallel RR Arbiter
RAM	Read Access Memory
RFC	Request For Comment
RND	Random
RR	Round Robin
RRGS	Round-Robin Greedy Scheduling
RRM	Round-Robin Matching
RTT	Round-Trip Time
SRAM	Static Random Access Memory
SRM	Self-Routing switch Modules
URL	Universal Resource Locator
VHDL	Very high speed integrated circuit Hardware Description Language
VLSI	Very Large Scale Integration
VOQ	Virtual Output Queue/Queued/Queueing
WFA	Wave Front Arbiter

WFQ	Weighted Fair Queueing
WPIM	Weighted Parallel Iterative Matching
WWFA	Wrapped Wave Front Arbiter

Design and Evaluation of the Combined Input and Crossbar Queued (CICQ) Switch

Kenji Yoshigoe

ABSTRACT

Packet switches are used in the Internet to forward information between a sender and receiver and are the critical bottleneck in the Internet. Without faster packet switch designs, the Internet cannot continue to scale-up to higher data rates. Packet switches must be able to achieve high throughput and low delay. In addition, they must be stable for all traffic loads, must efficiently support variable length packets, and must be scalable to higher link data rates and greater numbers of ports. This dissertation investigates a new combined input and crossbar queued (CICQ) switch architecture.

Some unbalanced traffic loads result in instability for input queued (IQ) and CICQ switches. This instability region was modeled, and the cause of the instability was found to be a lack of work conservation at one port. A new burst stabilization protocol was investigated that was shown to stabilize both IQ and CICQ switches. As an added benefit, this new protocol did not require a costly internal switch speed-up. Switching variable length packets in IQ switches requires the segmentation of packets into cells. The process also requires an internal switch speed-up which can be costly. A new method of cell-merging in IQ switches reduced this speed-up. To improve fairness for CICQ switches, a block and transfer method was proposed and evaluated.

Implementation feasibility of the CICQ switch was also investigated via a field programmable gate array (FPGA) implementation of key components. Two new designs for round robin arbiters were developed and evaluated. The first of these, a proposed priority-encoder-based round robin arbiter that uses feedback masking, has a lower delay than any known design for an FPGA implementation. The second, an overlapped round robin arbiter design that fully overlaps round robin polling and scheduling, was proposed and shown to be scalable, work conserving, and fair.

To allow for multi-cabinet implementation and minimization of the size of the cross point buffers, a distributed input port queue scheduler was investigated. This new scheduler minimizes the amount of buffering needed within the crossbar.

The two primary contributions of this dissertation are 1) a complete understanding of the performance characteristics of the CICQ switch, and 2) new methods for improving the performance, stability, and scalability of the CICQ switch. This work has shown that the CICQ switch can be the switch architecture of the future.

Chapter 1: Introduction

1.1 Background

Switches have been a focus of research from the outset of electronic communications. The very first switches were human operators for early telephone systems in the 1890s. Performance of these "switches" was a topic of interest and resulted in the very beginnings of queueing theory. In 1909, A. K. Erlang applied probability theory to solve the problem of telephone traffic and derived formulae for loss and waiting time, which are now well known in the theory of telephone traffic [9]. This represented the start of performance modeling of electronic communication systems. In the early 1960s digital communications between computers became a point of interest for defense purposes; this interest led to the ARPANET in 1969 [24], which has evolved into the Internet of today.

Packet switching is used to exchange digital data between computers, or hosts. A packet is a block of data typically ranging from 64 to 1500 bytes in length. When the number of hosts is more than "just a few", providing $N(N-1)/2$ links and $N-1$ interfaces per host (for N hosts), it becomes infeasible to enable any communications. A switch interconnecting the hosts reduces the total number of links to N and interfaces to 1 per host making it more feasible to enable communications. Switching, then, is the foundation of a packet switched network. Packet switches are also called bridges, gateways, or routers, depending on the protocol layer at which they operate.

In the Internet, routing is done at the Internet Protocol (IP) layer. A packet switch requires buffering to store packets from temporary overload situations that occur when, for example, two hosts simultaneously send packets to a given host. Figure 1.1 shows packet switches in local area networks (LANs) that connect to the Internet at the edge between LANs and the Internet, and at the Internet's core. A typical LAN is based on the Ethernet standard [44] and may have link data rates of 10, 100, or 1000-Mbps. Link data rates on the Internet can be as high as 160-Gbps (Optical Carrier level 3072 (OC-3072)). These link data rates are increasing, along with the number of hosts in networks.

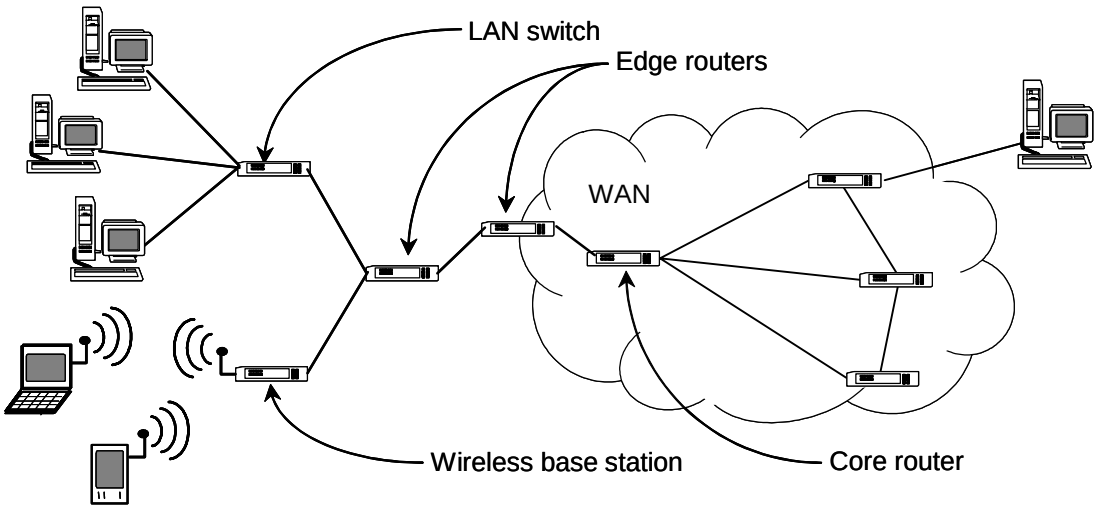

Figure 1.1 – Packet switches in the Internet

Packet switches forward packets to specific locations according to a set of rules. These rules form the basis of packet routing. The organization of components (e.g., buffering and switching elements) in a switch is commonly called its "architecture". Switch architectures are based on the location of buffering (at input or output of a

switch), the type of switching elements, and so on. This dissertation focuses on the performance of packet switch architectures and not on packet forwarding rules.

1.2 Motivation

Link data rates on the Internet are increasing faster than memory cycle rates (see Gilder's Law [30]), a fact that is driving the need to investigate switch architectures that use input buffering rather than shared memory or output buffering. In input buffered switches, the memory speed need not exceed link data rate. In shared and output buffered switches, memory speed must be *N* times link data rate (where *N* is the number of ports in the switch). Thus, for emerging 10-Gbps link data rates, shared memory and output buffered switch architectures are infeasible. The reason for this "*N* times link data rate" is described in Chapter 2 of this dissertation.

The Internet has grown exponentially over the past 30 years, with traffic reaching the maximum switch speed in 1997 [96] as shown in Figure 1.2. In the years hence, switch speed has been forced to increase at a rate equivalent to that of the growth of Internet traffic [96]. Without switch architectures that have the capability to accommodate greater link data rates and larger numbers of ports, the Internet can not continue to grow in terms of number of users or types of applications (e.g., video delivery, which requires greater data rates. The fastest commercially available switch architecture is a virtual output queued (VOQ) switch in which the input buffer in each port is partitioned into *N* queues with one queue for each of the N output ports. Though it does not require an *N* time internal speedup and can scale up to a relatively large switch speed and size, VOQ IQ switches present the following open problems:

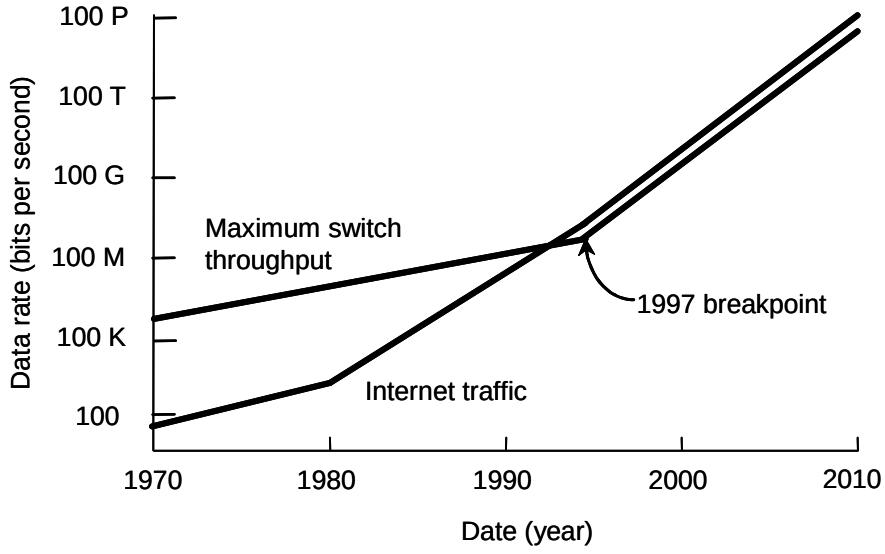

Figure 1.2 –Internet traffic vs. switch speed

1) Almost two-times the normal internal speed-up is required to support variable-length packets [106]. Internal speed-up refers to the situation in which the switch fabric and buffers must be run faster than the link rate. Speed-up adds cost and reduces the implementation feasibility of a switch.
2) An additional two-times the normal internal speed-up is needed to achieve stability for unbalanced (schedulable) traffic [20].
3) Iterative scheduling, used for VOQ IQ switches (and described in detail in Chapter 2 of this dissertation), has become a bottleneck of greater significance than issues of memory speed [90].
4) A VOQ IQ switch requires complex internal control interconnections between ports. These control interconnections can limit the number of ports that can be implemented [80].

5) A VOQ IQ switch does not exploit the high density of modern VLSI [21].
6) IQ switches cannot use existing Weighted Fair Queueing (WFQ) schedulers for providing per-flow quality of service [106].

This dissertation addresses the first five of these open problems, through an evaluation of whether they are present when a combined input and crossbar queued (CICQ) switch is used. (CICQ is a switch architecture that is described in Chapter 2.)

1.3 Contributions of this dissertation

This dissertation presents a design and performance evaluation of the CICQ switch. Methods for improving performance, eliminating instability, and improving scalability of the CICQ switch will be investigated. The major contributions of this dissertation are as follows:

1) It represents the first published performance evaluation of the round-robin polled CICQ switch [119].
2) It proposes and evaluates a new method for eliminating instability in IQ and CICQ switches; most notably, is the fact that this method does not require complex hardware or internal speed-up of switch fabric and buffers.
3) It investigates new methods for native switching of variable length packets. These methods are lower in cost and higher in performance than existing cell-based methods.
4) It demonstrates the feasibility of the CICQ switch architecture by implementing and simulating a 16-port 10-Gbps switch using FPGA technology.

5) It proposes and evaluates two new designs for faster and scalable round robin arbiters. These designs improve the scalability of the CICQ architecture to greater port counts and faster link data rates.
6) It proposes and evaluates a new method of distributed scheduling (based on rate control) within a CICQ switch. This method improves the scalability of the switch to allow for distributed, multi-rack implementations of switch components.

1.4 Organization of this dissertation

The remainder of this dissertation is organized as follows:

1) Chapter 2 provides a background of single-stage packet switch architecture leading up to the CICQ switch. The CICQ switch is shown to address open problems that exist in other non-CICQ architectures.
2) Chapter 3 is a performance evaluation of the CICQ switch architecture using simulation modeling.
3) Chapter 4 addresses open problems of instability in IQ and CICQ switches. A method to eliminate instability is proposed and evaluated.
4) Chapter 5 investigates efficient ways of handling variable-length packets. New methods for packet-to-cell segmentation for IQ switches and block transfers for CICQ switches are presented and evaluated.
5) Chapter 6 studies round robin polling as a bottleneck to scaling CICQ switches to greater port counts and link data rates. Two new round robin arbiter designs are proposed and evaluated.

6) Chapter 7 investigates a distributed VOQ scheduler for CICQ switches that does not require feedback from the switch fabric. This simplifies the implementation of CICQ switches and allows them to scale to multi-rack, distributed implementations.
7) Chapter 8 summarizes the dissertation and describes possible directions for future research.
8) End material includes a list of publications resulting from this research

Chapter 2: Background on Packet Switching

Packet switches can be classified into various categories. Classifications based on blocking vs. non-blocking, single-stage vs. multi-stage, loss vs. lossless are all possible. Comprehensive classifications of packet switch architectures have been addressed in several works including [50][94], and [112]. This chapter will focus on a review of the buffering architectures and scheduling algorithms for single-stage crossbar switches, given that these switches are non-blocking and are able to scale-up to ever-increasing link data rates and larger port counts (multi-stage switches are inherent to internal blocking). Another reason for the focus on single-stage crossbar switches, is the fact that their queueing and scheduling strategies are the dominant performance factors of packet switches.

2.1 Packet switch buffering architectures

Figure 2.1 shows an abstract view of a single-stage crossbar switch. The basic components of the crossbar switch are input ports, output ports, crossbar fabric, memories, and control units. In Figure 2.1, buffer memories and control units are not shown, as they can be implemented in various locations within the switch. The crossbar fabric contains a set of switch elements, each of which can establish a unique path between an input and an output port. Memories are needed to buffer temporary packet overloads caused by the statistical nature of packet-switched traffic. Single-stage

switches can have a number of different buffer architectures, including output queued (OQ), shared buffer, and input queued (IQ) types. Virtual output queued (VOQ) and combined input and crossbar queued (CICQ) architectures are also possible.

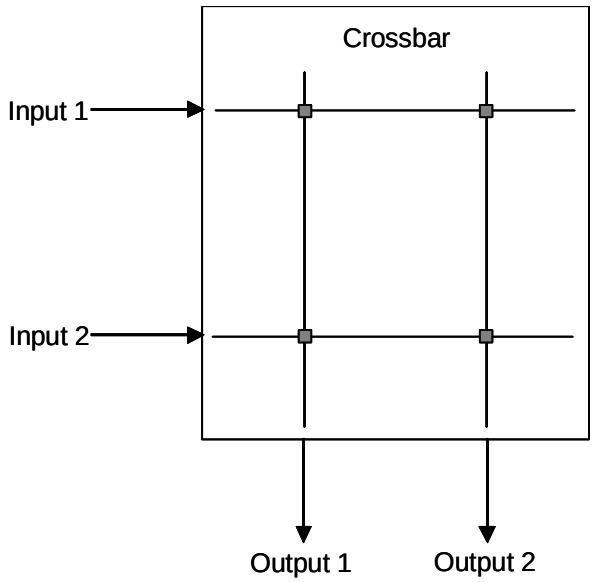

Figure 2.1 – Single stage crossbar switch

2.2 Output queued switches

OQ switches (Figure 2.2) have buffering in place at the output ports to which arriving packets are immediately forwarded. For an OQ *N* port switch, each of the *N* buffer memory speeds must operate at *N* times the link data rate if packet loss is to be prevented (*N* times link data rate is needed to support *N* writes from *N* ports forwarding a cell/packet at one time). The *N* times case occurs when *N* input ports all simultaneously forward a packet to a single output port. This "hot spot" or fan-in case occurs very frequently in client/server applications where a popular server is connected to a single

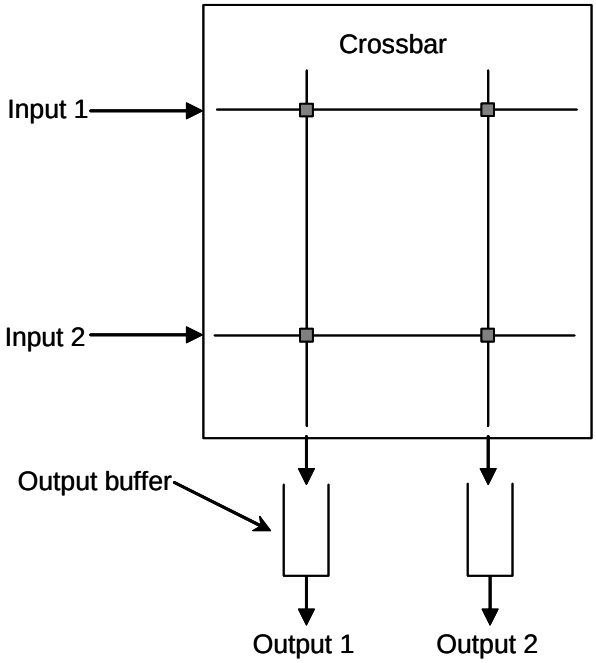

Figure 2.2 – Output queued switch

switch port and client requests arrive at the other N ports. Since link data rates are increasing appreciably faster than memory speeds, the OQ architecture is generally considered infeasible for future packet switch architectures. With a 128 bit bus and a currently feasible 2 nanosecond SRAM cycle time, memory can operate at a maximum of 64-Gbps; thus, an *N*-time link data rate is limited to a 10-gigabit switch with small port counts. Furthermore, the aggregate memory bandwidth of the OQ switch is N^2L for an *N* port switch with a link data rate of *L*. Thus, the OQ switch does not scale to large *N*.

2.3 Shared-buffer switches

Shared-buffer switches require significantly less aggregate memory bandwidth than OQ switches because the shared-buffer switches use a single memory that is shared by all

ports as shown in Figure 2.3. Packets arriving to any of the N input ports are multiplexed into a single stream and queued to the shared memory. Queued packets are then retrieved, de-multiplexed, and forwarded to the output ports. This switch architecture requires an aggregate memory bandwidth of $2NL$; at most, N packets can be read and N packets written during one packet transmission cycle. The implementation of shared-buffer switches is impossible as the $2NL$ memory speed requirement is not realistic for current link data rates, and switch ports are typically located on different boards or on different chips.

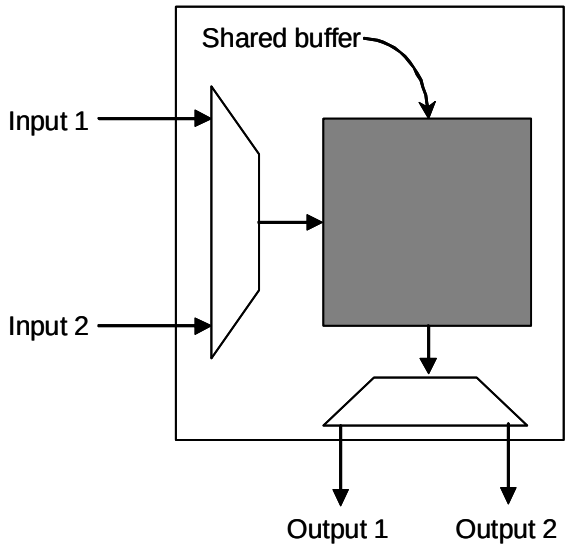

Figure 2.3 – Shared-buffer switch

2.4 Input queued switches

Input queued (IQ) switches provide buffering functions at the input ports. IQ switches with a single queue per input port as shown in Figure 2.4 have been studied

since 1987 [51]. IQ switches resolve the impractical memory speed requirements of shared buffer and OQ switches by requiring each memory to read and write one packet ($2L$ bandwidth per queue and proportional to link data rate) during a packet transmission cycle. To avoid input-output conflicts, IQ switches require an arbiter for scheduling packet forwarding. These switches have overlapped cell-transferring and switch-matrix-scheduling cycles that require fixed-length (internal to the switch) cells. Thus, upon arrival, a variable length packet is first segmented into multiple fixed-size cells. Each input port first sends a request to an output port for which it has a head-of-line (HOL) cell. Once the arbiter resolves input and output port conflicts, it grants permission to the input ports and sends control signals to set up the crossbar cross points to switch the cells within the transfer cycle.

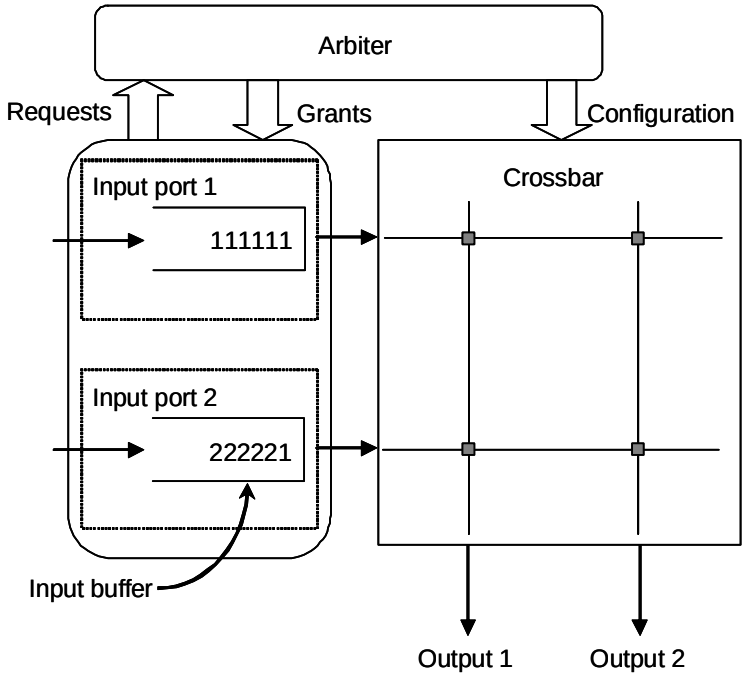

Figure 2.4 – Input queued switch

Since packets are variable in length, the last cell of a segmented packet will often contain padding bytes to ensure that the cell is fixed in length; this results in the transfer of additional bytes within the switch fabric. Thus, packets requires switch buffers and fabric to operate faster than the link data rate. In other words, an internal speed-up is needed to achieve queue stability for high offered loads. In the worst case of S byte cells and contiguous arrivals of $S+1$ byte packets, the speed-up needs to be almost a factor of 2 for the $2S$ bytes of cell data needed to switch $S+1$ bytes of packet data.

The problem with IQ switches is HOL blocking in the input queues. HOL blocking occurs when an HOL packet at an input port is not selected, due to an output port contention, and, thus, blocks the packets behind it that would have been selected. Because of this problem, IQ switches have been found to have a limited throughput of 58.6% for Bernoulli packet arrivals (for fixed-length packets, or cells) with uniformly and randomly selected output ports [51]. For bursty arrivals with non-uniformly selected output ports, throughputs of less than 58.6% are possible [60]. A simple HOL blocking scenario is illustrated in Figure 2.4: at input port 2, packets destined to output port 2 are blocked by the HOL packet due to a contention at output port 1.

One solution to the limited throughput resulting from the HOL blocking problem is to use internal speed-up. However, this would require the buffer memory to operate faster than the link data rate, a condition that is unrealistic for large N. A more practical solution to the problem would be to select queued cells other than the HOL cell for forwarding [50]. This can be accomplished either by relaxing the first-in first-out (FIFO) queueing discipline, or through input smoothing [50].

The relaxation of the FIFO queueing discipline improves performance by allowing the first *w* packets of each input queue to be available for switch outputs [50]. First, the HOL packet at each input port is used for matching. At inputs not selected to transmit the HOL packets, the packet behind the HOL packet becomes available for unmatched outputs during this time slot, and the process repeats up to *w* times. The maximum throughput for input queueing with FIFO (58.6%) is increased to 87.5%, with the relaxation of the FIFO discipline at the input queues; such look-ahead queue servicing is, however, difficult to implement.

Input smoothing improves the performance of input queueing by storing packets within a frame of *b* time slots at each input and forwarding them in parallel into a *Nb* x *Nb* switch fabric [50]. At most, *Nb* packets can go through the switch fabric, out of which *b* packets are destined to each output within a given time slot. The packets are then multiplexed onto the output line. Although input smoothing increases the throughput, it requires a costly increase in the switch fabric size (N^2 to $(Nb)^2$).

2.5 Virtual output queued IQ switches

IQ switches were considered a mere academic curiosity, due to their poor performance caused by HOL blocking. A breakthrough in IQ switch architectures, however, occurred when virtual output queueing (VOQ) for cell-based packet switching was invented in 1988 by Tamir and Frazier [110] and further developed by Anderson, et al. [3] in the early 1990s. In a VOQ IQ switch, as shown in Figure 2.5, each input buffer is partitioned into *N* queues with one queue for each output port (hence the name "virtual" output queueing). HOL blocking is eliminated because each arriving packet is

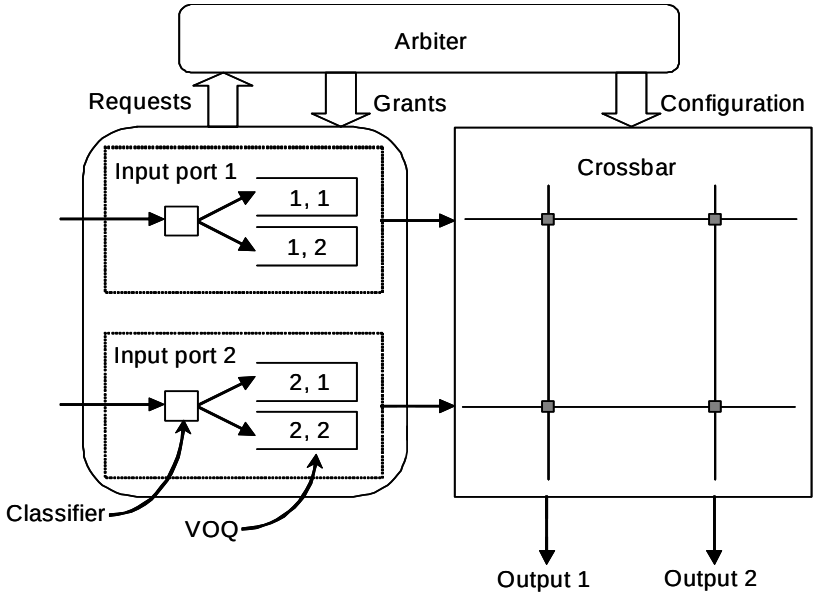

Figure 2.5 – Virtual output queued IQ switch

classified and then queued in the appropriate VOQ according to its determined destination port. The input buffer memory only needs to operate at link data rate since the queues are internal implementations within a single memory module.

For VOQs with queued cells, a scheduler must perform the one-to-one matching of input ports to output ports; therefore, the throughput of the switch is a function of the number of matches made by the scheduler in each cell transmission cycle. A VOQ IQ switch will overcome HOL blocking if a satisfactory scheduling of input ports to output ports is achieved.

2.6 VOQ scheduling algorithms

VOQ scheduling algorithms are used in VOQ IQ switches to achieve a high throughput. As with pure IQ scheduling, these scheduling algorithms are designed to

schedule fixed length (internal to the switch) cells. VOQ scheduling algorithms require complete interconnections between all input ports for the parallel sharing of the scheduling state information (the number of interconnected lines increases as $O(N^2)$. Once a match has been determined, an arbiter sets-up the crossbar cross points to switch the cells in the transferring cycle. The trade-offs in VOQ switch matrix scheduling are as follows:

1) Stability - Any schedulable load needs to be carried. Define $\lambda_{i,j}$ as the offered load from input port i to output port j where $i, j = 1, 2, \ldots, N$ for an N port switch, then a schedulable flow is

$$\sum_{i=1}^{N} \lambda_{i,j} \leq 1, \ \forall j, \text{ and} \tag{2.1}$$

$$\sum_{j=1}^{N} \lambda_{i,j} \leq 1, \ \forall i. \tag{2.2}$$

2) Fairness - No starvation should occur and bounded delay should be known for any queued packets.
3) Implementation complexity – The size of chip I/O limits the switch speed and size.

The matching of input ports and output ports in the context of a VOQ IQ switch is equivalent to bipartite graph matching. A bipartite graph is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are directly connected. A bipartite graph, G, with two disjointed sets, I (input set) and O (output set), depicts a set of requests from input ports to output ports on an NxN VOQ IQ switch. Each set has N vertices, and the edges between the vertices represent input-to-

output requests (see Figure 2.6 (a)). Bipartite graph matching is equivalent to finding a match, *M,* of any subset in *G* such that no edges in *M* have a common vertex (see Figure 2.6. (b)). Bipartite graph matching can be further subdivided into maximum or maximal matching.

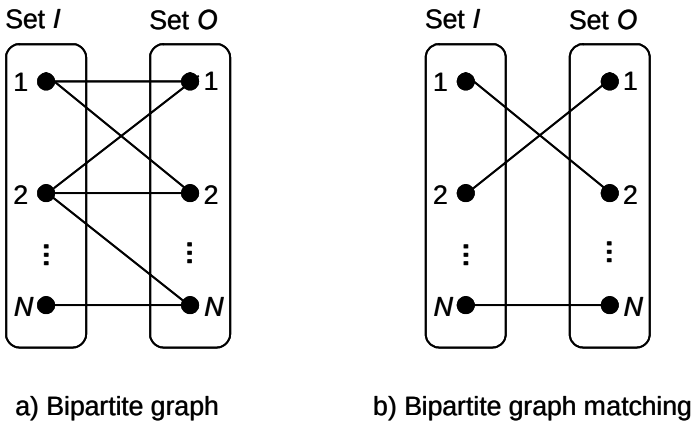

Figure 2.6 – A bipartite graph for *NxN* VOQ IQ switch

2.6.1 Maximum matching

A maximum match is a maximum cardinality bipartite matching of inputs with packet queued to *N* outputs. Maximum matching is further divided into maximum size matching and maximum weight matching. Maximum size matching [20] maximizes the number of edges in *G*, resulting in the highest possible throughput; however, it may result in instability for some schedulable flows.

Instability occurs when a schedulable flow can not be carried by the switch, a situation which results in ever increasing queue lengths (i.e., no steady state). Even with schedulable flows consisting of identical and independently distributed (i.i.d.) Bernoulli

arrivals, maximum size matching has been proven unstable [72]. Furthermore, the complexity of the fastest maximum size matching is at best $O\left(N^{5/2}\right)$ [42].

Maximum weight matching achieves a matching, *M*, that maximizes the total weight, $W = \sum_{(i,j)\in M} w_{ij}$, where w_{ij} is the weight of an edge from a vertex *i* in *I* to a vertex *j* in *O* on *G*. Stability can be achieved for all schedulable flows without speed-up if a weighted maximal match is implemented based on VOQ queue length (e.g., Longest Queue First (LQF)) or cell age (e.g., Oldest Cell First (OCF)) [77]. LQF can cause starvation of packets in a short queue and is thus unfair; however, the OCF algorithm uses the age of the HOL cells as a weight to eliminate starvation. By doing so, it guarantees that all of the HOL cells will ultimately be served since their age will continue to increase with time. Even though the OCF algorithm eliminates starvation, weighted matching requires state information to be exchanged between input and output ports and is thus generally considered infeasible to implement.

Maximum matching algorithms have a time complexity of $O(N^3)$, thus VOQ IQ switch matrix scheduling algorithms have focused on achieving maximal, not maximum, matching. With maximal matching, no edges can be added without first removing previously matched edges. The maximal matching algorithm, which has a time complexity of $O(N^2)$, is of two types: sequential and parallel.

2.6.2 Sequential matching algorithms

Sequential matching algorithms sequentially allocate unmatched output ports. All non-empty input ports make requests to the output port with the highest priority. The

output port with the highest priority then grants the request of one of the input ports via round-robin (RR). The process is repeated for all unmatched output ports; thus, it requires O(N) iterations. The time complexity of sequential matching is O(N^2).

2.6.2.1 Wave front arbiter and wrapped wave front arbiter

Wave front arbiter (WFA) [111] is a sequential matching algorithm that performs matching for requests that lie on the same diagonal of a request matrix in parallel. A request matrix is an NxN matrix with a binary entry in a cell at row i and column j indicating a request from an input i to an output j. The parallelism of WFA is based on the observation that there are no output port conflicts for the requests that lie on the same diagonal of the request matrix.

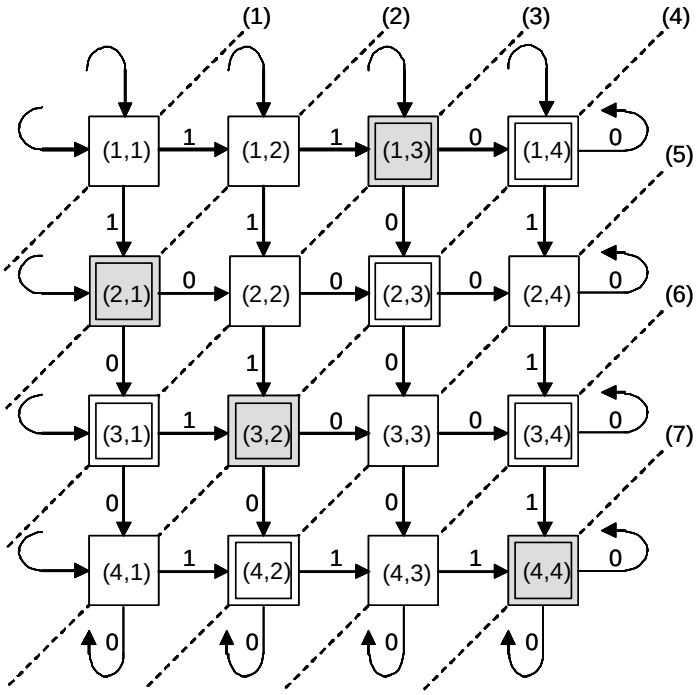

Figure 2.7 – Wave front arbiter

WFA begins with one top priority cell and makes a diagonal sweep of the request matrix from the top left to the bottom right corner of the arbiter to complete the matching process. Figure 2.7 shows the operation of WFA with a top priority cell (1,1), where the number in the diagonal indicates the progress of the wave front, and the double squares and the shaded squares indicate requests and grants, respectively. The time complexity of the WFA is $2N-1$ as at least $2N-1$ diagonals are needed to cover the entire request matrix. To maintain fairness, the top priority cell is reassigned equally among all cells in every cell cycle.

With the realization that N "wrapped" diagonals (see Figure 2.8) are guaranteed not to conflict, Wrapped WFA (WWFA) [111] was proposed to reduce the time complexity to N: all cells in a wrapped diagonal are on different rows and different columns. Thus, N wrapped diagonals provide coverage of the entire request matrix. Figure 2.8 shows the operation of WWFA with a high priority diagonal consisting of cells (1,1), (2,4), (3,3), and (4,2).

2.6.2.2 Two-dimensional round-robin

Two-dimensional round-robin (2DRR) schedulers [61] are a generalization of the WFA that achieves fairness among inputs and outputs by using a set of generalized diagonals to match the request matrix. A generalized diagonal is a set of N elements in an NxN matrix, such that no two elements are in the same row or column [61]. Two matrices aid the 2DDR: a diagonal pattern matrix and a pattern sequence matrix. The diagonal pattern matrix is a set of generalized diagonals. The pattern sequence matrix is a scheduling matrix indicating the order in which the generalized diagonals in a diagonal

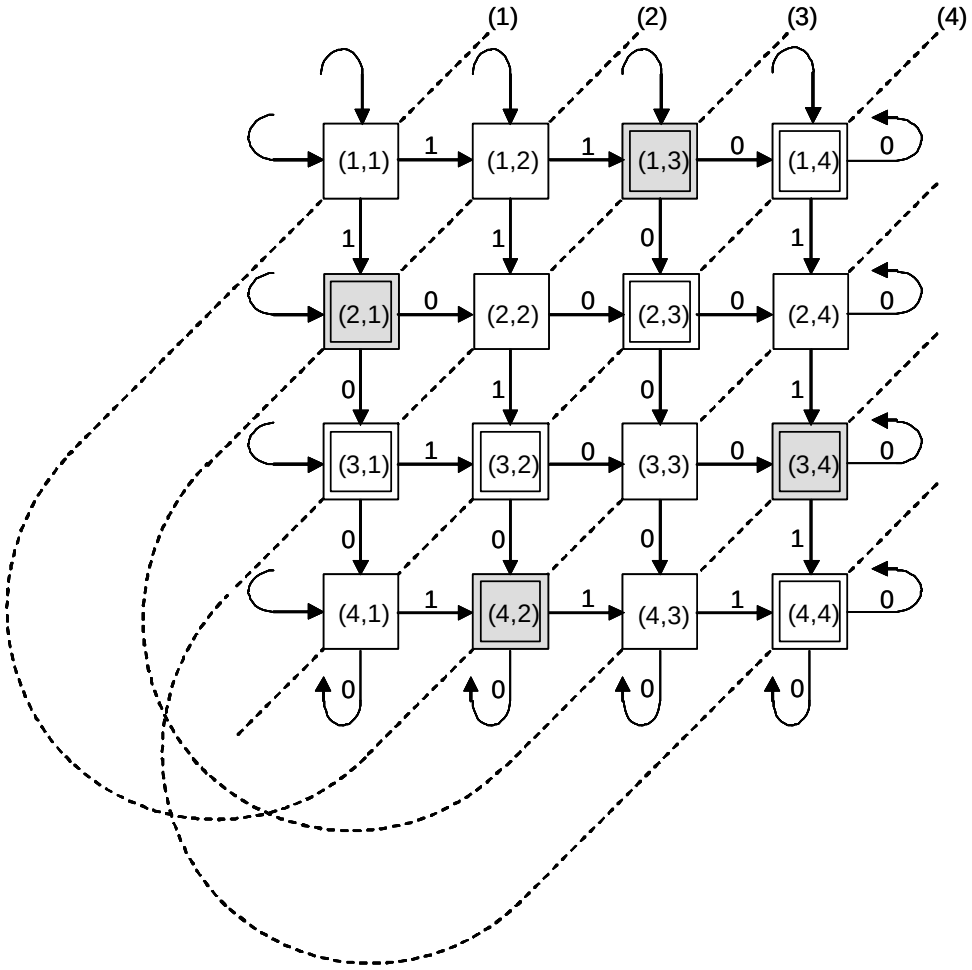

Figure 2.8 – Wrapped wave front arbiter

pattern matrix will be used in the different time slots. The pattern sequence matrix is intended to improve fairness among inputs and outputs by permuting the sequence of the generalized diagonals. Thus, 2DRR provides a fairness guarantee (bound) of *N* time slots for each of the VOQs.

2.6.3 Parallel matching algorithms

Parallel matching algorithms use request-grant-accept scheduling cycles to achieve matching between input ports and output ports. They can match multiple input and output port pairs in parallel and converge faster than sequential matching. Parallel matching algorithms can be classified as either non-iterative algorithms that terminate in a single iteration or as iterative algorithms that perform multiple iterations.

2.6.3.1 Parallel iterative matching

The first scheduling algorithm for IQ switches based on maximal matching is parallel iterative matching (PIM) [3]. For each iteration of PIM, three steps are executed as shown in Figure 2.9. A key characteristic of PIM is the random selection made by outputs and inputs in steps 2 and 3, respectively. This randomness guarantees that no VOQ is starved. Furthermore, PIM does not require previous state information to perform next maximal matching: no memory for storing previous input-output match information is needed.

1.(Request): Each unmatched input sends a request to every output for which it has a queued cell.

2.(Grant): If an unmatched output receives any requests, it randomly selects one to grant.

3.(Accept): If an input receives any grants, it randomly accepts one.

Figure 2.9 – PIM algorithm

PIM can only achieve a throughput of 63%, since the probability of an input remaining not granted is $\left(\frac{N-1}{N}\right)^N$, or $1-\frac{1}{e} \approx 63\%$ for a large N [3]. Higher throughput can be achieved by iterating the PIM algorithm. Each iteration of PIM achieves three-fourths of the remaining possible matches and the algorithm converges to a maximal match in O(log$N$) iterations [3]. Thus, it has been shown that running more than four iterations achieves no significant improvement for a 16 x 16 switch [3].

2.6.3.2 Statistical matching

Statistical matching [3], based on PIM, provides bandwidth guarantees between individual input-output pairs. Statistical matching performs iterative matching similar to PIM, with the exception that the request-grant-accept cycle is reduced to a grant-accept cycle. Statistical matching, in the grant stage, randomly selects one of the inputs to grant, proportional to bandwidth reservation. In the accept stage, an input receiving any grants 1) reinterprets the grants as zero or more virtual grants based on bandwidth allocation, 2) randomly selects a virtual grant, and 3) grants the output corresponding to the virtual grant. The drawback is that statistical matching can only allocate bandwidth up to 72%. This is due to not having a request step; thus, the grant pointer may point to an input that does not have a cell waiting to be transmitted to the output.

2.6.3.3 Weighted PIM

Weighted PIM (WPIM) [107] further improves bandwidth allocation support of PIM. WPIM allocates credits to each input-output pair based on their bandwidth requirements

in every frame (a time slot consisting of an integer multiple of cell-transmission time) and supports both connection-level and flow-level bandwidth allocation within a frame. The second stage of the request-grant-accept cycle of PIM is sub-divided into two stages as shown in Figure 2.10.

WPIM eliminates the 72% capacity limitation of the statistical matching algorithm. The problem with PIM and its variants is that they require a very fast generation of random numbers, which is considered impractical.

2.1.(Mask): If an unmatched output receives any requests, it generates a mask consisting of one bit per request as follows: the mask bit is set to 1 for each input that has transmitted at least as many packets as is their credit to the output port in the current frame. Otherwise, the mask bit is set to 0.

2.2.(Grant): An unmatched output randomly grants a request with a mask bit of 0 (unmasked).

Figure 2.10 – Grant stage of WPIM

2.6.3.4 Round-robin matching

Round-robin matching (RRM) is designed to improve on two aspects of PIM: computation complexity and fairness [71]. The RRM algorithm uses priority encoder-based round-robin (RR) arbiters that are much simpler than the random arbiters used by PIM [71]. Furthermore, RRM, with a cyclic property of the RR arbiters, results in fairer scheduling than PIM. The three steps in RRM are shown in Figure 2.11.

Accept and grant counters are maintained in each input and output port, respectively. RRM grant arbiters tend to synchronize, and multiple arbiters tend to grant to the same

input. As a result, RRM can only achieve a 63% throughput for uniform i.i.d. Bernoulli arrivals. This poor throughput is due to the synchronization effect of grant arbiters [71].

1.(Request): Each unmatched input sends a request to every output for which it has a queued cell.

2.(Grant): If an unmatched output receives any requests, it grants the one that appears next in the round robin, starting from the one indicated by the grant pointer of the grant arbiter. The grant pointer is then updated to modulo *N* to one position next to the granted input.

3.(Accept): If an input receives any grants, it accepts the one that appears next in the round robin, starting from the one indicated by the accept pointer of the accept arbiter. The accept pointer is then updated to modulo *N* to one position next to the granted output.

Figure 2.11 – RRM algorithm

2.6.3.5 SLIP

SLIP [71] is designed to improve RRM by reducing the degree of synchronicity of the grant arbiters. SLIP achieves a randomized matching under high utilization due to a "slip" between counters (hence the name: SLIP). SLIP is based on RRM providing grants and accepts in round robin sequences. The only difference between RRM and SLIP is that SLIP updates the grant counters differently. In SLIP, a grant counter is updated only if the granted path is accepted by the corresponding input port. The request-grant-accept cycle of the SLIP is shown in Figure 2.12. An example of the three steps of SLIP is shown in Figure 2.13. Due to the round robin sequences, SLIP guarantees to fairly provide connections to each input-output combination. Furthermore,

SLIP can achieve almost a 100% throughput under uniform i.i.d. Bernoulli arrivals due to the slip effects achieving a low degree of synchronicity of the grant arbiters.

1.(Request): Each unmatched input sends a request to every output for which it has a queued cell.

2.(Grant): If an unmatched output receives any requests, it grants the one that appears next in the round robin, starting from the one indicated by the grant pointer of the grant arbiter. The grant pointer is updated to modulo *N* to one position next to the granted input *if and only if the grant is accepted in step 3*.

3.(Accept): If an input receives any grants, it accepts the one that appears next in the round robin, starting from the one indicated by the accept pointer of the accept arbiter. The accept pointer is updated to modulo *N* to one position next to the granted output.

Figure 2.12 – SLIP algorithm

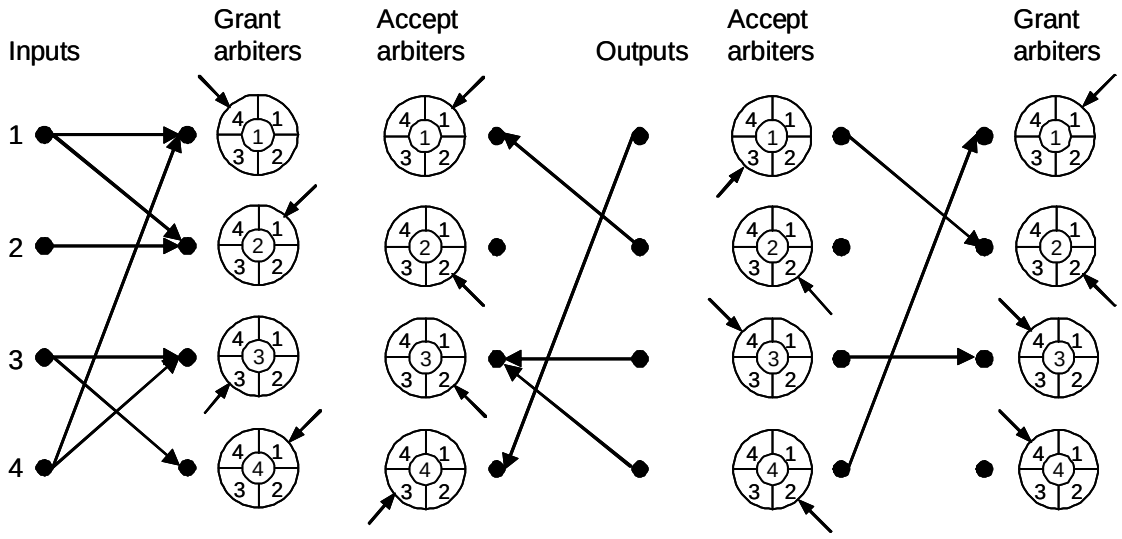

Figure 2.13 – SLIP with request and grant arbiters

2.6.3.6 Iterative SLIP

SLIP can be iterated to improve its performance [71]. Iterative SLIP (iSLIP) involves three steps as shown in Figure 2.14. iSLIP is used on Tiny-Tera developed at Stanford University [74]; and ESLIP, a variation of iSLIP that can handle unicast and multicast scheduling, is used in Cisco backplane routers [70].

1.(Request): Each unmatched input sends a request to every output for which it has a queued cell.

2.(Grant): If an unmatched output receives any requests, it grants the one that appears next in the round robin, starting from the one indicated by the grant pointer of the grant arbiter. The grant pointer is updated to modulo *N* to one position next to the granted input *if and only if the grant is accepted in step 3 of the first iteration*.

3.(Accept): If an input receives any grants, it accepts the one that appears next in the round robin, starting from the one indicated by the accept pointer of the accept arbiter. The accept pointer is updated to modulo *N* to one position next to the granted output only if the input has accepted the grant in the first iteration.

Figure 2.14 – iSLIP algorithm

2.6.3.7 FIRM

Significant research has been done to improve the performance of iSLIP, resulting in several variations. FIRM (FCFS in round-robin matching) [102] is identical to iSLIP except for a difference in the round-robin policy at the output ports (grant stage). In the grant stage of FIRM, if an unmatched output receives any requests, it grants the one that appears next in the round robin, starting from the one indicated by the grant pointer of the

grant arbiter. The grant pointer is updated to modulo *N,* to one position next to the granted input if and only if the grant is accepted in the following step (the accept stage). If a grant is not accepted, the grant pointer is advanced to point to the granted input. This results in a fairer scheduling than is available with iSLIP, by better approximating first-come first-served (FCFS). FIRM also reduces the maximum waiting time for any of the VOQ HOL cells from $(N-1)^2 + N^2$ for SLIP to N^2 for FIRM.

2.6.3.8 Shakeup techniques

Shakeup (randomization) techniques, used in conjunction with other matching algorithms, have been studied [32]. These techniques assume an initial bipartite graph matching generated through other matching algorithms. Each unmatched vertex in *I* is allowed to establish a match for itself even if it may "knock-out" any existing matching. The idea is to help a scheduler to escape from a local maximum solution [32]. Shakeup techniques can be performed after various existing maximal matching algorithms to improve their performance. For instance, simulation evaluations show that the shakeup techniques improve both PIM and iSLIP for uniform, non-uniform, and bursty traffic.

2.6.3.9 Dual round-robin matching

Dual round-robin matching (DRRM) [12] eliminates the accept stage requiring only request and grant stages, thus making the scheduler both faster and simpler to implement. DRRM operates as shown in Figure 2.15. Unlike RRM, SLIP, and its variations, only one request per input is made during step 1, and each input receives at most one grant. Thus, step 3 (the accept stage) is not needed. DRRM is shown to have a lower delay than

iSLIP for non-uniformity on the output side [48]. In general, 3-stage schedulers (with request-grant-accept cycles) outperform 2-stage schedulers (with request-grant cycles) for non-uniformity on the input side, while the latter outperforms the former in non-uniformity on the output side [48]. DRRM is implemented on the SATURN switch [13].

1.(Request): Each unmatched input sends a *single* request to an output that appears next in the round robin, starting from the one indicated by the request pointer of the request arbiter. The request pointer is updated to modulo *N* to one position next to the requested output if and only if the request is granted in step 2.

2.(Grant): If an unmatched output receives any requests, it grants the one that appears next in the round robin, starting from the one indicated by the grant pointer of the grant arbiter. The grant pointer is updated to modulo *N* to one position next to the granted input.

Figure 2.15 – DRRM algorithm

2.6.3.10 Load-balancing Birkhoff-von Neumann switch

An entirely new approach using a two-stage switch with a single stage buffer was introduced in [11]. A load-balancing switch is followed by an input-buffered Birkhoff von Neumann switch that performs switching for load balanced traffic as shown in Figure 2.16. At the first stage, the load balancing switch produces uniform traffic. At the second stage, the capacity decomposition is performed. The decomposition approach reduces the two-dimensional rate assignment problem to a one-dimensional problem so that the Packetized Generalized Processor Sharing (PGPS) algorithm [92] (the Weighted Fair Queueing (WFQ) in [28]) can be applied. The load-balancing Birkhoff and von Neumann switch provides uniform service guarantees for all non-uniform traffic.

Specifically, if the two conditions (1) and (2) are true (if input traffic is a schedulable flow), then there exists a scheduling algorithm, such that $C_{i,j}(t) - C_{i,j}(s) \geq \lambda_{i,j}(t-s) - s_{i,j}$ for $\forall i, j, s \leq t$, and some $s_{i,j} \leq N^2 - 2N + 2$ [11]. It was also proven that load-balancing Birkhoff and von Neumann switches can achieve a 100% throughput without internal speed-up [11].

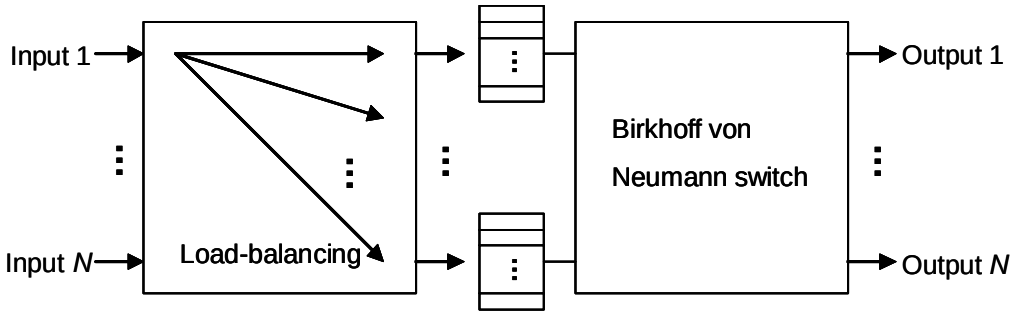

Figure 2.16 – Load balancing Birkhoff-von Neumann switch

2.7 Combined input and output queued switches

Combined input and output queued (CIOQ) switches provide buffering at both the input ports and output ports as shown in Figure 2.17. CIOQ switches improve the performance of IQ switches by having output buffers with a modest speed-up. CIOQ switches were first studied in the late 1980's and were shown to improve maximum throughput over IQ switches with a limited speed-up at output ports [88]. Subsequently, it was found that the performance limiting factor was the HOL blocking for an output speed-up greater than three [37]. The advent of VOQ eliminated the HOL blocking of the CIOQ switch (as with IQ switch). It was shown that $(N/2)$x output speed-up is sufficient to exactly emulate an OQ switch with a VOQ CIOQ switch [73]. Further

studies demonstrated that a 2x output speed-up is sufficient to exactly emulate an OQ switch with a VOQ CIOQ switch [56], [108], [20].

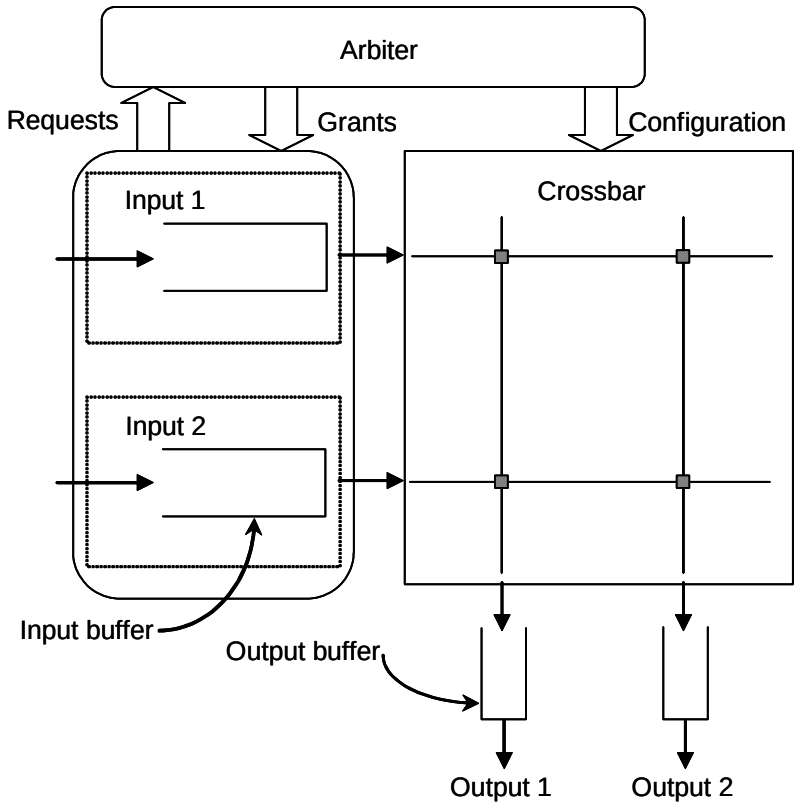

Figure 2.17 – Combined input and output queued switch

2.8 Crossbar queued switches

Both VOQ IQ and VOQ CIOQ switch architectures scale-up to very high speeds and have been the subject of intense research in the past decade. These VOQ switches require centralized switch matrix scheduling algorithms to match input ports to output ports. These scheduling algorithms are currently one of the bottlenecks in the process. A

buffered crossbar switch, as described in this section, is based on distributed schedulers and scales to large switch sizes and link data rates.

Buffered crossbars go back to a 1982 patent [6]. In 1987, a physically large multi-cabinet buffered crossbar was used by Nojima [85] to implement a bus matrix switch (BMX). The BMX uses a cross-point (CP) buffer as a packet queueing medium as shown in Figure 2.18. CP buffers are implemented with dual port memories allowing asynchronous operation among input and output ports. Thus, parallel operations of packet switching on each bus as well as variable-length packet switching are possible. By increasing the number of buses, the BMX can increase the switch capacity by a factor of 16 (160-Mbps to 2.6-Gbps for CMOS), and simulation results confirm that the switching delay is independent of the number of buses.

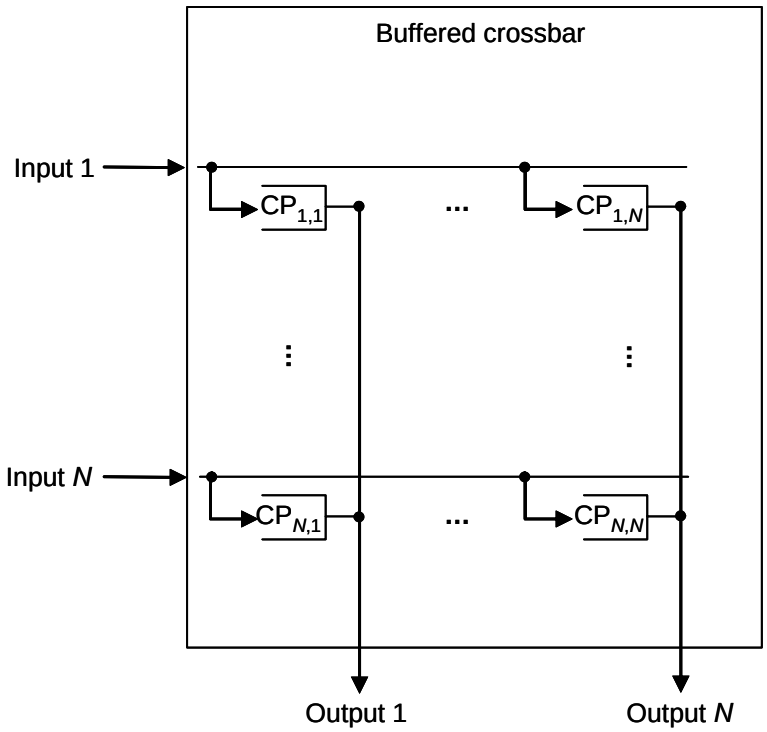

Figure 2.18 – Buffered crossbar switch

Multi-stage crossbar switches with buffered 2x2 crossbar stages have been studied in [54], [31], [125], and others. A multi-stage self-routing (MSSR) switch proposed in [31] is realized by connecting self-routing switch modules (SRM) in a three stage configuration that is conceptually equivalent to a single stage crosspoint buffered switch. Multiple routes between the first stage SRM and the third stage SRM result in efficient routing. As stated earlier, this dissertation focuses on single stage buffered crossbar switches.

A CP buffered only switch performs better than IQ and OQ switches without speedup because no internal collisions occur within the CP buffered only switch, given an infinite CP buffer size [94]. It is clearly seen that the buffered crossbar with an infinite CP buffer size is equivalent to an OQ switch having dedicated memory for packets from each input.

By limiting the CP buffer size to 53-bytes (ATM cell size) and by having a sufficient input buffer size, a CICQ switch significantly reduces the total memory size required [39]. This CICQ switch, with selection based on HOL blocking, achieves an 87.5% throughput under uniform traffic for a 16x16 switch [39]. This work was extended to support two levels of delay-dependent priority classes, which resulted in an increase in throughput from 87.5% to 91% [40]. The improvement was due to the preemption of low priority packets from the HOL position by the arrival of high priority packets. The CICQ switch is further studied in [29] and [95]. It has been proven that throughput for a CP buffered switch with FIFO and random (RND) selection policy can approach 100% throughput [95].

A switch that supports QoS and variable-length packets was developed and evaluated [106]. This switch has both buffering and packet fair queueing servers [28] within the

input ports, crossbars, and output ports. A speed-up of slightly less than 2x is needed to support variable-length packets.

2.9 VOQ CICQ switches

In 2000, the first VOQ CICQ switch was proposed by Nabeshima [84]. The VOQ CICQ switch significantly reduced the amount of buffering in crossbar. The HOL cell with the longest delay is selected for both the input port (by a polling of all VOQs) and the CP buffer (by a polling of all CP buffers) (see Figure 2.19). The VOQ CICQ switch entirely eliminates HOL blocking, and it has a lower mean cell delay than a pure IQ switch above 65% loads. A popular textbook belief is that crossbar switch fabrics are limited in scalability by an N^2 increase in cross points. In fact, scalability is not limited by the transistors required for N^2 cross points, but, instead, by pin count of an integrated circuit (IC). As VLSI density has continued to increase, it is now feasible to implement small amounts of buffering at each cross point in a crossbar [46], [104], [97].

Xilinx implements a buffered crossbar in FPGA technology for its Virtex-Extended Memory (Virtex-EM) devices [104]. In 2000, the commercially available Xilinx XCV812E device contained over 1.12 Mbytes of block RAM, enabling 4Kbytes of buffer space for each cross point of a 16x16 switch. This is the emergence of the modern CICQ switch, which, when coupled with VOQs at input port buffers, is the switch architecture for the future. For a cell-based switch, the buffering at each cross point is sufficient to hold one cell. The CP buffer occupancy status is reported from each crossbar row to its input port where an independent RR selection of VOQs is made for the next available CP. The buffer occupancy status must be reported at a rate equal to the maximum cell

transmission rate, which can be done asynchronously for each input port. No communication of state is necessary between output and input ports.

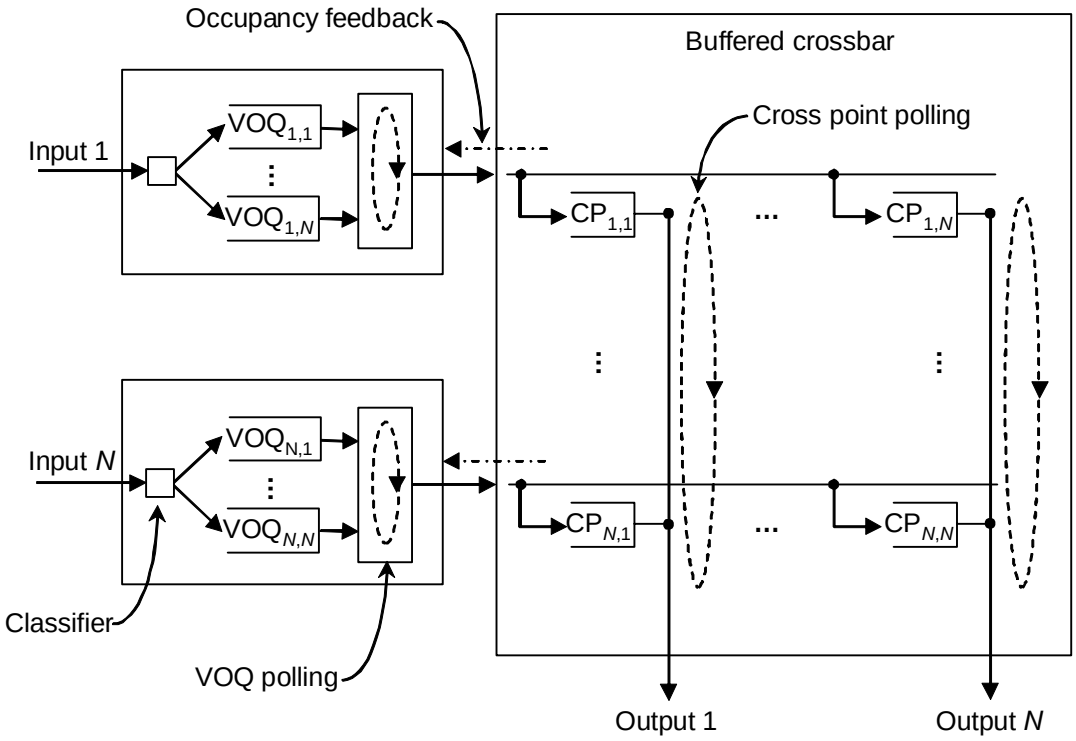

Figure 2.19 – VOQ CICQ switch

A VOQ CICQ switch can have round robin polling of the VOQs at the input ports and round robin polling of the CP buffers [119], and is thus a RR/RR CICQ switch (referred to as a CICQ switch throughout the remainder of this dissertation). The CICQ switch in [119] natively supports variable-length packets: it does not require complex packet segmentation or reassembly mechanisms. It was shown that the CICQ switch has a lower delay than a VOQ IQ switch with iSLIP for both cell and packet switching under uniform traffic [119].

A Combined Input-One-cell-CP Buffer crossbar (CIXB-1) with VOQs at the inputs and round-robin arbitration was shown to achieve 100% throughput under uniform traffic [97]. The mean delay of CIXB-1 was proportional to burst length and very close to that of an OQ switch. A Combined Input-CP-Output Buffered (CIXOB-*k*, where *k* is the size of the CP buffer) with VOQs at the inputs and round-robin arbitration requires buffers at each input, output, and CP [98]. A CIXOB-k switch improved CIXB-1 to achieve 100% throughput under uniform as well as non-uniform traffic. A full-scale system design of a terabit switch incorporating ideas for the CIXOB-k switch architecture is described in [13]. Scheduling algorithms for the VOQ CICQ switch are investigated in [41], [78], [93].

Chapter 3: Performance Evaluation of the CICQ Switch

This chapter describes the evaluation of the CICQ switch via simulation modeling. The simulation model, traffic input, experiments, and results are covered.

3.1 The simulation model

A discrete-event queueing model of single-stage switches was built using CSIM18 [101]. CSIM18 is a process-oriented discrete event simulation function library for C and C++. It maintains a linked list of "events" in simulated time order. A call to a CSIM function generates a CSIM process that models the active elements of a system. For instance, a VOQ arbiter, CP arbiter, and packet can all be represented by a CSIM process. A CSIM process can be in an *active*, *holding*, or *waiting* state. Only in an *active* state can the process be executed; otherwise, the process has to wait in the *holding* state for a period of time to elapse, or it has to wait in the *waiting* state for an event to occur.

3.1.1 Switch model

The CICQ, VOQ IQ with iSLIP scheduling algorithm, OQ with FIFO schedulers, and their variants are all implemented at the system level. Figure 3.1 shows a list of the modules, each with a list of the CSIM functions developed for modeling the CICQ switch. Mapping of the modules to the CICQ switch architecture is shown in Figure 3.2.

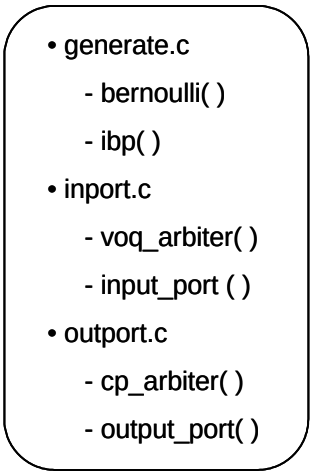

Figure 3.1 – List of modules and functions for CICQ switch model

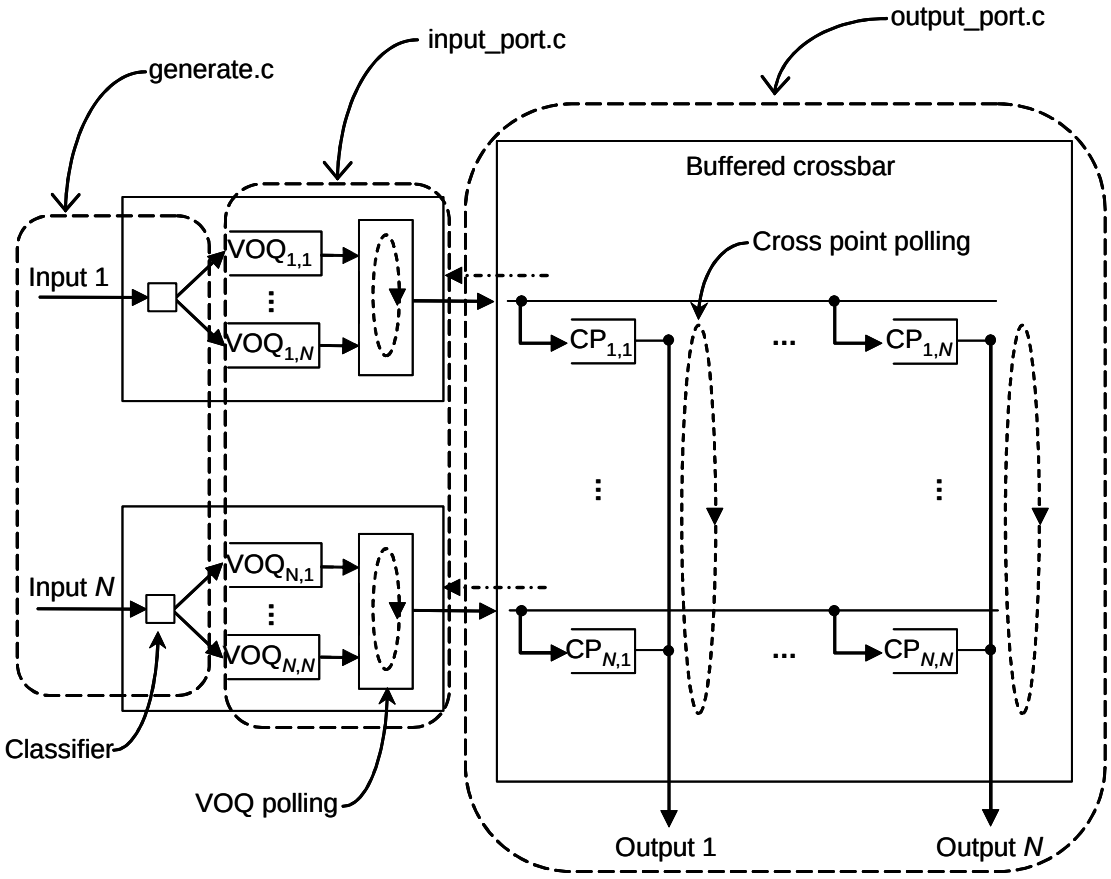

Figure 3.2 – CICQ switch model

A sequence of processes to represent cell or packet arrivals to the input ports of the switch is created using a traffic generator: generate.c. This module includes a bernoulli( ) function for Bernoulli arrival and an ibp( ) function for Interrupted Bernoulli Process arrival of cells (details of these probability distributions are described in the next section). Figure 3.3 shows a CSIM code for bernoulli( ) function. A bernoulli_arrival process is first created by create('bernoulli arrival") (line 7). In every iteration of the loop, simulation time is incremented by hold(CELL_TIME) (line 13). Based on a random variable generated by uniform( ) (line 16), a cell arrival to an input port is generated (line 17-line 23).

```
1. void bernoulli (int in_id)
2. {
3.   double z;              // Uniform RV from 0.0 to 1.0
4.   double org_time;   // Origination time of a cell
5.   int    out_id;         // Destination outport number
6.
7.   create("bernoulli_arrival");
8.
9.   // Do forever
10.  while(TRUE)
11.  {
12.    // Hold for one cell time
13.    hold(CELL_TIME);
14.
15.    // Determine if there is to be a cell in this slot
16.    z = uniform(0.0, 1.0);
17.    if (z <= Lambda)
18.    {
19.      // Generate output port id and origination time
20.      out_id = random_int(0, N - 1);
21.      org_time = clock;
22.      in[in_id](in_id, out_id, org_time);
23.    }
24.  } // end of while loop
25. }
```

Figure 3.3 – Source code for bernoulli ( )

Each cell or packet process consists of a source (input port) id, a destination (output port) id, and an arrival time that are set upon process creation. These cell or packet processes are controlled by a series of events inside the input port module, input_port.c, and the switch fabric and output port module, output_port.c. When they depart from the output ports, simulated internal switch delay times are computed and recorded for later statistics.

Multi-level priority support was also implemented in the iSLIP and CICQ switch models. For the iSLIP switch, no priority is assumed for the transmission of packets from the output queues (iSLIP speed-up results in queueing, and hence buffer requirements, at the output ports). For priority support in a CICQ switch, an RR poller per priority queue is implemented. All high priority VOQs and CP buffers are serviced before any low priority VOQs or CP buffers are serviced.

For validation purpose, performance of CICQ, IQ, and OQ switch models are carefully compared with simulation results in [72], [32], [46]. The simulation model is available from [15] and requires CSIM18 libraries [101].

3.1.2 Stopping criteria

Two stopping criteria were used in the simulation experiments. Some experiments were run for a fixed number of cells or packets. Other experiments were run until a specified accuracy level was achieved with a 95% confidence interval. CSIM18 has a built-in run length control algorithm that monitors simulation statistics and terminates simulation when desired statistical conditions are achieved. In this chapter, all simulation experiments were run until a 2% accuracy was achieved, unless otherwise stated.

3.2 Traffic models for evaluating the CICQ switch

Theoretical and empirical probability distributions were used to generate packet arrivals to the simulation. The theoretical probability distributions used to generate traffic are a Bernoulli arrival process and an Interrupted Bernoulli Process (IBP). The Bernoulli model is a common traffic model for evaluating switch performance [3], [71], [32], [46], [76]. An IBP arrival process is used to approximate the bursty nature of packet switched traffic [119], [86], [93].

3.2.1 Bernoulli and Interrupted Bernoulli Process arrival processes

For fixed-length cell traffic, Bernoulli and Interrupted Bernoulli Process (IBP) arrival processes are used. At most, one cell arrival occurs at each port during a time slot as shown in Figure 3.4. For Bernoulli traffic, the output ports are uniformly selected for each cell. For IBP traffic,

$$\alpha = \Pr[\text{arrival at } t \mid \text{IBP is in on state}], \tag{3.1}$$

$$p = \Pr[\text{IBP is in on state at } t+1 \mid \text{IBP is in on state at } t], \text{ and} \tag{3.2}$$

$$q = \Pr[\text{IBP is in off state at } t+1 \mid \text{IBP is in off state at } t]. \tag{3.3}$$

The mean length of an on state is $1/(1-p)$ and the mean length of an off state is $1/(1-q)$. In an off state, there are no arrivals. An off state is at least one slot in length. The mean arrival rate or offered load, ρ, is

$$\rho = \frac{\alpha(1-q)}{2-p-q}, \tag{3.4}$$

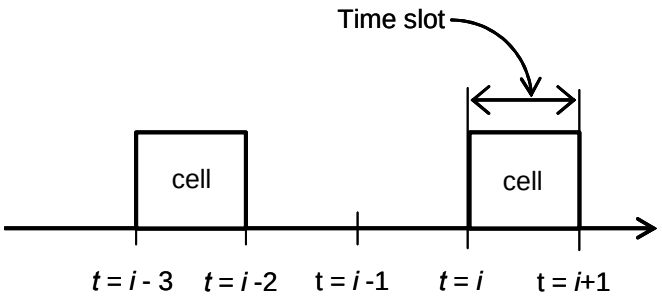

Figure 3.4 – Cell arrivals in time slot

and the Coefficient of Variation (CoV*)* is

$$\mathrm{CoV} = 1 + \alpha\left(\frac{(1-p)(p+q)}{(2-p-q)^2} - 1\right). \tag{3.5}$$

In this dissertation, α is always set to 1.0, so that traffic is generated at line data rate in the ON state, while no traffic is generated in the OFF state. This condition is better described as a two-state Markov chain as shown in Figure 3.5. The parameters *p* and *q* are varied to achieve a desired CoV and offered load. All packets in a burst are destined to the same output port, which is uniformly selected at the start of the burst.

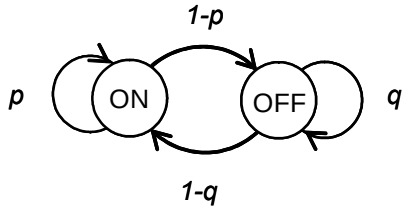

Figure 3.5 – Two-state Markov chain

3.2.2 USF synthetic traffic

For variable-length packet traffic, Poisson arrivals with uniformly selected output ports are used. Packet lengths are independently pulled from an empirical "USF distribution" based on over 5 million packets collected during the middle of a day in November 2001 at the University of South Florida Gigabit Ethernet backbone (USF traced traffic #1). Figure 3.6 shows the packet length histogram where all packet lengths from 64 to 1518 bytes are represented. The mean length is 364.7 bytes. The most common packet length is 64 bytes (with 41.5%) followed by 1518 bytes (8.2%), 558 bytes (7.0%), 90 bytes (5.9%), and 570 bytes (5.5%). All other packet lengths occur at less than 2.5%. Using this real packet length distribution allows for speed-up issues in an iSLIP switch to be studied.

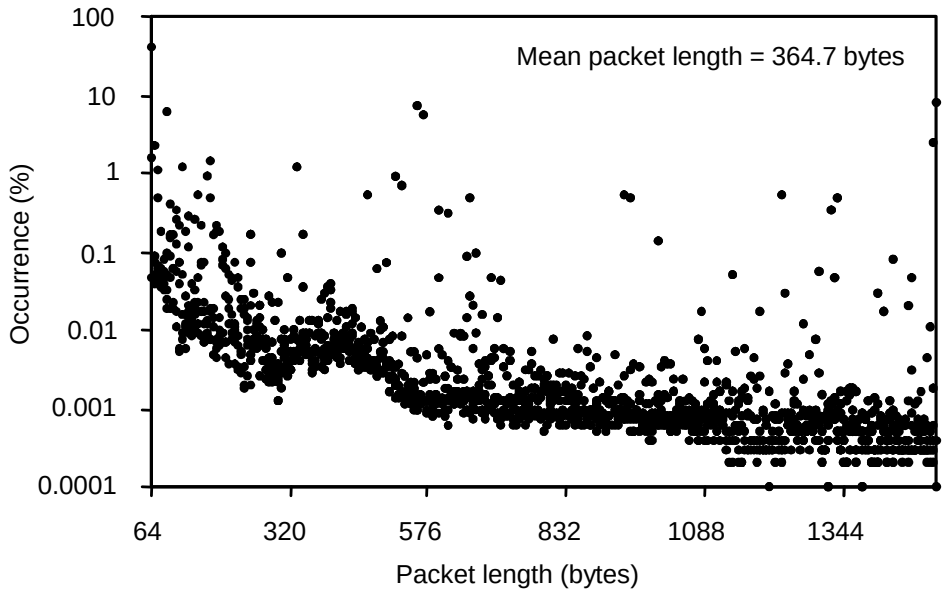

Figure 3.6 – Histogram of "USF distribution" of Ethernet packet lengths

3.3 Simulation experiments

For all simulation experiments, the mean and standard deviation of switch delay, or response time is measured. Switch queueing delay is the performance criteria (switch queueing delay plus cell or packet transmission time). For cell traffic, response time is in cell times, and each CP buffer size is set to hold one cell. For variable length packet traffic, the link data rate is assumed to be 10 Gbps, response times are in microseconds, and each CP buffer size is set to 1518 bytes. The performance of infinite buffer size, 16-port CICQ, iSLIP (for four iterations), and output buffered switches are compared. The delay performance of the OQ switch serves as a lower bound on response time. The experiments are as follows:

1) *Bernoulli experiment:* Bernoulli arrival of cells with uniformly selected outputs. Offered load is ranged from 50% to 98%. This is the "classic" experiment for evaluating switch performance and also serves as a validation of the iSLIP switch model with the simulation results given in [69].
2) *IBP experiment:* IBP arrivals of cells with uniformly selected outputs for bursts (i.e., for on periods) of cells. The CoV is fixed at 2.0 and the p and q values solved (using Eq. (3.1) and (3.2)) for offered loads from 50% to 90%. For 50% offered load, the mean on and off periods are 5 cell times each. For 90% offered load, the mean on period is 105 cells, and the mean off period is 11.67 cell times. This experiment evaluates switch performance for bursty cell traffic.
3) *Packet experiment:* Poisson arrivals of variable length "USF distribution" packets with uniformly selected outputs. The offered load is varied from 50% to 98%. A CICQ switch is compared with an iSLIP switch with packet segmentation and re-

assembly, and with an output buffered switch. For the iSLIP switch, an internal cell size of 64 bytes is used with no speed-up, 1.05x speed-up, and 2x speed-up of memories and crossbar. A 3% accuracy (95% confidence interval) was used as the simulation stopping criterion to achieve acceptable simulation run times.

4) *Fairness experiment:* To test fairness and traffic isolation in VOQ packet switches, the performance of the iSLIP and CICQ switch with two-level priority support was evaluated. A periodic constant bit rate (CBR) stream of 10% offered load of 1500 byte packets was sent to port 0 and destined to port 0. An interfering load of 50% to 88% of low (no) priority Poisson arrivals were sent to ports 0 through 15. This Poisson stream was destined uniformly for ports 0 to 15 and had packet lengths pulled from the "USF distribution". The periodic stream is the modeled real-time traffic and is measured for cases, 1) with no priority, and 2) with priority. This experiment tests fairness and traffic isolation in VOQ packet switches. To evaluate the performance of the priority implementations, infinite buffer size, 16-port CICQ, and iSLIP packet switches were similarly modeled. The speed-up of the iSLIP switch was set to 1.05x.

3.4 Experiment results

Figure 3.7 shows the mean response time results for the Bernoulli experiment. The mean response time for CICQ is lower than that of iSLIP for offered loads greater than 75%. The iSLIP and output buffered switch results exactly match the results shown in Figure 10 of [69], serving as a validation of the iSLIP and output buffered switch models. At lower offered loads, the two store-and-forward operations within the CICQ switch

dominate the response time (the crossbar buffering was implemented as store-and-forward memory in the simulation model). If one store-and-forward delay is removed, the series shown with the dotted line is achieved and the response time is less than that of an iSLIP switch for all offered loads. The output buffered switch remains as a lower

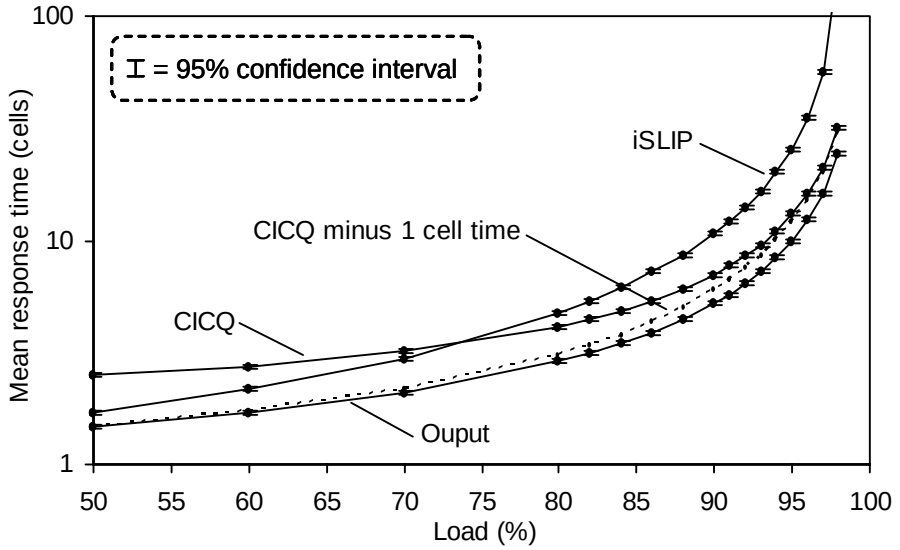

Figure 3.7 – Results for the Bernoulli experiment (mean response time)

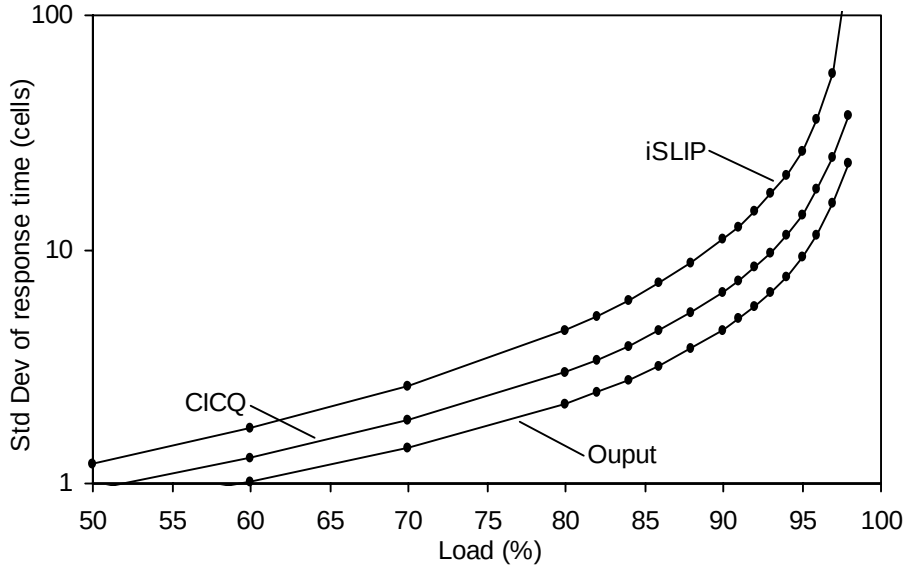

Figure 3.8 – Results for the Bernoulli experiment (std dev of response time)

bound to delay. The ordering for standard deviation of response time are the same as the mean response times, except that CICQ is always less than iSLIP as shown in Figure 3.8.

Figure 3.9 shows the mean response time results for the IBP experiment. iSLIP has lower delay than CICQ at low loads while they have roughly similar delay at high loads. The order for standard deviation of response time is the same as for the mean response time as shown in Figure 3.10.

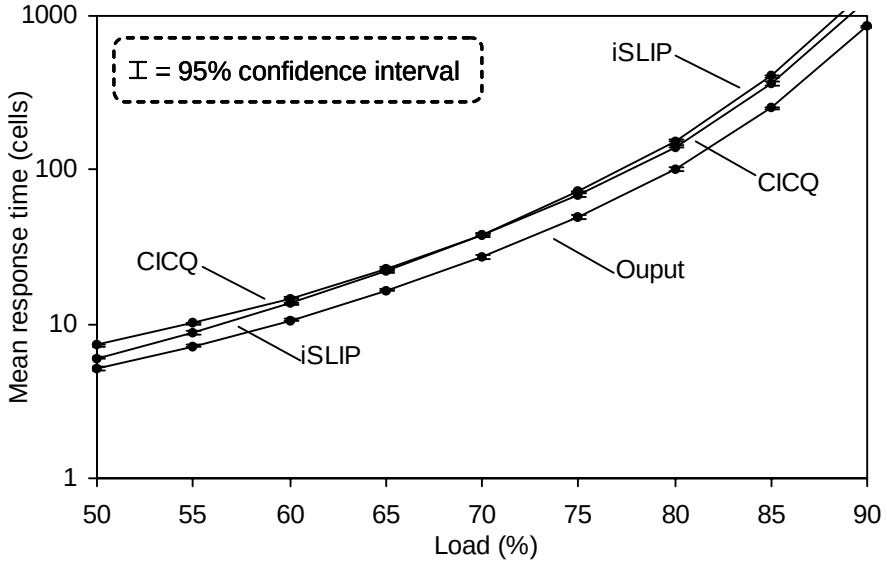

Figure 3.9 – Results for IBP experiment (mean response time)

Figure 3.11 shows the mean response time results for the packet experiment. For the iSLIP switch without speed-up, a 98% offered load cannot be carried. With a mean packet length of 364.7 bytes, six 64-byte cells of total 384 bytes are needed for 5% overhead. It can be seen that internal speed-up is needed to achieve full throughput. The results show that CICQ is better than iSLIP, with no speed-up and 1.05x speed-up.

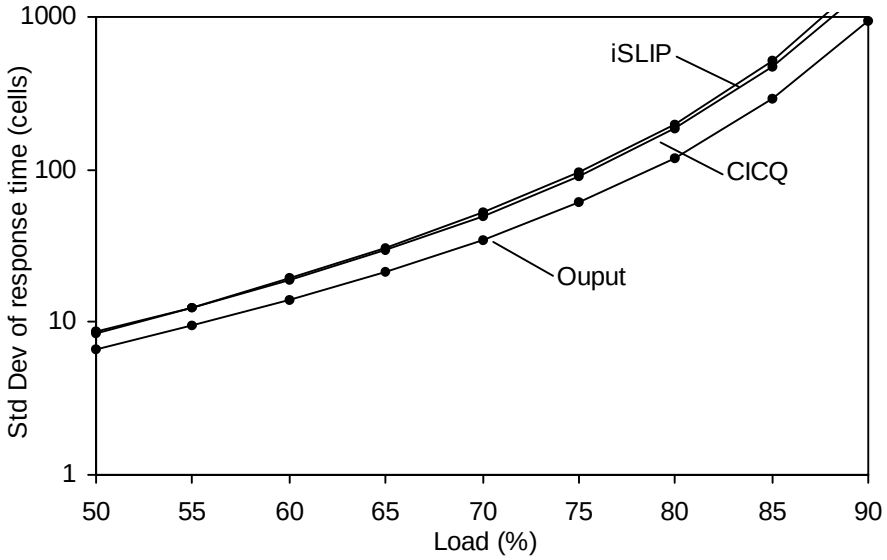

Figure 3.10 – Results for IBP experiment (std dev of waiting time)

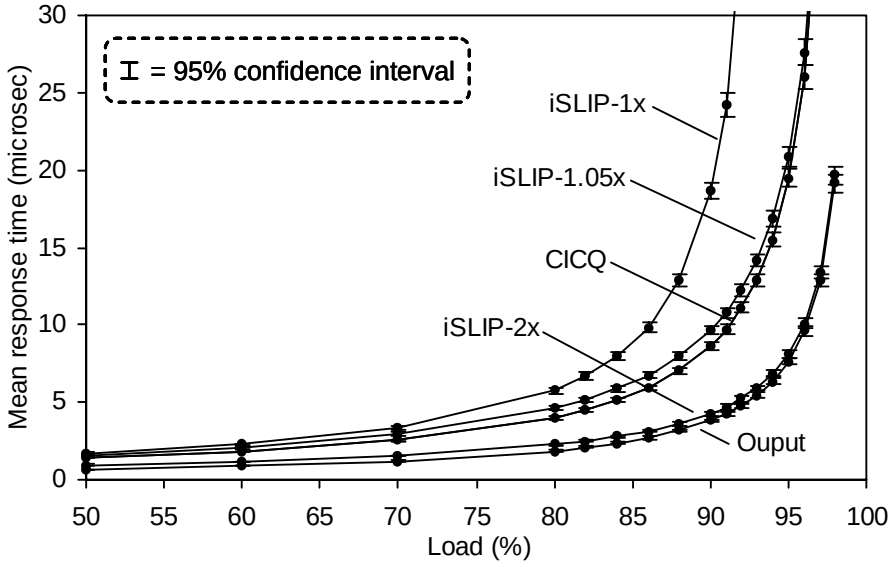

Figure 3.11 – Results for packet experiment (mean response time)

For 2x speed-up, iSLIP is almost identical to an output buffered switch. For CICQ compared to output buffered, at 50% offered load there is an additional 0.7 microseconds of delay and at 95% an additional 11.8 microseconds. At 10-Gbps the transmission time

for a 1500 byte packet is 1.2 microseconds. The order for standard deviation of response time are the same as for the mean response time as shown in Figure 3.12.

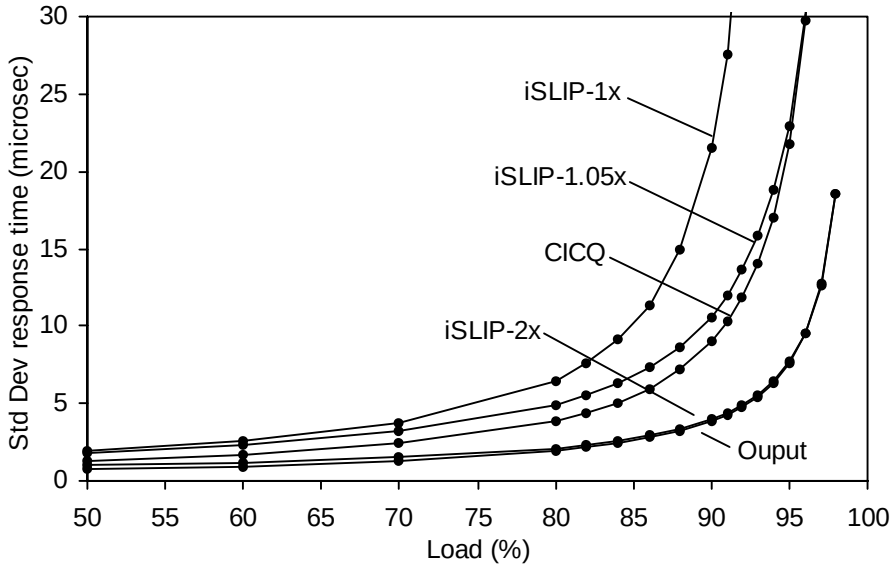

Figure 3.12 – Results for packet experiment (std dev of waiting time)

Figures 3.13 and 3.14 show the mean and standard deviation results for the fairness experiment, respectively. For case (1), the mean and standard deviation of response time for the CBR are very high. For case (2), low mean and standard deviation of delay is maintained even at very high offered loads. At 98% offered load and 0.5 seconds of simulated time, the maximum and 99% response time for the real-time priority stream for the iSLIP switch was 25.1 and 17.8 microseconds, respectively. For the CICQ switch, the maximum and 99% response times were 13.8 and 8.9 microseconds, respectively. A CICQ switch has lower response times for priority streams at high loads. This simple priority implementation for iSLIP and CICQ can starve low priority traffic.

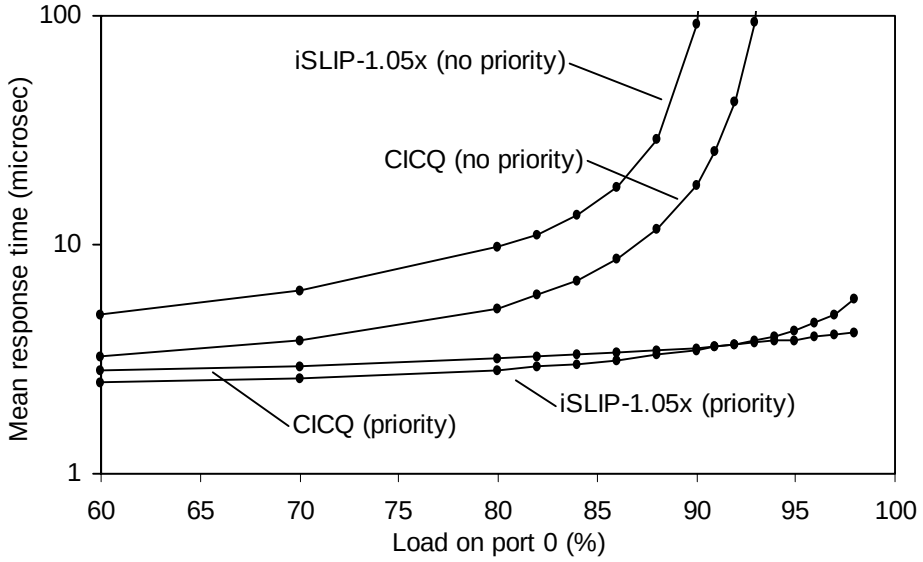

Figure 3.13 – Results for fairness experiment (mean response time)

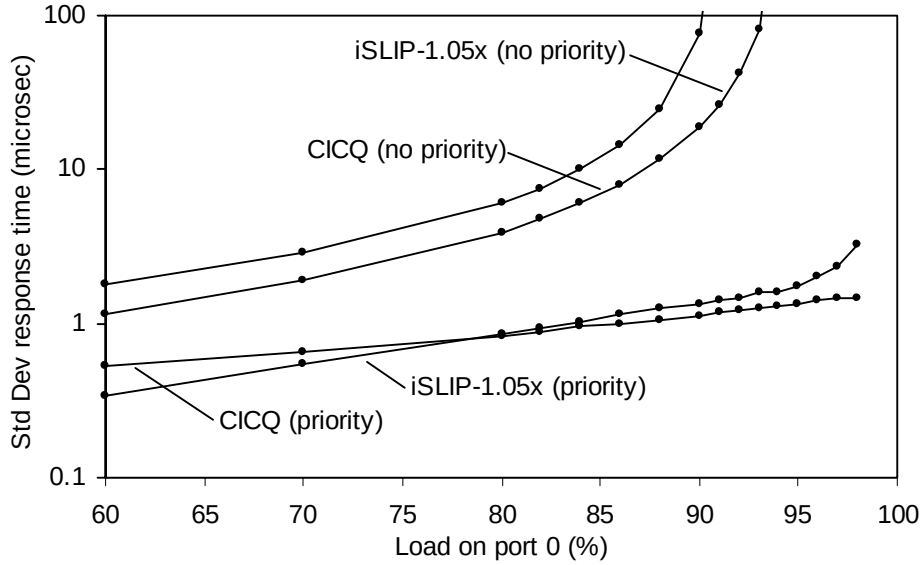

Figure 3.14 – Results for the fairness experiment (std dev of response time)

Chapter 4: Eliminating Instability in IQ and CICQ switches

A major issue in VOQ switch matrix scheduling is stability. Stability refers to bounded queue length for schedulable loads. IQ cell switches use iterative request-grant-accept scheduling cycles to achieve a maximal one-to-one matching. Existing scheduling algorithms for IQ cell switches based on an unweighted maximal matching (such as PIM [3] and iSLIP [71]) are not stable unless internal speed-up is used. A 2x speed-up has been proven to be sufficient for stability for all schedulable flows for CIOQ switches [20]. CICQ switches can use independent RR selection of VOQs and CP buffers; however, instability occurs unless OCF or LQF is used to select VOQs in an input port [46]. Both OCF and LQF require comparisons between all N ports during each scheduling cycle. This requires either N sequential comparisons or $\text{Log}_2(N)$ comparisons with a tree circuit containing $N-1$ comparators. Simpler methods [79], [99] for achieving stability are needed. This need is addressed in this chapter. Parts of this chapter resulted from a collaboration with Dr. Neil J. Gunther from Performance Dynamics Company. The instability region identified in a previous literature is described, and a method for achieving stability in this region is proposed and evaluated.

4.1 Unstable regions in VOQ switches

An unstable region in the iSLIP and CICQ switches is considered in this section. Bernoulli arrivals with rate λ_{ij} for $i, j = 1, \ldots, N$ where i is the input port number, j the

output port number, and $0 \le \lambda_{ij} \le 1.0$ are assumed. A Bernoulli model is a common traffic model for studying switch performance (e.g., as used in [3], [46], [72], [76]).

Definition 1. Let λ_1 be the offered load at port 1. The fraction f of offered traffic going to:

1) VOQ_{11} *is* $f\lambda_1 = \lambda_{11}$

2) VOQ_{12} *is* $(1-f)\lambda_1 = \lambda_{12}$

where $1/2 < f < 1$.

Corollary 1. $\lambda_1 = \lambda_{11} + \lambda_{12}$

Remark 1. Note that: $\lambda_{12} \neq 1 - \lambda_{11}$ *unless* $\lambda_1 = 1$. The mean interarrival time of cells at port 2 is $\tau_2 = \lambda_2^{-1}$. But $\lambda_{21} \equiv \lambda_2$ by virtue of $\lambda_{22} = 0$.

A region of instability for iSLIP IQ and CICQ switches for a schedulable, asymmetric traffic load to two ports is demonstrated [46]. For any two ports arbitrarily identified as ports 1 and 2, let $\lambda_1 = \lambda_{11} + \lambda_{12}$, $\lambda_{21} = \lambda_{12}$, and $\lambda_{22} = 0$. Within a region of $\lambda_{11} > 0.5$ and high offered traffic load, instability occurs. This instability condition is not limited to a two-port switch, but can occur between any two ports of a large switch. The offered load that causes instability for an CICQ switch ranges from a low of approximately 0.9 in the range of $0.6 < \lambda_{11} < 0.7$ to a high of 1.0 at $\lambda_{11} = 0.5$ and $\lambda_{11} = 1.0$. This instability exists for a switch of size *N* input and output ports where any two of the *N* ports have the traffic load specified above. The instability range for an iSLIP IQ switch is larger in area. The simulation model is developed and used to reproduce the instability results in [46]. Infinite size VOQ buffers are assumed for both an iSLIP IQ and CICQ switch. For the iSLIP IQ switch, four iterations are used per scheduling cycle.

As an experimental mean to detect instability, simulation experiments were run for 100 million cell times and terminated as unstable if any queue length exceeded 5000 cells. Five thousand cells are equivalent to 256 microseconds in drain time on a 10-Gbps link data rate and 320 Kbytes in size. A similar experimental means of detecting instability is used in [32]. Figure 4.1 shows the simulation results for the iSLIP and CICQ instability regions, which exactly match the results in [46]; the region for iSLIP is slightly larger than for the CICQ switch. For this same simulation experiment, OCF/RR and LQF/RR for a CICQ switch do not exhibit instability.

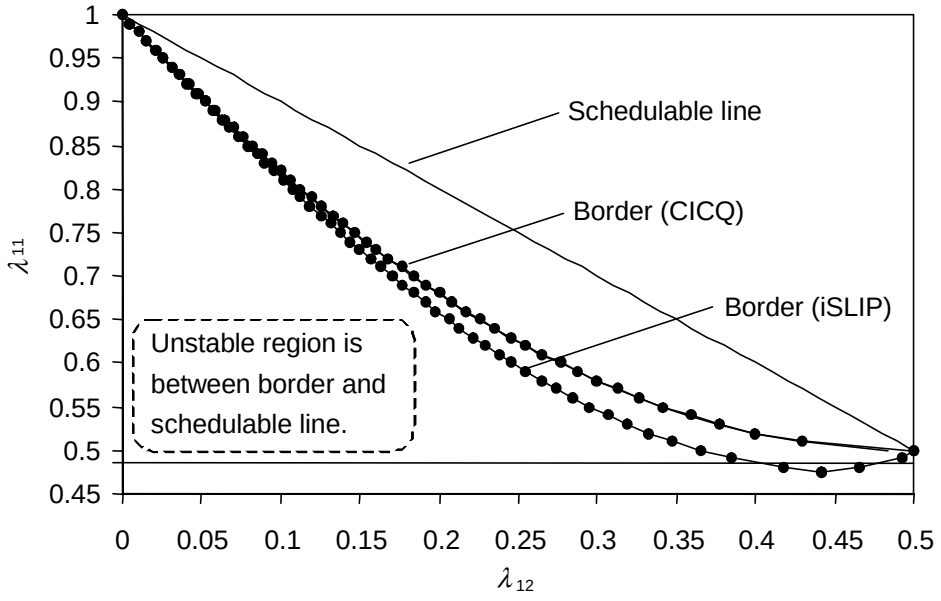

Figure 4.1 – Instability in CICQ and iSLIP

The instability in the CICQ switch is caused when VOQ_{12} is empty (drained) and VOQ_{11} is blocked from transferring a cell to its cross point buffer (CP_{11}) due to CP_{11} being already full and CP_{21} transferring to output 1. A solution is to service VOQ_{12} less

aggressively so that VOQ_{12} will have queued cells that can be transferred (to CP_{12}) when VOQ_{11} is blocked. In this case, work conservation of input port 1 can be maintained. Both OCF and LQF achieve stability by more aggressively draining VOQ_{11} than VOQ_{12} in this configuration. This observation was used to propose a burst stabilization protocol (described in Section 4.3) that does not require a comparison of state information between VOQs [120].

4.2 An Erlang space model for unstable region

Analysis was made to gain deeper insight of instability caused by the asymmetric traffic load to two ports. Consider each VOQ in Figure 2.5 (and Figure 2.19) as separate queues with polling suppressed. Let S_{ij} be the cell service time. The respective server utilization is then bounded by

$$\rho_{ij} = \lambda_{ij} E\{S_{ij}\} \le 1, \quad i, j = 1,2 \tag{4.1}$$

since $\lambda_{ij} \le 1$ and $\mu^{-1} = E\{S_{ij}\} = 1$ cell time. Similarly, the total capacity μ_i (in Erlangs) of port *i* is bounded $u_i \le 1$. This capacity conservation can be used to bind the region of CICQ stability in Erlang space (ρ_{ij}, ρ_{ii}) shown in Figure 4.1, although the traffic at port 1 is asymmetric $\lambda_{11} \ge \lambda_{12}$, $\rho_{11} \ge 1/2$ and $\rho_{12} \ge 1/2$ such that utilization is conserved across the two servers,

$$\left(\rho_{11} - \frac{1}{2} \right) = \left(\frac{1}{2} - \rho_{12} \right). \tag{4.2}$$

The trivial solution is

$$\rho_{11} = 1 - \rho_{12} \tag{4.3}$$

which corresponds to the linear boundary of the unstable region in Figure 4.1. Generalizing eq. (4.2) and noting that $\rho_{22} = 0$, capacity conservation across the servers in both ports 1 and 2 can be written as

$$\left(\rho_{11} - \frac{1}{2} \right) = \left(\frac{1}{2} - \rho_{12} \right)\left(\frac{1}{2} - \rho_{21} \right) + \left(\rho_{21} - \frac{1}{2} \right)\left(\rho_{12} - \frac{1}{2} \right) \tag{4.4}$$

Since the indices can be permuted, the following simplification ensues:

$$\rho_{11} = \frac{1}{2} + 2\left(\frac{1}{2} - \rho_{12} \right)\left(\frac{1}{2} - \rho_{21} \right)$$

$$= \frac{1}{2} + 2\left(\frac{1}{2} - \rho_{12} \right)^2 \tag{4.5}$$

$$= 1 - 2\rho_{12} + 2\rho_{12}^2 \tag{4.6}$$

Eq. (4.5) is recognizable as a conic section with eccentricity $e = 1$ and vertex $\{h, k\} = (1/2, 1/2)$ and corresponds to the parabolic locus in Figure 4.1. Both eq. (4.3) and eq. (4.6) confirm the alternative derivation in [120].

4.3 The new burst stabilization protocol

A good solution to instability in VOQ switches should not require internal speed-up or the comparison of state information between VOQs. A newly proposed burst stabilization protocol ('threshold and burst method' in [120]) neither requires costly

internal speed-up nor the comparison of state information between VOQs. When a VOQ in an input port is selected for the forwarding of a cell in the next cycle, a threshold comparison is made. As long as the current VOQ queue length exceeds a set *THRESHOLD*, then up to *BURST* cells can be transmitted from the VOQ before another VOQ from the same input port is allowed to be matched. This is similar in general principle to Threshold RRM (T-RRM) in [25]; in T-RRM *BURST* is effectively always 1.

Each VOQ has a cell burst counter that decrements on consecutive cell transfers (from the VOQ). This burst counter is set to *BURST* when a VOQ drains, or when the accept pointer is incremented (in iSLIP IQ) or the RR poll counter is incremented (in CICQ). In a CICQ switch, if a full CP buffer blocks the currently selected VOQ then the input port RR poll counter is always incremented. Specifically, for CICQ: The RR poll counter in an input port is not incremented if the currently selected VOQ is above *THRESHOLD* in queue length and the cell counter is greater than zero. The cell counter decrements on consecutive cell transfers from a VOQ. This counter is set to *BURST* when a VOQ drains or the RR poll counter is incremented (in CICQ). If a full CP buffer blocks the currently selected VOQ, then the input port RR poll counter is always incremented. This method can also be applied to iSLIP switches [120]. For the remainder of this chapter, we consider only the CICQ switch.

4.4 Simulation evaluation of burst stabilization protocol

Using a CSIM18 [101] simulation model, the effect of *THRESHOLD* and *BURST* values on stability and delay was studied for Bernoulli arrival of cells. Figure 4.2 shows mean switch delay (for VOQ_{11}, VOQ_{12}, and VOQ_{21} combined) for iSLIP and CICQ with

THRESHOLD set to 32 and *BURST* set to 0 and 64 for $\lambda_1 = 0.99$. Also shown are results from an OCF/RR CICQ switch (the VOQs are scheduled with OCF and the CP buffers with RR). These results show that with no bursting (*BURST* = 0) instability occurs, but with bursting the switch is stable. Figure 4.3 shows the mean switch delay for each VOQ for iSLIP and CICQ switch with *THRESHOLD* and *BURST* set to 32 and 64, respectively. This shows that iSLIP and CICQ switches with *THRESHOLD* and *BURST* have roughly similar delays for all VOQs, except VOQ_{11}.

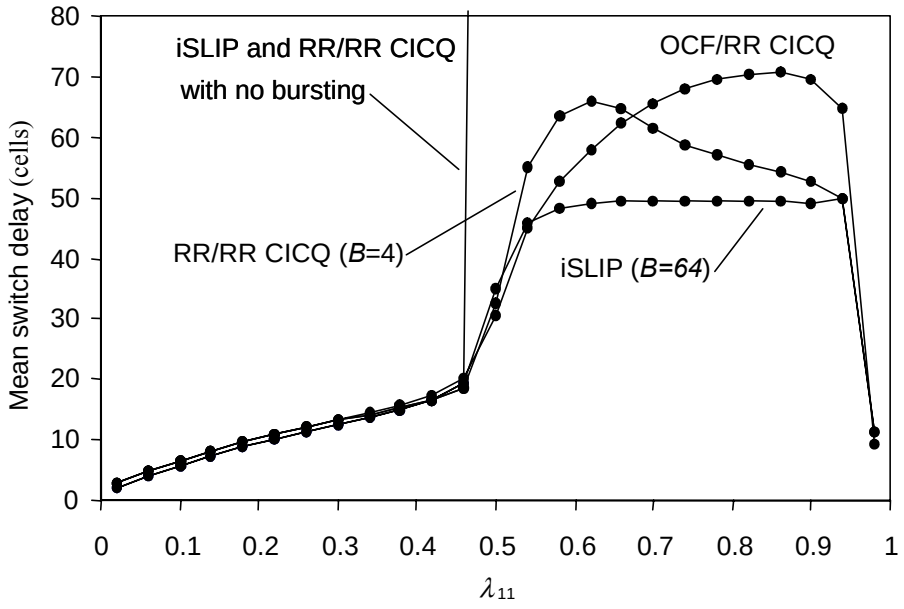

Figure 4.2 – Stability results for CICQ and iSLIP

To understand how the *THRESHOLD* and *BURST* method affects delay, three experiments varying *BURST*, *THRESHOLD*, and λ_{11} were conducted (all with $\lambda_1 = 0.99$). The measured variable was mean switch delay. The experiments were as follows:

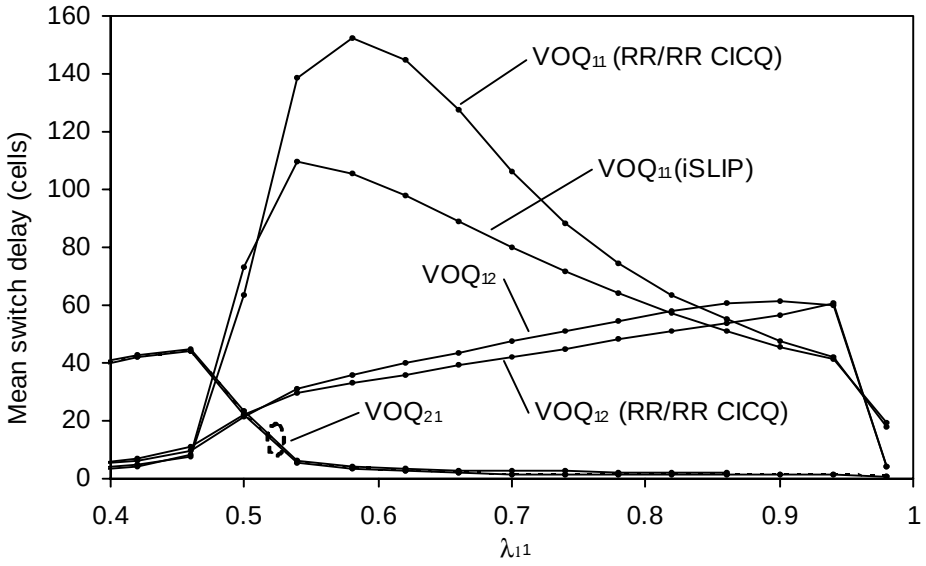

Figure 4.3 – Results for individual VOQs

1) *BURST experiment #1:* The effect of *BURST* is evaluated by varying λ_{11} and *BURST* for a fixed *THRESHOLD* = 32.
2) *BURST experiment #2:* The effect of offered load is evaluated by varying *BURST* and λ_{11} for a fixed *THRESHOLD* = 32.
3) *THRESHOLD and BURST experiment:* The effect of *THRESHOLD and BURST* was evaluated by varying *THRESHOLD* and *BURST* for a fixed $\lambda_{11} = 0.80$.

Figures 4.4 and 4.5 show the CICQ switch mean delay for VOQ_{11} for *BURST* experiments #1 and #2, respectively. Figure 4.4 shows that the mean delay for all *BURST* values is identical for $\lambda_{11} < 0.65$. Only the *BURST* value of 64 achieves stability for all λ_{11}. Figure 4.5 shows that larger λ_{11} requires larger *BURST* values. These results show that a too small *BURST* value results in instability for large λ_{11}. For all cases, the mean delay for iSLIP is similar to that of CICQ and is not shown. Figure 4.6 shows the

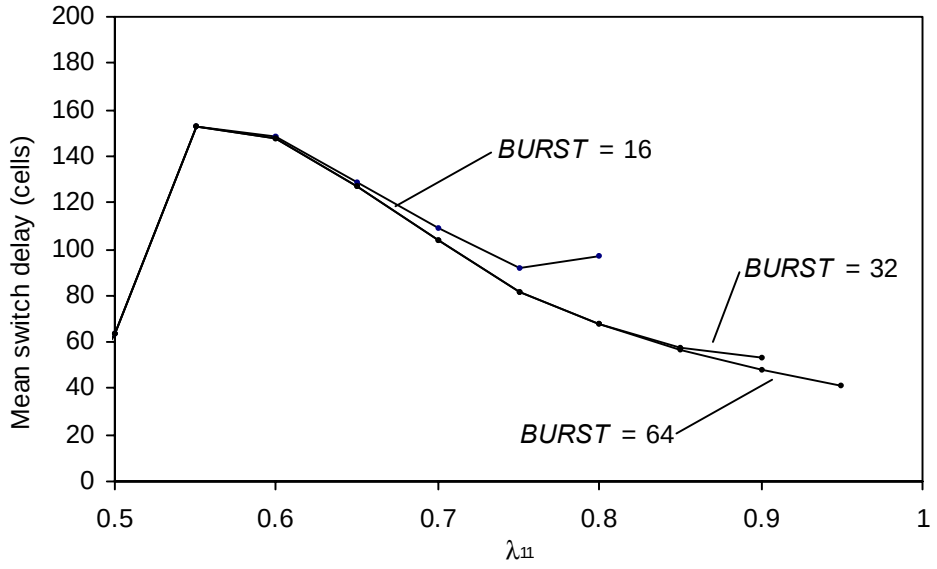

Figure 4.4 – Results for BURST experiment #1

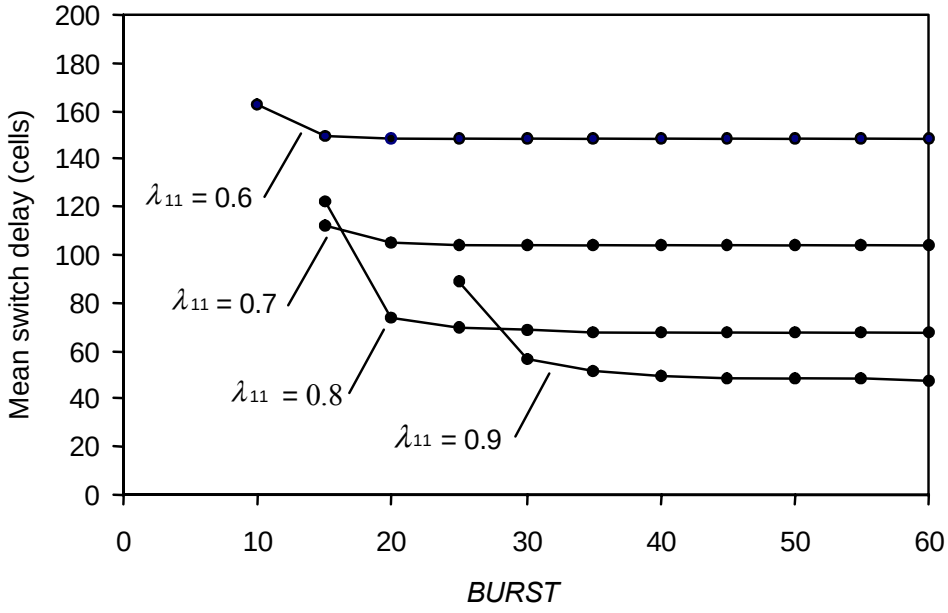

Figure 4.5 – Results for BURST experiment #2

mean delay for experiment #3 with *BURST* ranging from 15 to 55. *THRESHOLD* = 32 results in lower delay than *THRESHOLD* = 64, which in return has the lower delay than

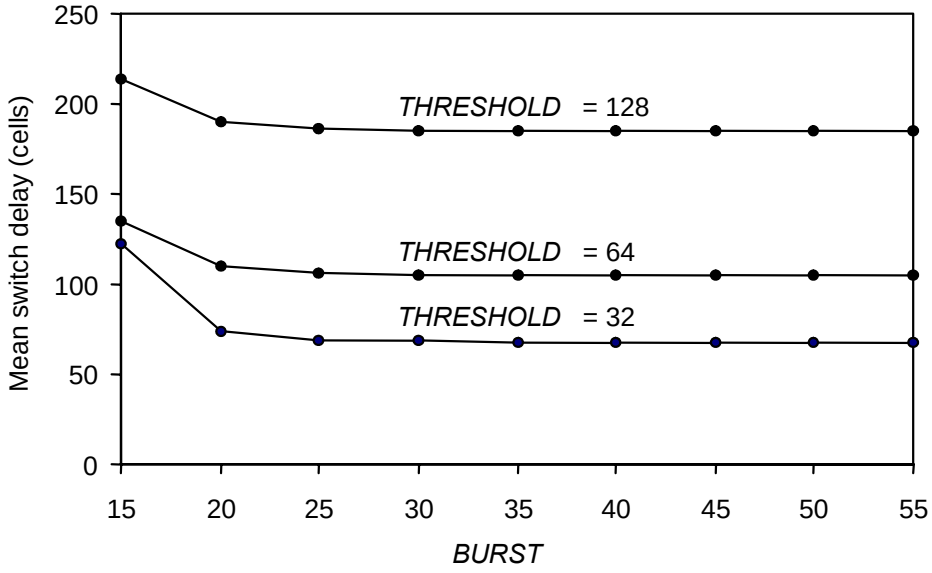

Figure 4.6 – Results for THRESHOLD and BURST experiment

THRESHOLD = 128. *THRESHOLD* values of 8 and 16 both result in instability. These results show that lower *THRESHOLD* values achieve lower delay, but too small of a *THRESHOLD* value results in instability. When *THRESHOLD* is too small, bursting (and its queue grows without bound). The mean delay for VOQ_{12} decreases as the delay for VOQ_{11} increases. The mean delay for VOQ_{21} is not significantly affected by the values of *THRESHOLD*, thus these results are not shown in the graph.

4.5 An analytical model of burst stabilization

In this section, an analytical model to predict the minimum *BURST* value needed to stabilize the system is developed. The work in this section is largely by Dr. Neil J. Gunther [36]. Goals to be established concerning the burst stabilization protocol are as follows:

1) Whether it is sufficient to inhibit such instabilities. This is discussed in Section 4.5.1.
2) Whether the magnitude of the *BURST* parameter can be predicted with sufficient accuracy for the important range of loads expected in a real switch. Section 4.5.5 contains those details.

The polled queues at each port are intrinsically stable if inputs and outputs are not oversubscribed. That is, $\sum_{i=1}^{N} \lambda_{ij} \leq 1, \ \forall\, i < j.$

The instability (described in Section 4.1) arises primarily from the blocking of a VOQ on one port by the transmission from a VOQ on another port via the corresponding output CP buffer on the crossbar. For example, buffer CP_{ii} blocks transmissions from VOQ_{ii} because of the presence of a cell in the downstream buffer CP_{ji} due to transmissions from VOQ_{ji} where $\geq \forall j > i$. This study shows that $\hat{B}_{ij}$, the estimate for the minimum *BURST* parameter, comprises two terms:

$$\hat{B}_{ij} = B_i + B_j \tag{4.7}$$

where B_i is due to traffic arriving at port i and B_j due to traffic arriving at port $j > i$. The interaction between these two traffic sources is such that their contribution to the minimum *BURST* size is both additive and load-dependent. In the subsequent discussion, $\hat{B}_{ij}$ signifies the minimum value of the *BURST* parameter required to stabilize the VOQs. All queue lengths are defined with respect to the *THRESHOLD* value, which (as mentioned in Section 4.4) acts as an arbitrary reference level.

Stability analysis is intrinsically difficult because transient effects [26], [46], [56], [75] may not possess a closed analytic form and only asymptotic bounds may be represented [35], [115]. Moreover, the VOQs at each input port in Figure 2.5 (and Figure 2.19) are subject to asymmetric traffic and even the steady-state behavior of such asymmetric polling systems can be difficult to express analytically [109]. Surprisingly, however, an accurate steady-state bound for CICQ switch stabilization is presented based on the fact that certain aspects of this problem resemble the equilibrium queue length,

$$E\{Q\} = \frac{\lambda^2}{\mu - \lambda} \frac{\left(C_s^2 - 1\right)}{2\mu} + \frac{\lambda}{\mu - \lambda} \tag{4.8}$$

for an M/G/1 model of an exhaustive polling system [8], [10], and [59] with C_s^2 the squared coefficient of variation of the service time S.

4.5.1 Vacating server approximation

The principles of operation of the burst stabilization protocol are best understood in the context of a simplified model having a single burst-stabilized queue. The generalization from this simple burst model to the multi-queue configuration in the real CICQ switch proceeds in a straightforward way.

Consider a single queue with arrivals that are Poisson distributed with rate $\lambda \geq 1/2$ and serviced in FCFS order. Poisson-distributed events have interarrival periods that are exponentially distributed and the latter distribution is the continuous analog of the discrete Bernoulli distribution used in the simulations of Section 4.3. Upon servicing a single request, the server vacates the queueing center for one service period $E\{S\} = \mu^{-1}$.

During that vacation period, other requests may arrive in the queue. From the standpoint of an arriving request, the expected service time appears to be $E\{S\} = 2$ because processing time is split equally between servicing the next request at the head of the queue and the next vacation period, i.e., an effective rate $\mu = 1/2$. Since $\lambda \geq \mu$, such a queue is non-ergodic, and therefore subject to unstable queue growth.

Now, suppose that when the queue size exceeds *THRESHOLD* the server ceases vacating the queue and proceeds to service at most *BURST* requests at a rate $\mu = 1$ before taking the next vacation period. Is it possible to find a *BURST* value large enough to bind the queue size over a sufficiently long time period? This question can be addressed using an M/G/1 generalized service time distribution model [8]; however, it will prove more instructive for later discussion to present a simpler rate matching argument.

In a fluid approximation [26], [46] the necessary condition (see Proof 1 in [36]) for stability is that *BURST* (*B*) requests be serviced in a time τ such that:

$$\frac{B}{\tau} = \lambda \tag{4.9}$$

The time-averaged rate of service must match the mean arrival rate in the long run. Applying this condition to our vacating server model, at most *B* requests must be serviced in a period $\tau = B + 1/\mu;$ where the "1" refers to the mean vacation period. The rate equation,

$$\frac{B\mu}{B+1} = \lambda \tag{4.10}$$

corresponds to the fraction of time for which the effective service rate reaches $\mu = 1$ in this model. It follows that the *BURST* size must be

$$B \geq \frac{\lambda}{\mu \quad \lambda} \tag{4.11}$$

which is finite and bounded, provided $\lambda < \mu$.

It is noteworthy that a term similar to eq. (4.11) arises in the steady-state limit of the rate processing function for leaky bucket queue management [34]. More importantly, it corresponds to the second term in eq. (4.8). This observation is further investigated in Section 4.3.

4.5.2 Port 2 analysis

Continuing this line of thought, it is easier to understand the contribution to *BURST* from port $j = 2$ interactions before turning to those due to port $i = 1$ traffic.

Definition 2. Let λ_2 be the offered traffic at port 2. The fraction of offered traffic going to VOQ21 is $\lambda_{21} = \lambda_{12} + (1 - \lambda_1)$ at instability and $\lambda_{22} = 0$.

Corollary 2. $\lambda_{12} = \lambda_{21}$ iff $\lambda_1 = 1$.

The mean interarrival time of cells at port 2 is $\tau_2 = \lambda_2^{-1}$. But $\lambda_{21} \equiv \lambda_2$ by virtue of $\lambda_{22} = 0$. Then $\tau_2 = \tau_{21} = \lambda_{21}^{-1}$, and $\tau_{21} \equiv \lambda_{21}^{-1} = 1 + B_2$ by analogy with the discussion in Section 4.5.1. Applying Proof 2 in [36], the contribution to *BURST* from port 2 is

$$B_2 = \frac{1 - \lambda_{21}}{\lambda_{21}} \equiv \frac{\lambda_{11}}{1 - \lambda_{11}} \tag{4.12}$$

Finally, eq. (4.12) can be rewritten in terms of the fraction of traffic going to VOQ_{11} as

$$B_2(f) = \frac{f\lambda_1}{1 - f\lambda_1} \tag{4.13}$$

by application of Definition 1.

This result states that the mean number of cells burst from VOQ_{11} prior to processing being blocked by transmissions from VOQ_{12} is the same as the equilibrium queue size for the vacating server model in Section 4.5.1. The reason is that a cell will be present at VOQ_{21} every τ_{21} cell times (on average) causing the scheduler to vacate VOQ_{11} and service VOQ_{12}.

4.5.3 Port 1 analysis

The study now turns to the analysis of port $i = 1$. Because bursts can be interrupted with a mean time τ_{21}, there is also the possibility that VOQ_{12} can burst. As seen from the standpoint of VOQ_{11}, the vacation period is extended beyond that accounted for by eq. (4.13). Since cells will continue to arrive into VOQ_{11} the burst size will need to be larger than B_2.

The average number of arriving cells L_1 that accumulate during this extended vacation period is directly proportional to the arrival rate at port 1 according to Little's Law,

$$L_1 = \lambda_1 W_1 \tag{4.14}$$

where W_1 is the expected waiting-time. Eq. (4.14) corresponds to the first term in eq.

(4.8) viz,

$$L_1 = \frac{\lambda_1^2}{1-\lambda_1}\frac{(C_s^2 - 1)}{2} \tag{4.15}$$

with $\mu = 1$. In polling systems near saturation, waiting-times approach a gamma distribution and higher moment effects vanish under heavy traffic [114].

The instability of interest to the present study, however, arises from heavy asymmetric traffic into VOQ_{11}. This has the effect of making the residual service time in eq. (4.15) a load-dependent function via the squared coefficient of variation,

$$C_s^2(f) = 1 + \frac{4}{5}\left(f - \frac{1}{2}\right) \tag{4.16}$$

such that B_1 in eq. (4.7) becomes:

$$B_1(f) = \frac{2}{5}\left(\frac{\lambda_1^2}{1-\lambda_1}\right)\left(f - \frac{1}{2}\right) \tag{4.17}$$

with $f \geq 1/2$ the asymmetric fraction of the offered load arriving at VOQ_{11}. Note that f is the only variable in eq. (4.17).

As expected the length of the waiting line, and hence its contribution to the minimum *BURST* size, is directly proportional to the imbalance in VOQ_{11} traffic. When $f = 1/2$ (symmetric case) this term vanishes and does not contribute $\hat{B}_{ij}$.

4.5.4 Bounds on *BURST*

The complete expression for estimating the minimum *BURST* as defined in eq. (4.7) is

$$\hat{B}_{ij}(f) = \frac{2}{5}\left(\frac{\lambda_i^2}{1-\lambda_i}\right)\left(f - \frac{1}{2}\right) + \frac{f\lambda_i}{1-f\lambda_i}, \quad \forall i < j \tag{4.18}$$

More formally *BURST* corresponds to the infimum (greatest lower bound) of $\hat{B}_{ij}(f)$. As a practical matter, this can be evaluated most simply as the ceiling function $\lceil \cdot \rceil$ applied to eq. (4.18). For $\lambda_i \leq 1$ and $f = 1/2$, $\lceil B_i(1/2) \rceil = 1$ and $\lfloor B_j(1/2) \rfloor = 0$.

As expected from the simulation experiments (Figure 4.7), eq. (4.18) is an increasing function of *f* but not necessarily monotonic. It does not scale with the number of ports

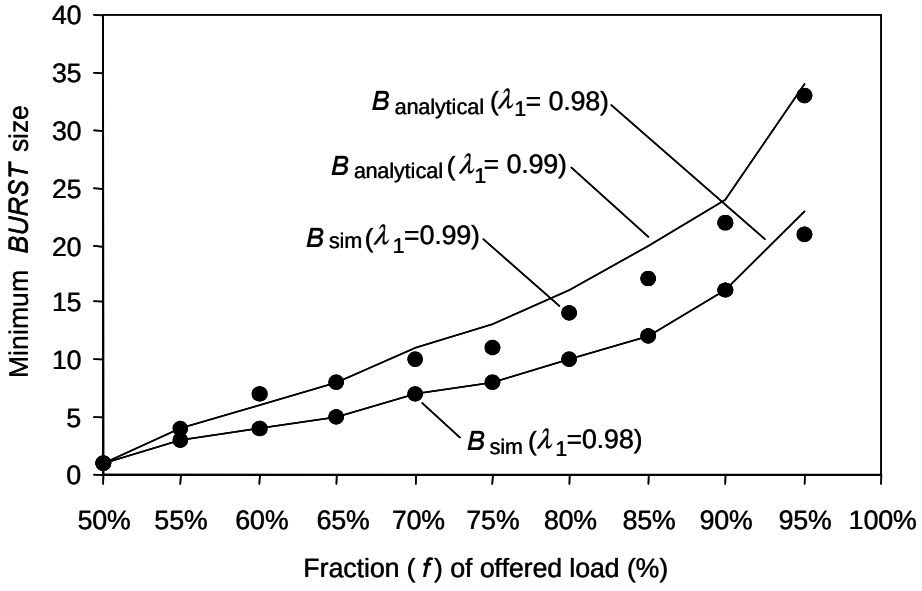

Figure 4.7 – Prediction of minimum BURST value

(N) since blocking between any pair of ports (as remarked in Section 3) is sufficient to cause unstable queue growth. Eq. (4.18) is rather remarkable in that it is based on a steady-state representation yet the fluctuations concerning these mean values can be very significant. It is also surprising that it can be expressed entirely in terms of the fractional load at VOQ_{ii}. It should be noted that eq. (4.18) is not valid for $f = 1$ since there is a cell in every slot and the behavior becomes D/D/1 (i.e., non-Poisson arrivals).

4.5.5 Numerical results

For the simulation results, the experiment of Section 3.2 was repeated (again using *THRESHOLD* = 32) with a *BURST* size that was incrementally increased across each run until the sampled drift in VOQ_{11} queue growth was observed to be adiabatically zero. Tables 4.1 and 4.2 present the results for a range of λ_1 values and a comparison with the corresponding predictions of eq. (4.18). Under- and over-estimations are indicated respectively by (–) and (+) signs. Figure 4.7 also shows the measured (B_{sim}) and the predicted ($B_{\text{analytical}}$) minimum *BURST* values. $B_{\text{analytical}}$ is defined as equivalent to $\left\lceil \hat{B}_{12} \right\rceil$.

These results demonstrate that increasing the value of *BURST* achieves stability by increasing the bandwidth for VOQ_{11} at the expense of VOQ_{12}. For $\lambda_1 = 0.98$ and $\lambda_1 = 0.99.$ the value of *BURST* was determined for the stability achieved. In this way it was established that eq. (4.18) predicts the exact *BURST* size with less than or equal to 10% relative error for $\lambda_1 = 0.98$ and less than 20% relative error for $\lambda_1 = 0.99.$ In all cases, except one, the relative errors are conservative since they are overestimates.

Table 4.1 – Calculated and simulated minimum BURST values for $\lambda_1 = 0.98$

Loads			Model			Comparison		
f	λ_{11}	λ_{12}	B_2	B_1	B_{12}	$\hat{B}_{12}$	B_{sim}	Error
0.50	0.49	0.49	0.96	0.00	0.96	1	1	0.00
0.55	0.54	0.44	1.17	0.96	2.13	3	3	0.00
0.60	0.59	0.39	1.43	1.92	3.35	4	4	0.00
0.65	0.64	0.34	1.75	2.88	4.64	5	5	0.00
0.70	0.69	0.29	2.18	3.84	6.03	7	7	0.00
0.75	0.74	0.25	2.77	4.80	7.58	8	8	0.00
0.80	0.78	0.20	3.63	5.76	9.39	10	10	0.00
0.85	0.83	0.15	4.99	6.72	11.71	12	12	0.00
0.90	0.88	0.10	7.47	7.68	15.16	16	16	0.00
0.95	0.93	0.05	13.49	8.64	22.14	23	21	+ 0.10

Table 4.2 – Calculated and simulated minimum BURST values for $\lambda_1 = 0.99$

Loads			Model			Comparison		
f	λ_{11}	λ_{12}	B_2	B_1	B_{12}	$\hat{B}_{12}$	B_{sim}	Error
0.50	0.54	0.45	0.98	0.00	0.98	1	1	0.00
0.55	0.54	0.45	1.20	1.96	3.16	4	4	0.00
0.60	0.59	0.40	1.46	3.92	5.38	6	7	– 0.14
0.65	0.64	0.35	1.81	5.88	7.69	8	8	0.00
0.70	0.69	0.30	2.26	7.84	10.10	11	10	+ 0.10
0.75	0.74	0.25	2.88	9.80	12.68	13	11	+ 0.18
0.80	0.79	0.20	3.81	11.76	15.57	16	14	+ 0.14
0.85	0.84	0.15	5.31	13.72	19.03	20	17	+ 0.18
0.90	0.89	0.10	8.17	15.68	23.86	24	22	+ 0.09
0.95	0.94	0.05	15.81	17.64	33.45	34	33	+ 0.03

Chapter 5: Switching Variable-Length Packets

As evidenced by the USF distribution packet length statistics, most network traffic consists of IP packets, often framed with an Ethernet header and trailer. IP packets are variable in length. Most applications generate data in variable sized blocks. Thus, the study of high-speed switches that support variable-length packets is significant. Section 5.1 investigates switching variable-length packets for IQ switches. Most of Section 5.1 resulted from collaborations with Dr. Allen Roginsky at IBM Corporation and Dr. Neil J. Gunther at Perfomance Dynamics Company. Section 5.2 investigates switching variable-length packets for CICQ switches.

5.1 Packet-to-cell segmentation schemes in IQ switches

Existing IQ switches used in IP networks segment external variable-length packets into internal fixed-length cells. Variable length packets are segmented into fixed-length cells in the input ports; the cells are then scheduled and transferred, and subsequently reassembled into packets in the output ports. Figure 5.1 shows a VOQ switch with each input port containing a packet classifier to determine the destination output port, packet-to-cell segmenter, VOQs, and a scheduler. Switch matrix scheduling algorithms for VOQ crossbar switches inherently require the use of internal fixed-length cells. This is because the crossbar is scheduled in cycles, one cycle for each set of cells forwarded

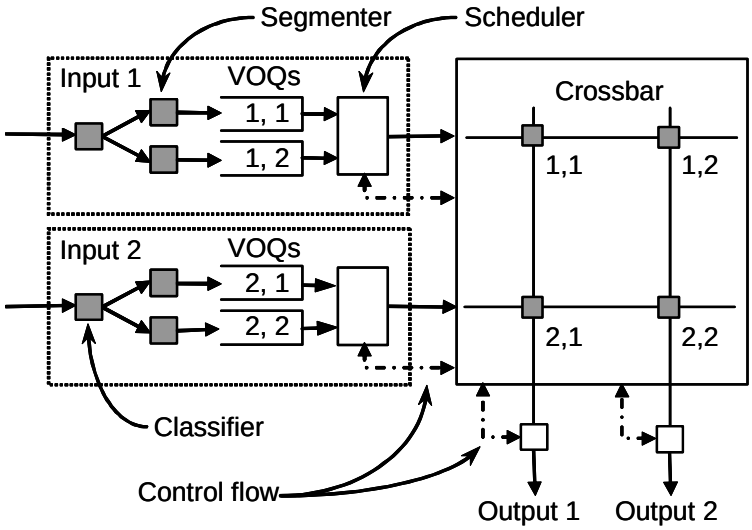

Figure 5.1 – VOQ switch showing packet-to-cell segmenter

from matched input ports to output ports. Thus, input buffered switches segment packets into cells, internally switch the cells, and then reassemble the cells into packets in reassembly buffers at the output ports.

The IP-PIM algorithm for IQ switches [86] uses a "cell train" in which the matching of a head-of-packet cell is maintained until all cells of that packet have been forwarded. Cell trains are also used in iSLIP based IQ switches [68], [65], [55], [81]. The use of cell trains within a switch simplifies packet reassembly at the output ports since the cells of a segmented packet are stored in the output buffer without interleaving of cells from other segmented packets. The use of cell trains provides better performance than individual cell-by-cell scheduling for variable-length packets for iLQF, iOCF, and iSLIP [66].

Few packets have a length that is an even multiple of the internal cell length requiring internal speed-up, and the overhead from speed-up adds cost to a switch implementation.

No existing work quantitatively addresses how much speed-up is needed. A new method for reducing speed-up needs to be investigated. These open problems are addressed.

5.1.1 Models of quantized service time queues

It is assumed that packet lengths are randomly distributed and their distribution is known. For a given packet of length L bytes and a cell of size S bytes, $ceil(L/S)$ cells are needed to segment the packet where ceil is the standard ceiling function. The last packet in the sequence, or "train", of cells will have $S-r$ padding or overhead bytes where $r = Rem(L,S)$ when L is not a multiple of S; $r = S$ otherwise. This results in an internal packet length (in cells) of $L+S-r$ bytes. Thus, an internal speed-up factor of $1+(S-r)/L$ is needed to switch the packet at link data rate. For a given distribution of packet lengths, the study investigates how this discretization (or quantization) caused by segmentation into cells changes the mean and variance of the number of bytes to be transported. This changes the service time of a queue modeling the input buffer of a packet switch that segments packets to cells as described. The study also models queueing delay of three classical queues given a ceiling of service time.

5.1.1.1 Ceiling of well known distributions

Let X be an arbitrary random variable for which it is known that $E(X)$ and $Var(X)$. Let Y be the integer-valued random variable that is the ceiling of X and let $Z = Y - X$. If X has a wide smooth distribution and is not concentrated near a particular integer or set of integers, it can be assumed that Y and Z are almost independent and the distribution of Z

on $(0,1)$ is roughly uniform. We have $E(Z)=1/2$ and $Var(Z)=1/12$. We have $E(Y)=E(X+Z)=E(X)+E(Z)\ =E(X)+1/2$ and, $Var(X)=Var(Y-Z)\ =Var(Y)+Var(Z)$ $=Var(Y)+1/12$, and hence $Var(Y)=Var(X)-1/12$.

For an exponentially distributed random variable X with $f_X(x)=\mu e^{-\mu x}$, let $Y=ceil(X)$. Then Y is geometrically distributed with

$$E(Y)=\frac{1}{1-e^{-\mu}} \text{ and} \tag{5.1}$$

$$Var(Y)=\frac{e^{-\mu}}{(1-e^{-\mu})^2}. \tag{5.2}$$

The proof of this is in [16]. As expected,

$$\lim_{\mu\to 0}\left(\frac{1}{1-e^{-\mu}}-\frac{1}{\mu}\right)=\frac{1}{2} \text{ and} \tag{5.3}$$

$$\lim_{\mu\to 0}\left(\frac{e^{-\mu}}{\left(1-e^{-\mu}\right)^2}-\frac{1}{\mu^2}\right)=-\frac{1}{12}. \tag{5.4}$$

Let H_2 denote a two-stage hyperexponentially distributed random variable X with $f_X(x)=\alpha_1\mu_1 e^{-\mu_1 x}+\alpha_2\mu_2 e^{-\mu_2 x}$ and $x>0$, $\alpha_1,\alpha_2\geq 0$, and $\alpha_1+\alpha_2=1$, let $Y=ceil(X)$. Then,

$$E(Y)=\frac{\alpha_1}{1-e^{-\mu_1}}+\frac{\alpha_2}{1-e^{-\mu_2}} \text{ and} \tag{5.5}$$

$$Var(Y)=\alpha_1\frac{1+e^{-\mu_1}}{(1-e^{-\mu_1})^2}+\alpha_2\frac{1+e^{-\mu_2}}{(1-e^{-\mu_2})^2}-\left(E(Y)\right)^2. \tag{5.6}$$

The proof of this is in [16].

Let E_2 denote a two-stage Erlang distributed random variable X with $f(x) = \mu^2 x e^{-\mu x}$ (i.e., both service rates are the same). Then Y has,

$$E(Y) = \frac{(\mu - 1)e^{-\mu} + 1}{(1 - e^{-\mu})^2} \text{ and} \tag{5.7}$$

$$Var(Y) = \frac{[(\mu + 1)e^{-\mu} - (\mu - 1)e^{-3\mu} - (\mu^2 + 2)e^{-2\mu}]}{(1 - e^{-\mu})^4}. \tag{5.8}$$

The proof of this is in [16]. It can be seen that, when $Var(Y)$ is evaluated numerically, it is usually greater than the value of $Var(X)$, which is equal to $2/\mu^2$. The reason for this is that the Erlang distribution does not satisfy the heuristic assumptions made earlier in this section. It has a peak, which makes the discrete picture more complicated.

5.1.1.2 M/G/1 analysis

The quantization described in Section 5.1.1.1 corresponds to taking the ceiling function of the continuous-valued service time. We introduce a superscript notation Δ for these quantized service times so that the quantized version of an M/M/1 queue is denoted $M/M^{\Delta}/1$, which is also equivalent to M/Geo/1 with mean service time $(1 - e^{-\mu})^{-1}$ defined by eq. (5.1). Since $E(Y)$ and $Var(Y)$ are continuous functions, the mean number of customers in the system $E(N)$ can be determined from the standard Pollaczek-Khintchine (P-K) M/G/1 formula [5]:

$$E(N) = \rho + \frac{\rho^2}{2(1 - \rho)}\left[1 + \frac{Var(Y)}{E^2(Y)}\right]. \tag{5.9}$$

where $\rho = \lambda E(Y)$ is the quantized load. For $M/M^{\Delta}/1$ eq. (5.9) reduces to

$$E(N) = \frac{\lambda}{2}\left(\frac{\lambda - 2}{e^{-\mu} + \lambda - 1}\right) \tag{5.10}$$

with μ the mean service rate of the underlying exponential distribution used to calculate the quantized service times. The study also solved $M/H_2^{\Delta}/1$ and $M/E_2^{\Delta}/1$ queues for $E(N)$, but those formulas are far more complex than eq. (5.10) and are not shown here. Moreover, beyond the special cases discussed in Section 5.1.1.1, an arbitrary response time distribution is not expected to possess an analytic quantized form, so a more general approach is needed.

5.1.1.3 Application of models to packet-to-cell segmenting

At this point in the study the $M/M^{\Delta}/1$ model is applied to predict $E(N)$ in packets given L, S, and a utilization ρ. The utilization, ρ, is based on the arriving rate of packets (in bits per second) divided by the link rate (in bits per second). Packet arrivals are Poisson, and packet lengths are exponentially distributed. These assumptions are very restrictive and unrealistic of real packet traffic. However, even with these restrictive assumptions, the general behavior of segmentation and speed-up can be observed. In the next section, these restrictive assumptions are removed in a simulation study.

For a given L and S, the mean service time in cell time units is $T_s = L/S$. For a given link utilization based on packets, ρ, the mean interarrival time is $T_a = T_s/\rho$. Then, the mean arrival rate is $\lambda = 1/T_a$ and mean service rate is $\mu = 1/T_s$. For the M/M/1, $\rho = \lambda/\mu$ and mean number of packets in the system $E(N) = \rho/(1-\rho)$. For the $M/M^{\Delta}/1$, the utilization for quantized service time is, $\rho' = \lambda \cdot E(Y) = \lambda/(1 - e^{-\mu})$. The

speed-up, σ, needed to achieve a carried load equal to the offered load (i.e., stability for all offered loads up to $\rho = 1$) is $\sigma = \rho'/\rho = \mu E(Y) = \mu/(1 - e^{-\mu})$. Figure 5.2 shows the numerical results for $E(N)$ for a range of $\rho = 0.50, 0.51, \ldots, 1.0$ and $L = 100$, 500, and 1000 bytes for $S = 64$ bytes. To achieve stability, the speed-up required is $\sigma = 1.354$ for $L = 100$ bytes, $\sigma = 1.065$ for $L = 500$ bytes, and $\sigma = 1.032$ for $L = 1000$ bytes. These numerical results show that, without speed-up, $L = 1000$ increases rapidly at high offered loads and that the smaller L is, the greater the effect of segmentation on $E(N)$. The analytical results in Figure 5.2 have been validated with a simulation model. This model was then used in the next section for more realistic traffic models.

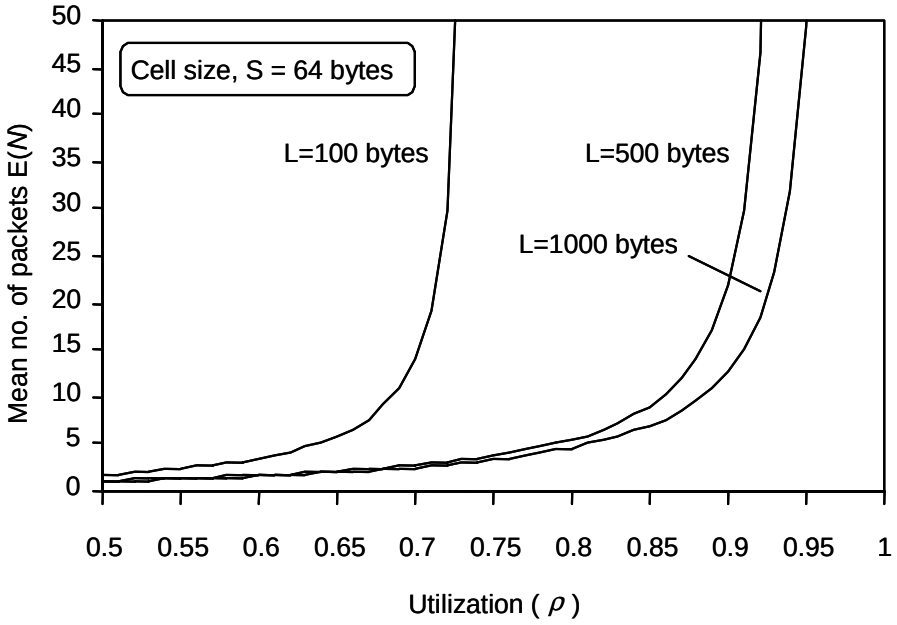

Figure 5.2 – Numerical results for $M/M^{\Lambda}/1$ for various values of L

5.1.2 Simulation of iSLIP with packet segmentation

The effects of discretization on a real network traffic using a simulation model with a packet trace as input was evaluated. A previously built and validated iSLIP simulation model from chapter 3 was used.

5.1.2.1 Traffic model

Traced traffic was used for the simulation evaluation of the cell merging method.

USF traced traffic #2: Over 60 million IP packets were collected from the University of South Florida (USF) Internet2 OC-3 (155-Mbps) link on December 10, 2002. USF traced traffic #2 is needed because the size of USF traced traffic #1 (5 million packets) is not large enough for the experiment. Packet size distributions from the two traced traffic collections resemble each other, and the shape of packet size distributions are bimodal for both traced traffic collections. Figure 5.3 shows the packet length histogram where all packet lengths from 64 to 1518 bytes are represented. The mean packet length was 764 bytes. Thus, an external-packet/internal-cell switch with a cell size of 64 bytes requires a speed-up of 1.005 to achieve stability $(12 \times 64)/764.1 = 1.005)$. The most common packet length was 1518 bytes (31.4% of all packets), followed by 64 bytes (28.7%), 1438 bytes (7.7%), 70 bytes (2.7%), and 594 bytes (1.4%). All other packet length occurs at less than 1%.

The trace file of 60 million packets was split into 16 smaller files, each with the same number of packets. Each of these smaller files was then input to a port in the modeled 16-port iSLIP switch. The destination output port was assigned using a modulo-16 function of the packet IP destination address. Output port utilization of a switch is not

uniform for real traffic. In this experiment, utilization is referred to as the maximum offered load among all 16 ports. Service time (i.e., simulated link data rate) is controlled to achieve a desired utilization. Using this traced traffic with a real packet length distribution allows speed-up issues in an IQ switch to be studied.

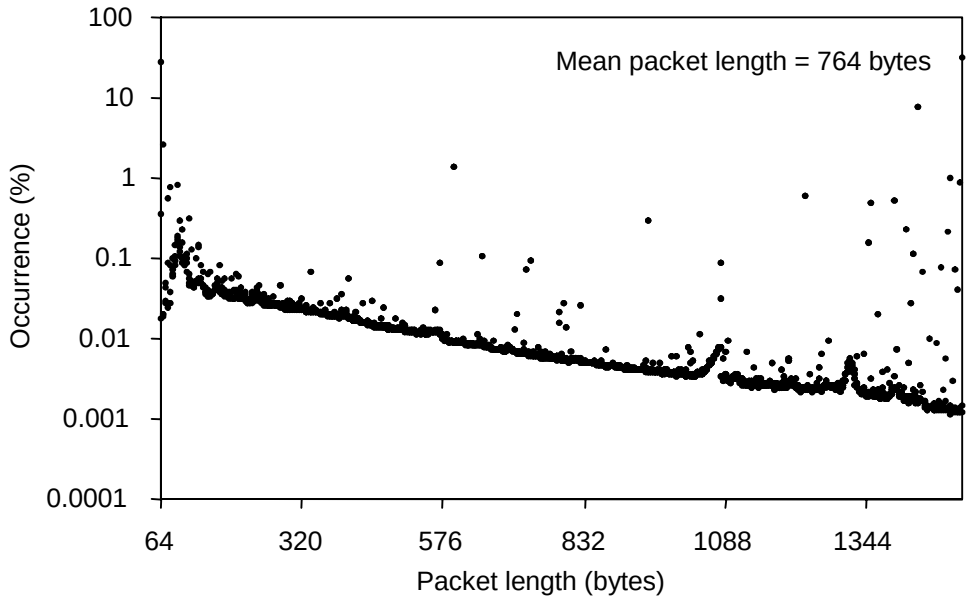

Figure 5.3 – Histogram for USF traced traffic #2 of Ethernet packet lengths

5.1.2.2 Simulation experiments

For all experiments, control variables are offered load and speed-up, and the response variable is mean queue length. An internal cell size of 64 bytes (minimum Ethernet packet size) was used. A 3% accuracy (95% confidence interval) was used as the simulation stopping criterion.

1) *Stability experiment:* For iSLIP with no speedup, 1.05x, and 1.1x speedups with segmentation and cell padding, mean queue length is measured for utilization ranging from 50% to 99%.
2) *Speed-up experiment:* The minimum speedup needed for 99% utilization is systematically identified. A queue length of 5000 or greater is considered a sign of instability.

5.1.2.3 Experiment results

Figure 5.4 shows the results for the *Stability experiment*. It can be seen that no speedup and the 1.05x speedup cases become unstable above 93% and 97% utilization, respectively. Only the 1.1x speedup case can achieve stability for the entire range of utilizations. For *Speed-up experiment*, it was found that the minimum speed-up needed for 99% utilization was 1.06x.

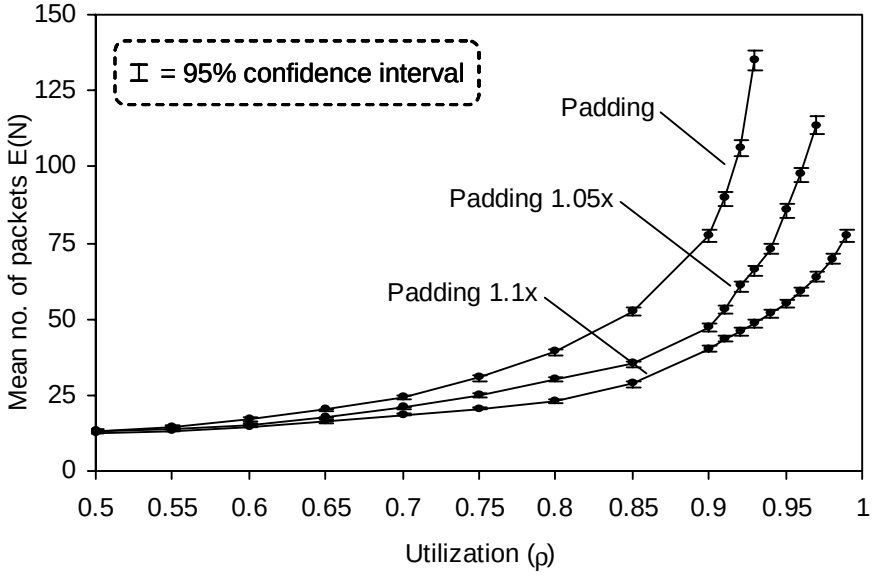

Figure 5.4 – Results for stability experiment

5.1.3 Packet-to-cell segmentation with the new cell merging method

For input buffered switches, a new method of segmenting packets into cells that reduces the amount of speed-up needed to achieve stability is proposed. When a packet is segmented into cells (i.e., in the segmenters shown in Figure 5.1) the last cell of a packet may be a partially filled cell. Instead of queueing this partial cell (with padding bytes) to the VOQ, it is held back to wait for the next arriving packet. The next arriving packet then starts its segmentation with the held-back cell from the previous packet; i.e., the header bytes of the arriving packet are merged with the trailer bytes from the previous packet. This is called cell merging. A finite state machine (FSM) for cell merging is shown in Figure 5.5. The EMPTY state occurs when there is no held back cell and the segmenter is idle.

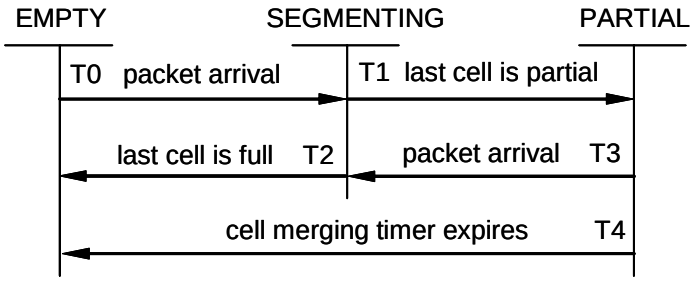

Figure 5.5 – FSM for cell merging

An arriving packet transitions (T0) the FSM to the SEGMENTING state where the packet is segmented into cells and the cells queued in the VOQ. If the last cell in a packet is a partial cell it is held back and the FSM transitions (T1) to the PARTIAL state (otherwise, the transition is to the EMPTY state (T2)). In the PARTIAL state, the segmenter is idle and waiting for an arriving packet to transition (T3) back to the

SEGMENTING state. In the PARTIAL state a cell merging timer is started when the VOQ is empty (e.g., all cells segmented and queued have been forwarded). If this timer expires before an arriving packet, then the held back cell is queued with padding bytes and transition (T4) is to the EMPTY state. The purpose of the timer is to prevent a packet from being unfairly starved if there are no subsequent arrivals for a long period of time.

For the cell merging mechanism, the effect of cell merging timer values was evaluated in advance, and the cell merging timer expiration value was set to 10 cell times. A large (100 or 1000 cell times) timer value results in high queueing delays at low utilizations, and no benefit at high utilizations (where a time-out would rarely occur due to frequently arriving packets). A value of 10 cell times maintained a low queueing delay at both low and high utilizations. For a 10-Gbps link, 10 cell times corresponded to a very small 512 ns.

5.1.3.1 Simulation evaluation of cell merging

Experiments from Section 5.1.2.2.were repeated for the cell merging mechanism with the cell merging timer expiration value set to 10 cell times. Figure 5.6 shows the mean queue of cell merging compared with the results of Figure 5.4 (padding). The packet merging mechanism with no speedup became unstable above 95%. The cell merging mechanism with a speedup of 1.05x and 1.1x achieved stability for all offered load measured. Cell merging resulted in a lower mean queueing delay for high utilizations.

From the *Speed-up* experiment it was found that the minimum speedup needed was 1.04x. Thus, cell merging required 2% less speedup than packet-to-cell segmentation

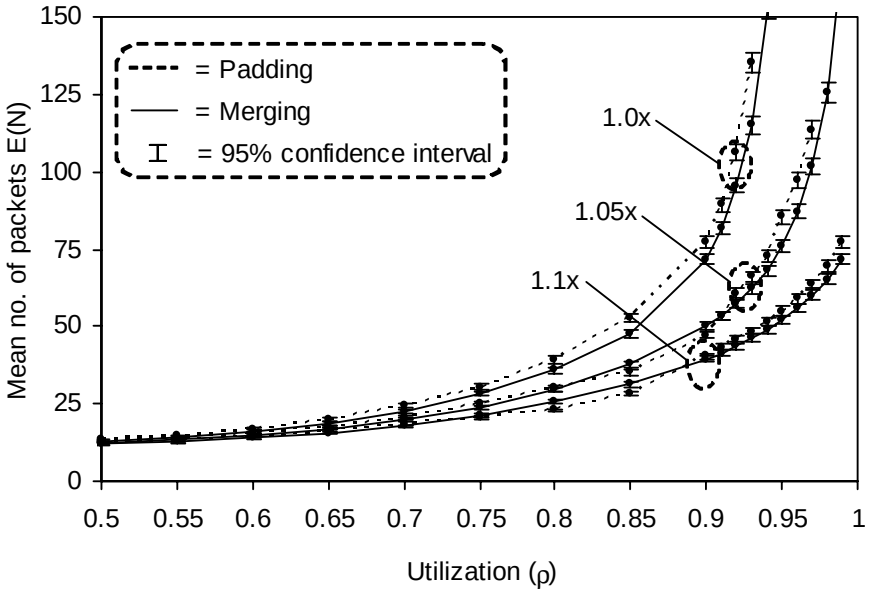

Figure 5.6 – Results for stability experiment with cell merging

without cell merging. Cell merging timer values were experimented with. A large (100 or 1000 cell times) timer value would result in high queueing delays at low utilization and no benefit at high utilizations (where a time-out would rarely occur due to frequently arriving packets). A value of 10 cell times maintained a low queueing delay at both low and high utilizations. For a 10-Gbps link, 10 cell times corresponds to a very small 51.2 ns.

In summary, the cell merging method reduces the required speed-up in an IQ switch. This is significant because increases in link data rate continue to outpace improvements in memory speed. It is not productive to justify up to a 2x speed-up just for handling the variable length nature of IP packets in the Internet. Even a small speed-up adds cost to high-speed packet switches.

5.2 Switching variable-length packets for CICQ switches

As seen in the previous section, a VOQ IQ switch requires costly internal speed-up to support variable-length packets. A CICQ switch does not require internal speed-up since it can natively forward variable-length packets [121]. In a CICQ switch, each VOQ has its own output buffer in the form of a CP buffer. When a complete packet finally exists in a CP buffer, that buffer is then eligible to transfer its contents to an output port on the next RR poll. Such a switch requires no speed-up since neither padding bytes nor segmentation/reassembly of the packets at the input/output ports are needed.

5.2.1 Unfairness among VOQs in CICQ switch

Variable-length packets result in unfairness among VOQs in the CICQ switch as follows: Figure 5.7 shows a 1x2 CICQ switch with two VOQs, $VOQ_{1,1}$ (VOQ at input port 1 for output port 2) and $VOQ_{1,2}$. Let $VOQ_{1,1}$ be saturated with packets of length L; let $VOQ_{1,2}$ be saturated with packets of length $2L$; and the size of CP $\geq 4L$. Both $CP_{1,1}$ (CP for packets from input port 1 to output port 1) and $CP_{1,2}$ can start forwarding packets to output ports 1 and 2, respectively, as soon as they receive packets, and, thus, are ready to accept the next packet from the VOQs (they always have a sufficient buffer available for at least one more packet of size $2L$). $VOQ_{1,1}$ and $VOQ_{1,2}$ are selected alternatively; thus, more bits are transferred from $VOQ_{1,2}$. This condition can obviously exist in a switch with any port count. It is desired that the complexity of the segmentation/reassembly of packets, the speed-up of switch fabric, and the unfairness among queues be reduced or eliminated.

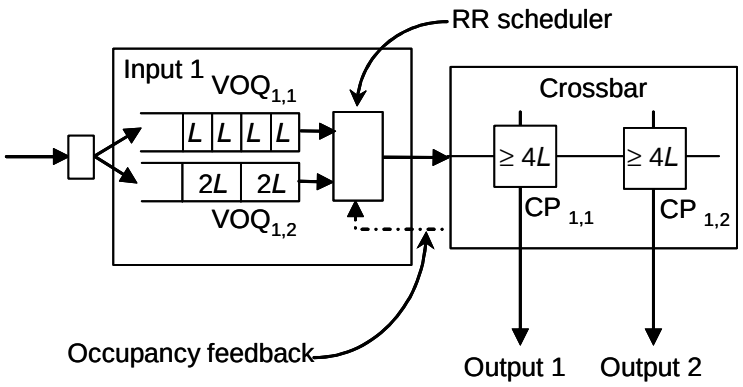

Figure 5.7 – 1x 2 CICQ switch showing packet-level unfairness

5.2.2 Block transfer mechanism

A block transfer mechanism for the CICQ switch to resolve the unfairness caused by variability of packet lengths is proposed and evaluated. The block transfer mechanism transfers up to a predefined number (*BLOCK*) of bytes of packet data from a selected VOQ to a CP. Figure 5.8 and 5.9 show the block transfer mechanism and the pseudocode for the block transfer mechanism, respectively. Each block of size (*transfer_size*) can be a set of entire packets and/or portions of a packet. If a selected VOQ has less than or equal to *BLOCK* bytes, entire packets are transferred (as is the case for $VOQ_{1,1}$ in Figure 5.5). If a selected VOQ has more than *BLOCK* bytes, the last packet may or may not be a part of the block. If the last packet is a part of the block, at least 64 bytes of the last packet (as is the case for $VOQ_{1,2}$ in Figure 5.8) are held back to ensure that the bus between an input port and a CP is optimally utilized during the scheduling time, even if all other VOQs are empty, (Here it is assumed that the scheduling time is less than or equal to 64-byte packet transfer time.) If the last packet is not a part of the block, a block of *BLOCK* bytes is transferred (as is the case for $VOQ_{1,3}$ in Figure 5.8).

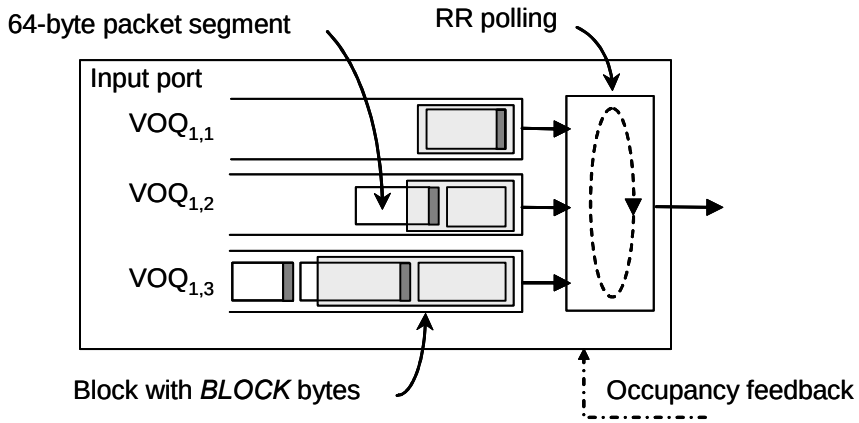

Figure 5.8 – Block transfer mechanism

```
At each input port_i,
 1. do forever
 2.   Select with RR the first non-empty VOQ_{i,j} where CP_ status_{i,j} = 0.
 3.   If (total packet size at VOQ_{i,j} is less than or equal to BLOCK) then
 4.      transfer_size = total packet size at VOQ_{i,j}
 5.   else
 6.      if (total packet size at VOQ_{i,j} – BLOCK) < 64 then
 7.         transfer_size = total packet size at VOQ_{i,j} – 64
 8.      else
 9.         transfer_size = BLOCK
```

Figure 5.9 – Pseudocode for block transfer mechanism

The complete packets at a CP are forwarded to a corresponding output link when selected by RR arbitration. $CP_{i,j}$ periodically (in every 64-byte data transfer time) checks if the $CP_{i,j}$ has at least *BLOCK* bytes of free space, and sends a status bit, $CP\_status_{i,j}$ (1 = full, 0 = empty), to an input $port_i$. The block transfer mechanism allows a block of up to *BLOCK* bytes to be transferred from a selected VOQ regardless of the individual packet size. Thus, the unfairness due to the variability of the packets is removed. Suppose

$VOQ_{1,1}$ and $VOQ_{1,2}$ from Figure 5.4 are saturated with 64-byte packets and 1518-byte packets, respectively. Without the block transfer mechanism, one 64-byte packet from $VOQ_{1,1}$ and one 1518-byte packet from $VOQ_{1,2}$ are scheduled alternatively. This results in $VOQ_{1,2}$ transferring 23.7-times (1518 divided by 64) more data than $VOQ_{1,1}$. With the block transfer mechanism, both $VOQ_{1,1}$ and $VOQ_{1,2}$ are scheduled with the same amount of data, resulting in perfectly fair scheduling.

5.2.3 Evaluation of block transfer mechanism

This section evaluates the performance of the CICQ switch with a block transfer mechanism. Using CSIM18 [101], simulation models were developed for an CICQ switch with native packet forwarding, a CICQ switch with the block transfer mechanism, an iSLIP-scheduled IQ switch with cell train, and an OQ switch. A 16-port switch is modeled, and infinite buffer sizes are assumed for all experiments. iSLIP is implemented for four iterations. For all experiments, control variables and response variables are offered load and switching delay, respectively.

5.2.3.1 Traffic models

Both synthetic and traced traffic were used for the simulation evaluation of the block transfer mechanism.

1) *USF synthetic traffic:* See Chapter 3.3.3 for the description.
2) *Bimodal synthetic traffic (64, 1472):* Arrivals are Poisson, and packet lengths are either 64 bytes or 1472 bytes. Both 64 bytes and 1518 bytes (minimum and maximum Ethernet frame sizes, respectively) are the common Internet packet

sizes. 1472 byte packets were used instead of 1518 byte packets, so that no speedup was required for an external-packet/internal-cell switch with a cell size of 64 bytes ($1472 = 64 \cdot 23$). Bimodal synthetic traffic is used since the realistic Ethernet packet length distribution is bimodal, as seen in both USF traced traffic #1 and #2 packet length distributions.

3) *USF traced traffic #2:* See Section 5.1.2.1 for the description.

5.2.3.2 Simulation experiments

Five simulation experiments are designed to evaluate block move. Output port destination configurations based on high-degree balanced, low-degree balanced, and low-degree unbalanced probability density functions [32] are examined using USF distribution (see section 3.2.3). These experiments evaluate how packet switches perform with different output port destination configurations. A diagonal scenario [67] is examined using the bimodal synthetic traffic. This experiment evaluates the fairness among different packet length. For all experiments, one second of simulated time with a 10-Gbps link data rate is assumed unless otherwise stated.

1) *High-degree balanced experiment:* With USF synthetic traffic, each of the 16 input ports chooses an output port with a uniform distribution over the 16 output ports. All input ports and output ports have an identical offered load ranging from 80% to 99%.

2) *Low-degree balanced experiment:* With USF synthetic traffic, each of the 16 input ports chooses an output port with a uniform distribution over k output ports (where $k < 16$). All input ports and output ports have an identical offered load; k

random permutations are used to assign input and output pairs; k is set 4, and the offered load ranged from 80% to 99%.

3) *Low-degree unbalanced experiment:* With USF synthetic traffic, half of the input ports chooses an output port with a uniform distribution over k ports, and the other half chooses an output port with a uniform distribution over $2k$ output ports. Each flow rate is identical. If half the input ports are loaded at λ, the other half of the input ports are loaded at 2λ. Each output port receives either k or $2k$ flows. $2k$ random permutations are used to assign input and output pairs; k is set 4, and the offered load ranged from 80% to 99%.
4) *Diagonal experiment:* With bimodal synthetic traffic, each of the 16 input ports chooses an output port with a uniform distribution over the 16 output ports. Packets from input port i are destined to output port j where $i = j$ for all packets of length 1472 bytes only. Otherwise, packet lengths are 64 bytes. All input ports and output ports have identical offered load, ranging from 80% to 99%.
5) *Traced packet experiment:* The USF traced traffic #2 was used, and modulo N of the destination IP addresses were assigned to each packet as its output port destination. The media rate of the simulation model was modified to achieve a desired offered load of 40 to 60%, and the simulation results were scaled to match the results of a 10-Gpbs switch. Simulation was run until a trace to any one of input port ran out.

5.2.3.3 Experiment results

Figure 5.10 shows the results for the high-degree balanced experiment. It shows that the mean delay of the iSLIP switch with cell train has the highest mean delay, and the iSLIP with cell train becomes unstable above a 96% load. The high mean delay and instability of the iSLIP with cell train is due to the overhead resulting from the transfer of empty padding bytes for every packet whose length is not divisible by 64 (the internal fixed-cell size in bytes). The mean delays of the iSLIP with cell train and 1.05x speed-up, that of the CICQ switch, and that of the CICQ switch with block transfer mechanism

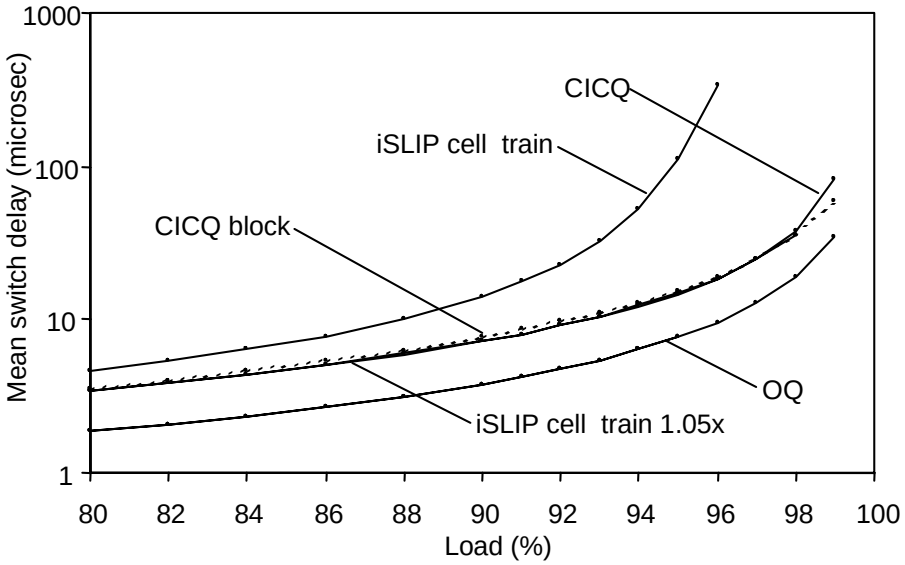

Figure 5.10 – Results for high-degree balanced experiment

are similar, except the CICQ has a slightly higher mean delay than the other two at a 99% load. The speed-up used with the iSLIP with cell train compensates for the transferring of empty bytes, achieving the switch stability for all offered load being measured. Both the CICQ switch and the CICQ switch with block transfer natively supports variable-

length packets, and they do not require speed-ups to achieve stability. As was expected, the OQ switch has the lowest mean delay.

Many matching algorithms have poor performances with unbalanced output port destination [32]. Figure 5.11 shows the results for the low-degree balanced experiment. The iSLIP with cell train, the iSLIP with cell train and 1.05x speed-up, and the CICQ switch became unstable above 90%, 94%, and 94% load, respectively. Similar results were obtained for iSLIP in [32]. The CICQ switch with block transfer mechanism and the OQ switch are stable for all offered loads being measured.

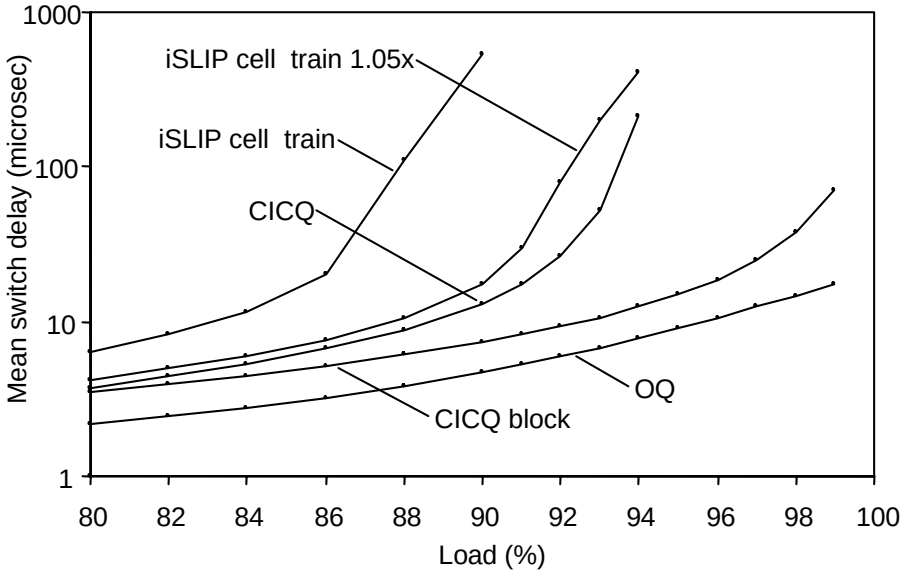

Figure 5.11 – Results for low-degree balanced experiment

As shown in Figure 5.12, the iSLIP with cell train and the iSLIP with cell train and 1.05x speed-up become unstable above a 92% and 96% load, respectively, for the low-degree unbalanced experiment. Thus, both the iSLIP with cell train and the iSLIP with

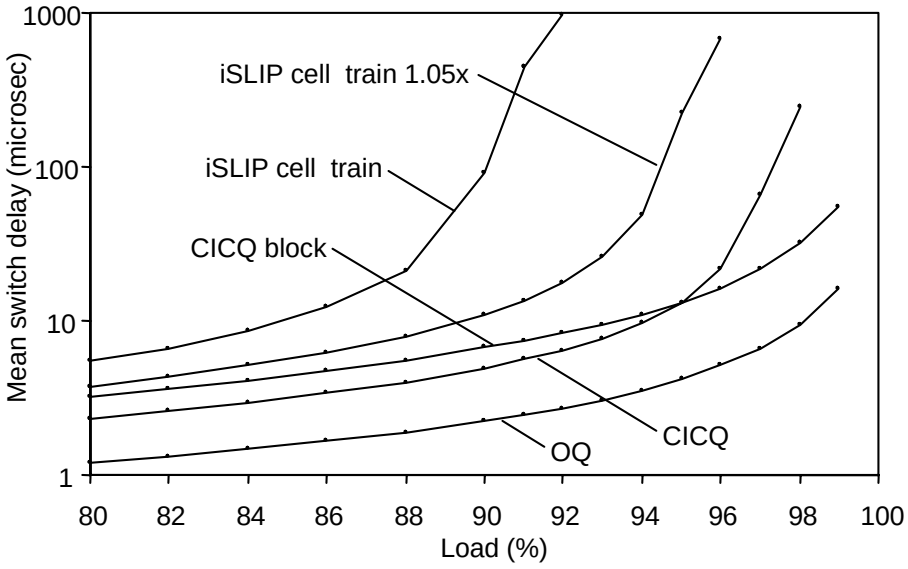

Figure 5.12 – Results for low-degree unbalanced experiment

cell train and 1.05x speed-up reduced the instability region obtained from the low-degree balanced experiment. This is because the total offered load of *N* output ports for the low-degree unbalanced configuration is less than that for the low-degree balanced configuration. Similarly, the instability region of the CICQ switch is reduced. The CICQ with block transfer mechanism and the OQ switch are stable for all offered loads being measured.

Figures 5.13, 5.14, and 5.15 show the results for the diagonal experiment. Figures 5.13 and 5.14 show the mean delays of large and small packets, respectively. The CICQ switch with block transfer mechanism has a higher large packet mean delay than the OQ switch; however, it has a lower small packet mean delay than the OQ switch. The CICQ without block transfer mechanism also has a lower small packet mean delay than the OQ switch at high offered loads.

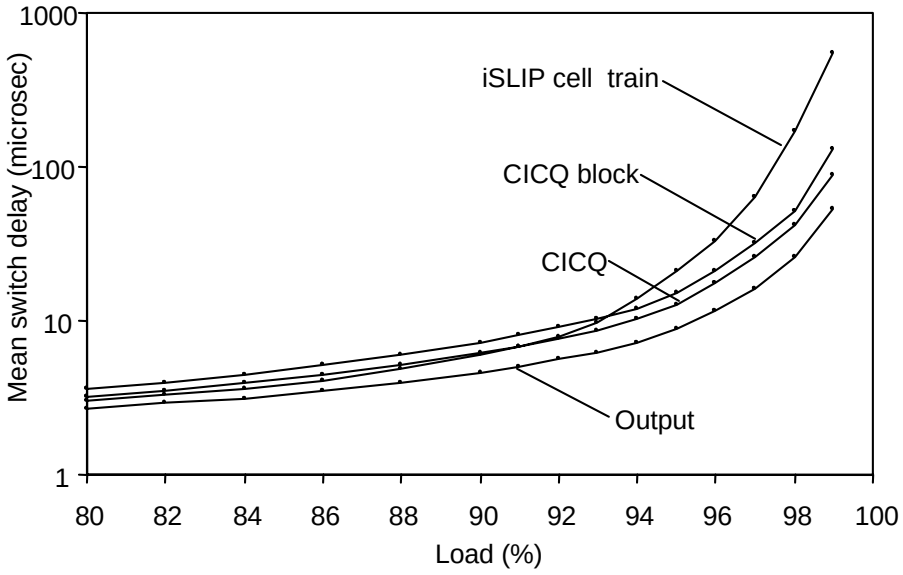

Figure 5.13 – Results for diagonal experiment (large packets)

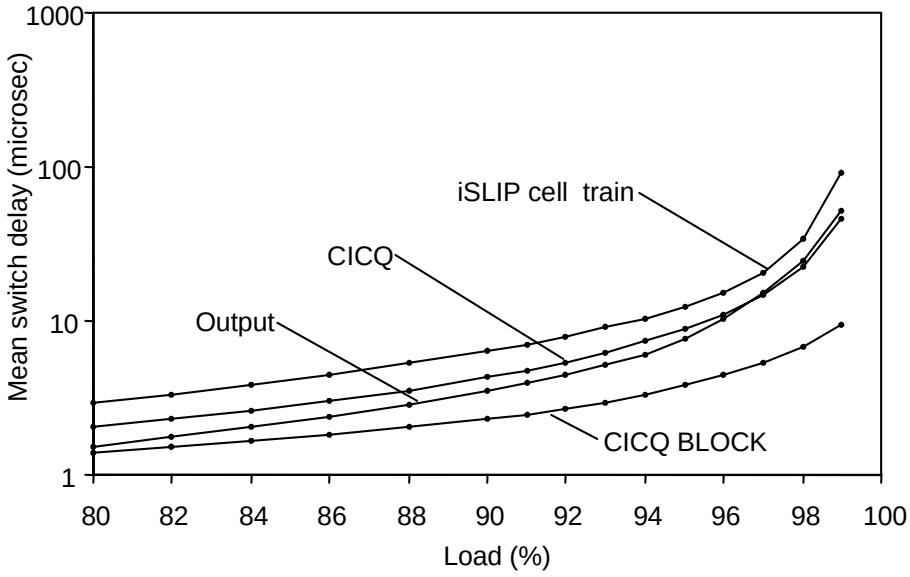

Figure 5.14 – Results for diagonal experiment (small packets)

The OQ switch is purely First Come First Serve (FCFS); however, the CICQ switch, having 2 stages of buffering, introduces priority based on packet lengths. In the CICQ

switch, the transfer time required for packets to transfer from a VOQ to a CP is proportional to the size of the packet, and a smaller packet requires less transfer time from a VOQ to a CP. Suppose the transfer of a small packet from port 1 to $CP_{1,1,}$ $Packet_{11}$, and the transfer of a large packet from port 2 to $CP_{2,1,}$ $Packet_{21}$, begins at the same time. $Packet_{11}$ arrives at $CP_{1,1}$ before $Packet_{21}$ arrives at $CP_{2,1}$. Thus, $Packet_{11}$ will be selected for transmission to an output link via RR scheduling. The effect is demonstrated by the smaller mean delay of the CICQ switch than that of the OQ switch for small packets (as in Figure 5.14) at a high offered load. The CICQ switch with block transfer mechanism further adds the packet size-based priority within each port. With the CICQ switch with block transfer mechanism, a multiple of small packets are scheduled from a single VOQ for each RR polling at an input port, giving higher priority to smaller packets over larger packets within the input port. This explains the small packet mean delay of the CICQ switch with block transfer mechanism being lower than the OQ switch for all offered loads measured. Figure 5.15 shows the mean delay of both large and small packets combined. The CICQ switch with block transfer mechanism has the lowest mean delay. The total number of small packets is about 16 times more than that of large packets (for $N = 16$). Thus, the mean delay of large and small packets combined is dominated by the small packet delay.

For the traced packet experiment, all switches become unstable above a 55% load as shown in Figure 5.16. Internet traffic has a high degree of burst, and packet destinations (and output port to which packets are destined within a switch) are not uniform. Figure 5.17 shows the individual output port utilization, relative to the port with the highest offered load (output port 4), of the OQ switch in the trace packet experiment. Output port

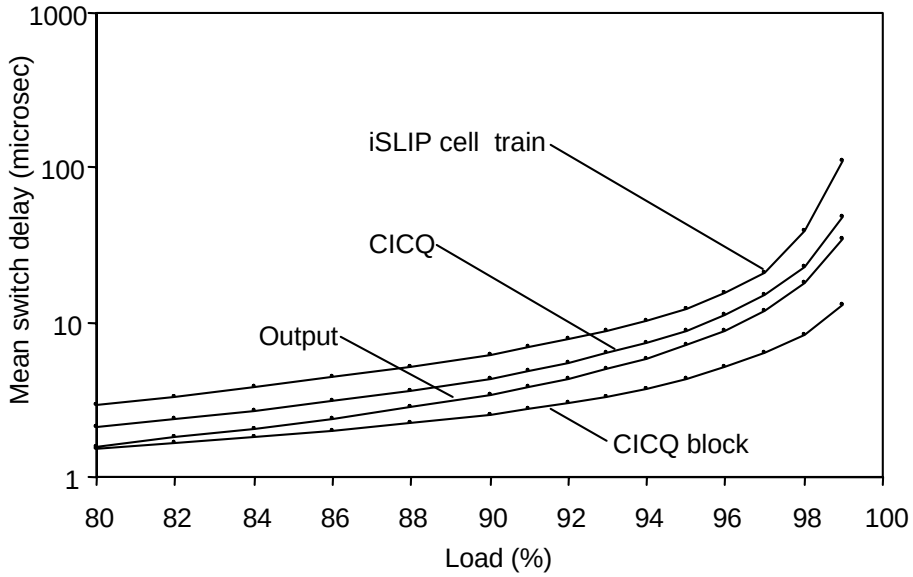

Figure 5.15 – Results for diagonal experiment (small and large packets)

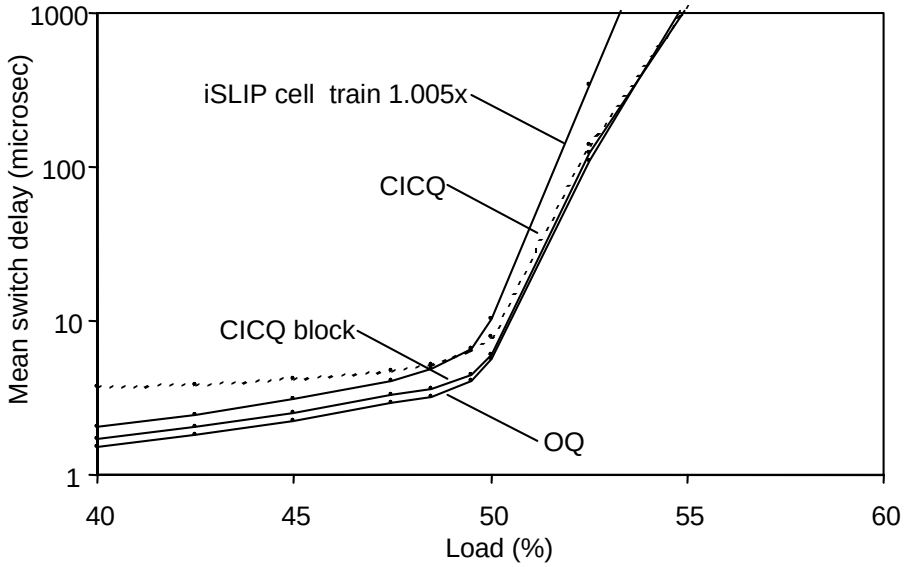

Figure 5.16 – Results for traced packet experiment

has the highest offered load, and its buffer (or corresponding VOQs for port 4 for iSLIP and CICQ switches) becomes saturated with buffered packets before average offered load

among all 16 ports becomes 100%. The instability of the switches at a low (55%) offered load occurs when the intensive traffic to output port 4 reaches 100%. At a 52.5% offered load (a measurement point before instability point), iSLIP with cell train and 1.005x speed-up has a mean delay of 340 microseconds, while the CICQ switch, the CICQ switch with block transfer mechanism, and the OQ switch have similar mean delays of 140, 121, and 110 microseconds, respectively.

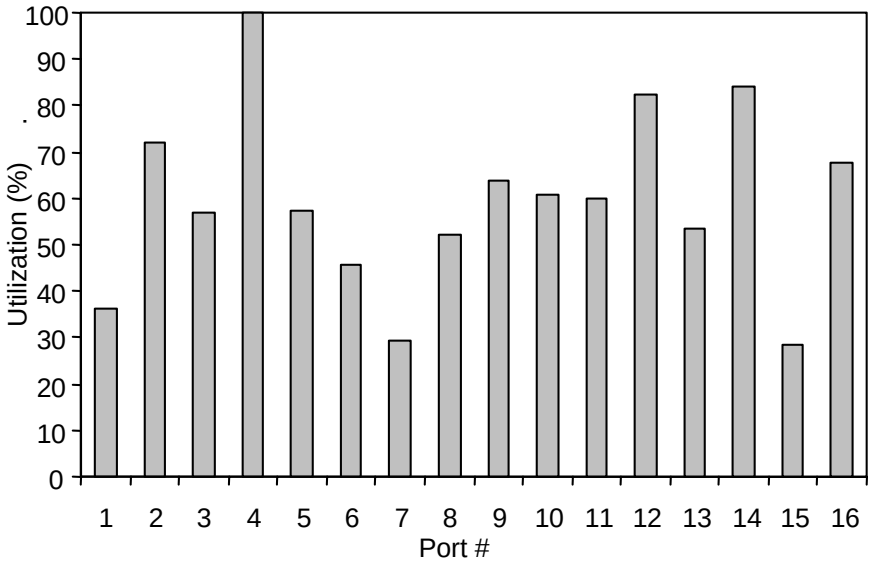

Figure 5.17 – Relative utilization of port for traced packet experiment

Chapter 6: Design and Implementation of CICQ Switches

In this chapter, the design and implementation of CICQ switches are described. The feasibility of the CICQ switch architecture for 24 ports and a 10-Gbps link data rate is shown with an FPGA-based design in section 6.1. The bottleneck of a CICQ switch with RR scheduling is the RR poller. In section 6.2, a priority encoder based RR poller that uses feedback masking was implemented. The feasibility of both the CICQ switch architecture and RR poller using FPGA technology provides a significant promise that the same switch architecture that supports greater link data rates and port sizes can be implemented with custom silicon. To improve the scalability of the RR polling unit in the switch scheduler, a fully scalable arbiter that is independent of the size of the inputs is proposed in section 6.3.

6.1 Design of an FPGA-based CICQ Switch

To show the feasibility of the CICQ switch architecture, a 24-port, 10-Gbps link data rate switch was designed. The target technology was the Xilinx Virtex II Pro series of FPGAs [116]. A Xilinx application note [104] describes a buffered crossbar. Cisco sells the 12416 Internet Router, which supports 10-Gpbs with 16 slots [22]. The cost of the Cisco 12416 is $100,000 [23], which is 400% greater (per port) than the estimated cost of our switch (the cost estimate for our switch is in Section 4.4).

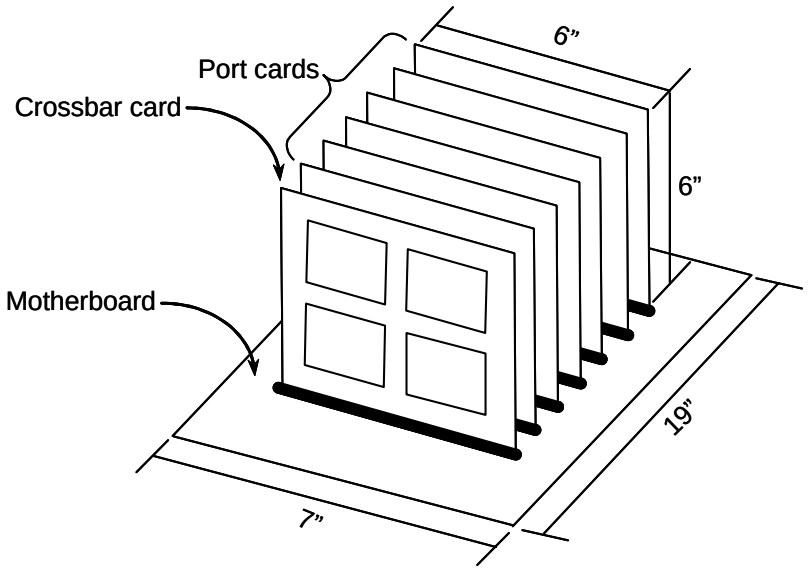

Figure 6.1 – Chassis-level design of FPGA CICQ switch

6.1.1 Chassis-level design

Figure 6.1 shows the chassis-level design of our FPGA-based CICQ switch for 24 ports and 10-Gbps link data rate. The CICQ switch consists of eight circuit boards (six port cards, a crossbar card, and a bus motherboard). The dimensions of the cards are selected to fit a rack based on the Network Equipment Building System (NEBS) standards [7]. A port card has 32 serial I/Os and 24 parallel I/Os, or a total of 56 I/Os. The crossbar card has 196 serial I/Os and 24 parallel I/Os, or a total of 216 I/Os. Each serial I/O of a port card has a dedicated path to/from a serial I/O of crossbar card through the bus board. Each line card shares a bus for its parallel I/Os with other line cards. These I/Os are described in detail in Sections 4.2 and 4.3.

The switch size is constrained by the number of serial I/Os available on the device being used. Serial I/Os available on the Xilinx Virtex II Pro will be increased from 16

(Q1 of 2003) to 24 (Q3 of 2003) [4]. Similar increase is expected to enable the implementation of a 32x32 10-Gbps switch with the same design.

6.1.2 Line card design

Figure 6.2 shows the board-level design of the port card. Each port card consists of four port devices. Each port device is a Xilinx Virtex-II Pro XC2VP20 device and an Intel XPAK TXN17201/9 optical transceiver, which is connected by serial I/Os. Four Rocket I/O transceivers (full-duplex serial 3.125 Gbps channels) handle 10-Gbps data transmission to/from the optical transceiver. Block SelectRAMs (fully synchronized dual port memory on Virtex-II Pro devices) are used to implement VOQs. Each VOQ consists of four 16-bit wide logical data slices each of which is connected to two Rocket I/O transceivers (one to the optical transceiver and another to a crossbar slice). The Media Access Controller (MAC) function can be implemented within the XC2VP20 device [117].

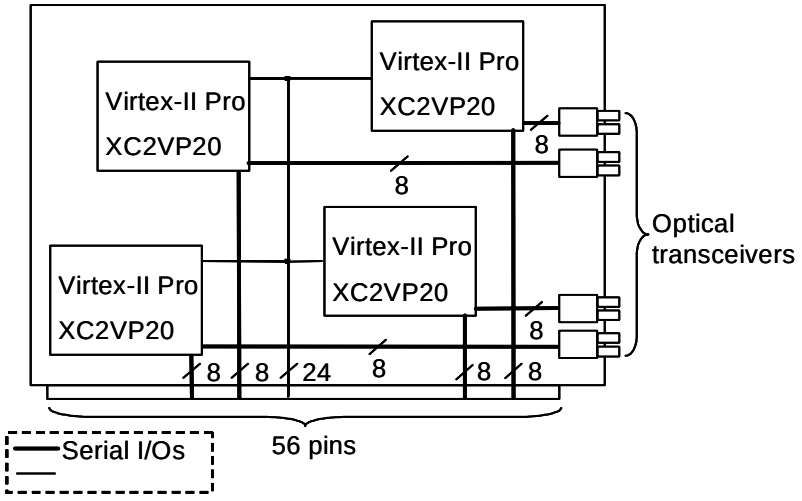

Figure 6.2 – Line card design of FPGA CICQ switch

A total of eight Rocket I/O transceivers are used: two Rocket I/O transceivers per data slice for four data slices. The XC2VP20 has a total of 200 Kbytes of storage that can store 3096 64-byte cells (it takes about 150 microseconds to drain a queue at 10-Gbps line rate). A queueing delay of over 100 microseconds would likely be undesirable in an operational 10-Gbps switch. Parallel I/Os (24 bit) are used at each port device to receive the state of crossbar buffer occupancy from the crossbar card.

6.1.3 Buffered crossbar design

Figure 6.3 shows the board-level design of a crossbar card. The crossbar card consists of four crossbar slices. Each crossbar slice is implemented using a Xilinx Virtex-II Pro XC2VP125 device. Crossbar slices have 24 Rocket I/O transceivers, each of them are mapped to a unique port I/O. Thus, a total of 96 Rocket I/O transceivers (24 Rocket I/O transceivers per crossbar slice times 4 crossbar slices) are used in the crossbar card. Parallel I/Os (24 bit) are used at one of the four crossbar slices to transmit the state of crossbar buffer occupancy to all of the 24 port devices in 24 clock cycles.

Block SelectRAMs are used to implement cross point buffers. The XC2VP125 has 556, 18 Kbit BlockRAMs, which is a sufficient enough number to implement 552 (23 *x* 23) individual CP buffers (For a switch with *N* ports, $(N-1)x(N-1)$ CP buffers are required since transmission to and from the same port is unnecessary).

The data width of the Block SelectRAM ports is configured as 16-bits to match the data slice width of the VOQs. Schedulers are needed in the crossbar slices and ports. For a CICQ switch, round robin polling is used in the port cards (one poller per input port to poll VOQs) and the cross bar card (one poller per crossbar slice column of CP buffers

dedicated to an output port). Thus, no communication between the crossbar slices is needed. The poller is the performance bottleneck for a CICQ switch, where it is required that a complete poll of all VOQs in an input port (or all CP buffers for an output port) be completed in a time frame that is less than the time required to forward a single cell. This bottleneck is addressed in Section 6.2 and 6.3 of this dissertation.

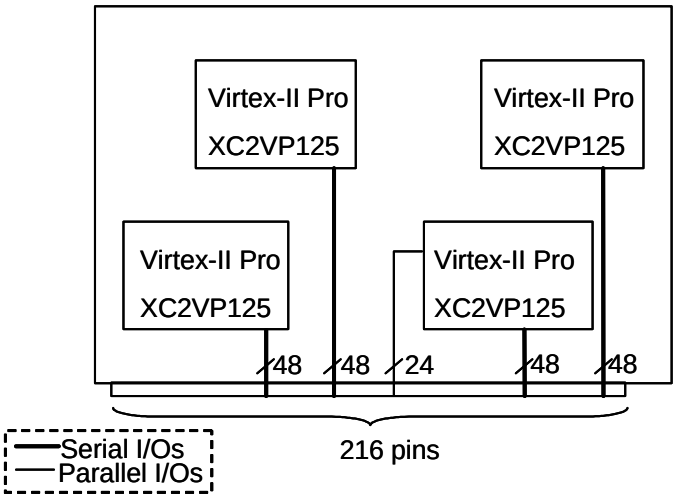

Figure 6.3 – Buffered crossbar design of FPGA CICQ switch

6.1.4 Cost estimate of the FPGA design

The parts used for the port devices, crossbar slice, and optical (fiber to/from copper) transceiver dominate the total cost of implementation. The cost of an XC2VP20 device, a Xilinx chip used for a port device, is under $500 for a purchase of over 100 units [87]. Although the price of an XC2VP125 device (a Xilinx chip used for a crossbar slice) is not specified as of September 2002, it is estimated to be no more than $1000. The Intel XPAK TXN17201/9 optical transceiver costs $500 [45]. Thus, the 10-Gbps switch with 24-port can be built for the estimated price of $30,000 ($500 x 24 port devices,

$1,000 x 4 crossbar slices, $500 x 24 optical transceivers, and less than $2,000 for chassis, boards, connectors, power supply, etc).

6.2 Fast RR arbiter

Many switch architectures, including iSLIP [71] and DRRM [12], use round robin (RR) arbiters as part of their switch matrix scheduling. The CICQ switch uses two levels of RR arbitration. A good switch matrix scheduler should enable 100% throughput for all schedulable offered loads and be feasible to implement.

Round robin polling is the bottleneck for increasing the link data rate and/or the number of ports in a switch. The worst case poll is *N* ports. For a 10-Gbps switch with 16 ports and 64 byte cell size, this is 3.2 nanoseconds per port. An increase in the switch size makes this a great challenge. Thus, to improve the scalability of switch designs that use round robin scheduling, new and faster methods of round robin arbitration need to be investigated. To achieve this requirement, time over space optimization can be justified. In this section, delay with increase in a cost of (cell level) space for an FPGA implementation is improved upon existing designs [38].

6.2.1 Existing fast RR arbiter designs

A priority encoder was implemented in [27] at the CMOS transistor level. This implementation uses a priority look-ahead approach that is similar to look-ahead adders with 4.1 nanosecond priority encoding for 32-bit input and 1 micron implementation. For this study, a fast round-robin poller was implemented at the logic gate level, where the individual priority encoder block can be implemented at either the gate or transistor level.

The most common design for a round-robin poller is the double barrel-shift RR poller (called SHFT_ENC in [38]), which is shown in Figure 6.4. It consists of two barrel shifters and a simple priority encoder, smpl_pe. Request bits *Req* of size *N* are rotated by an amount *P_enc* (*P_enc* is $\log_2(N)$ bits) indicating the VOQ with the currently selected buffer. This is inputted into a smpl_pe and again rotated by *P_enc* in the reverse direction. The outputs are grant bits *Gnt* of size *N* and a bit, *anyGnt*, indicating whether there is a grant. For example, let *Req* = 10110100, *P_enc* = 011, *Gnt* = 00000100, and $N = 8$. First, *Req* is shifted 4 positions to get 01001011. Second, the smpl_pe outputs 01000000. Finally, 01000000 is shifted 4 positions right to obtain 00000100.

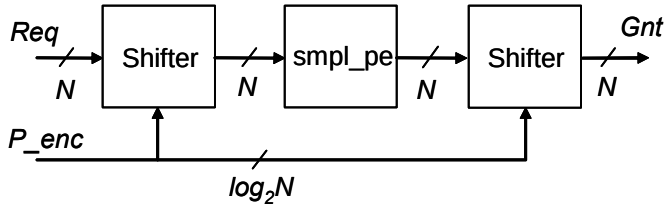

Figure 6.4 – Double barrel-shift RR poller [38]

The barrel shifters dominate the critical path delay. Other designs (in [38]) include fully serial ripple (RIPPLE), fully parallel exhaustive (EXH), and a new look-ahead approach called PROPOSED (shown in Figure 6.5). The PROPOSED design, which has better delay performance than any existing design, eliminates the programmable part of a programmable priority encoder (PPE) by pre-processing inputs. This is done by "thermometer" encoding of a $\log_2(N)$-bit wide vector *x* into an *N*-bit wide vector *y* with an equation, $y[i] = 1 \; iff \; (i < value(x)), \forall 0 \le i \le N$.

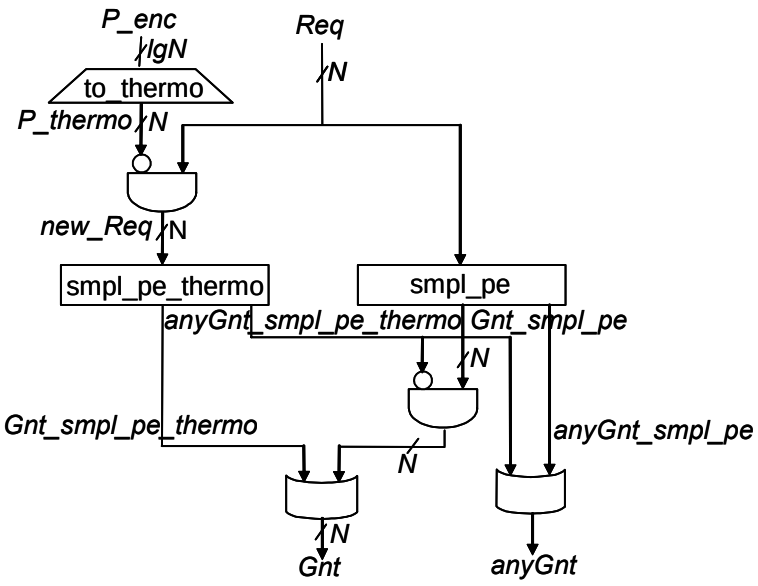

Figure 6.5 – McKeown PROPOSED RR poller [38]

The PROPOSED design eliminates a combinational feedback loop (e.g., as found in the carry look-ahead (CLA) design) that is difficult for synthesis tools to optimize, and a long critical path caused by a programmable highest priority level [38]. The four basic steps in the PROPOSED are shown in Figure 6.6.

1. Transform *P_enc* to thermometer encoded *P_thermo* using the thermometer encoder to *thermo*.

2. Bit-wise AND $\overline{P\_thermo}$ with *Req* to get *new_Req*.

3. Feed *Req* and *new_Req* into *smpl_pe* and *smple_pe_thermo* (another instance of a simple priority encoder), respectively.

4. Select *Gnt_smpl_pe_thermo* (the encoded value from smple_pe_thermo) if non-zero and select *Gnt_smpl_pe* (the encoded value by smpl_pe), otherwise (a bit signal *anyGnt_smple_pe_thermo* and *anyGnt_smpl_pe* are ORed used to compute *anyGnt*).

Figure 6.6 – McKeown's PROPOSED algorithm [30]

6.2.2 Masked priority encoder

For this study, the new Masked Priority Encoder (MPE) poller design (shown in Figure 6.7) was developed and evaluated. The MPE is a priority encoder that uses bit-wise masking to select an appropriate VOQ. As with the PROPOSED design, no programmability is required, and only a smpl_pe is needed. The four basic steps (marked in Figure 6.7) in the MPE are shown in Figure 6.8. The masking bits are generated (for $i = 0,1,...,N$) by $Msk[i] = \overline{Gnt[i]} \cdot \overline{Gnt[i+1]} \cdot \overline{Gnt[i+2]} \cdot ... \cdot \overline{Gnt[N-1]}$. The MPE directly uses a previously derived N-bit grant value for the next polling. Thus, it neither requires an encoder nor decoder to convert the N-bit form to or from a $\log_2(N)$-bit form. Figure 6.9 shows the logic diagram of the MPE poller for $N = 4$.

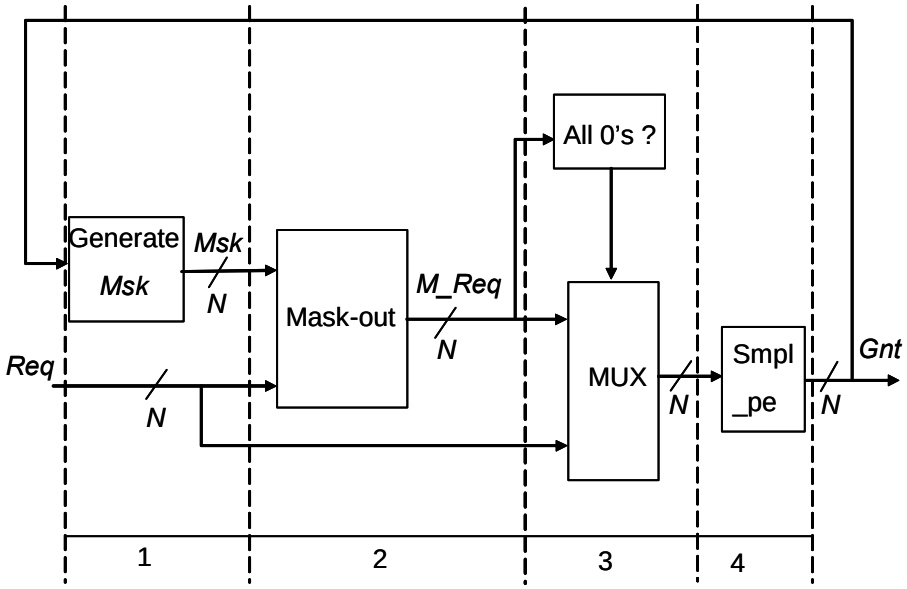

Figure 6.7 – MPE RR arbiter design (block diagram)

1. Generate masking bits *Msk* based on the previously selected VOQ value.

2. Mask out request bits that are equal or less than the value of the previously selected VOQ.

3. If masked requests *M_req* is nonzero then select *M_req*. Otherwise select *Req*.

4. Encode the selected bits (*M_req* or *Req*) with smpl_pe.

Figure 6.8 – MPE RR arbiter algorithm

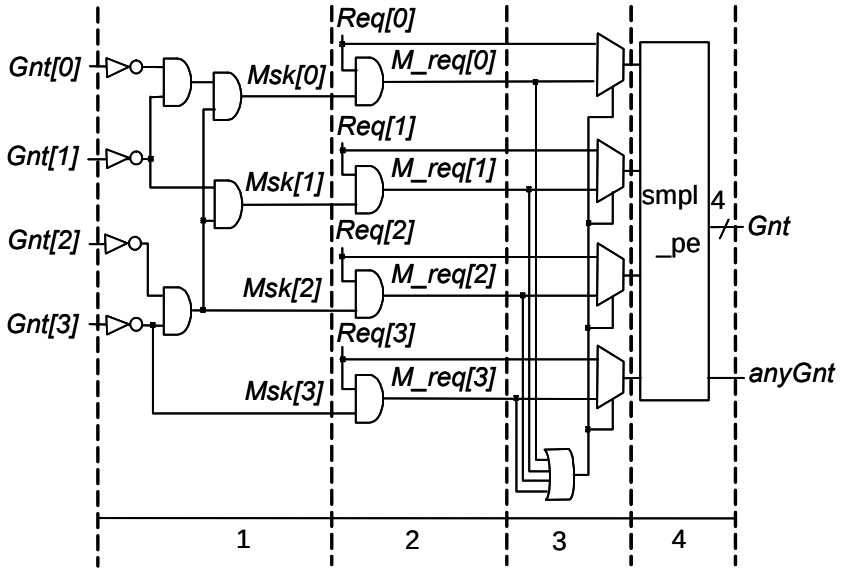

Figure 6.9 – MPE RR poller design (logic diagram)

6.2.3 Evaluation of MPE

In this study the programmable priority encoder RR poller designs in [38] were implemented using VHDL, and they were simulated with the Xilinx WebPACK 4.2 ModelSim XE [118]. The targeted device was the Xilinx Virtex II XC2V40 FG256. Simulations were run with the same time and space optimization settings for all designs.

The delay and space requirement was measured for each design. Table 6.1 shows the delay (in nanoseconds) and Table 6.2 shows the space in basic elements (BEL) of the round robin poller designs. BELs are the building blocks that make up a component configurable logic block (CLB) for FPGA, which includes function generators, flip-flops, carry logic, and RAM. The relative results do not exactly match with the results in [38]. In [38] two input gate equivalents are used to size the designs. The design in [38] uses a digital signal processor (DSP) as the target device; however, the target device used for this study was an FPGA, which is capable of handling the high-speed data on the chip. This difference in targeted devices results in the use of different simulation tools and configurations.

Study results show that the MPE has lower delay than any other design for all measured values of N. However, it requires more space than any other design, except EXH. The evaluation results in Tables 6.1 and 6.2 show this, with the last row indicating the improvement of MPE over the design with the next best performance. With modern VLSI technology, space is rarely the constraining factor. The better delay performance of the MPE is due to the fact that the MPE uses N bits to determine the value for the next poll. The MPE does not require an encoder or decoder to convert the N-bit form to and from a $\log_2(N)$-bit form, which would result in a speed-up at the cost of space required to accommodate N bits versus $\log_2(N)$ bits. In addition, since encoders and decoders are so common in FPGA designs, many synthesizers have space-optimized models for them. The two designs that do not require an encoder or decoder, SHFT_ENC and MPE, are similar in size possibly because a space-optimized macro was not used.

Table 6. 1 – Evaluation of delay (nanoseconds)

Design	$N = 8$	$N = 16$	$N = 32$	$N = 64$
RIPPLE	17	24	41	73
CLA	14	17	23	23
EXH	10	16	26	50
SHFT_ENC	15	24	37	64
PROPOSED	13	21	33	55
MPE	10	11	1 3	16
Improvement	0.0 %	47.6 %	43.5 %	30.4 %

Table 6.2 – Evaluation of space (FPGA BELs)

Design	$N = 8$	$N = 16$	$N = 32$	$N = 64$
RIPPLE	17	31	126	380
CLA	21	41	145	418
EXH	132	473	2391	10134
SHFT_ENC	58	143	350	836
PROPOSED	37	74	150	318
MPE	65	134	355	798
Improvement	–282.6%	–332.3	–181.7 %	–150.9 %

6.3 Scalable RR arbiter

A scheduler must be work-conserving. For a switch to be work-conserving, output port links must be fully utilized as long as any cells destined to them exist at any input ports. A switch that is non-work-conserving can not achieve 100% throughput. The RR arbiter is one of the bottlenecks of a switch as the number of ports and link rates increase.

Fast schedulers are needed to support ever-increasing switch size and link data rate. For example, for a switch with 16 ports and 100-Gbps link data rates, a 64-byte cell has to be forwarded every 5.12 nanoseconds, and a scheduling cycle also completed in this time.

In this study a scheduler for N queues was considered. Each queue buffers fixed length cells. A scheduler must exhibit fairness (i.e., short and long term fairness). In a *short-term* scheduler, all N nodes receive an opportunity to forward a cell within every N cell forwarding time. In a *long-term fair* scheduler, all N nodes receive an opportunity to forward a cell in a finite time (i.e., scheduling delay is bounded).

6.3.1 Existing scalable RR arbiters

In the context of RR scheduling, a slotted system with N queues, Q_i, was considered, where $i = 1.2,..., N$. Each queue buffers fixed-length cells arriving from external sources. This is a time slotted system where a slot either contains a cell or is empty. The cell queues may correspond to Virtual Output Queues (VOQ) at a switch input port. The time to forward a cell is T_c.

A poller visits queues, Q_i, in sequential RR fashion with a delay of T_p for each queue visited. The delay T_p occurs whether the visited queue is occupied or empty. If $N \cdot T_p < T_c$ then a simple two-stage RR arbiter that can select the next queue while the currently selected queue is forwarding a cell is sufficient, meaning that the arbiter can be work-conserving, such that the output link is never idle if there are cells queued in any of the N queues. Most existing RR arbiters are based on a two-stage approach when scheduling is done simultaneously with cell transmission. That is, the cell forwarded in

time slot t_i was scheduled in time slot t_{i-1}. In existing two-stage designs, 100% throughput can only be achieved if $N \cdot T_p < T_c$. For the purposes of this study, an overlapped round robin (ORR) arbiter is proposed, which will remove this limitation.

Existing RR arbiters can be categorized as follows:

1) Sequential polling
2) Non-sequential polling
3) Tree arbitration
4) Pipeline structures

Sequential polling is the simplest implementation of RR polling. A RIPPLE design is described in [38]. Such a design has O(*N*) scalability with each new node adding a delay T_p to the scheduling delay. Token tunneling, a form of non-sequential polling, proposed in [13], allows a pointer to skip a set of ports if none of the ports has packets to send. Token tunneling reduces the arbitration time of sequential polling to $O(\sqrt{N})$.

Tree-structured RR arbiter designs [14], [38], [103], [122], [124] reduce the arbitration time over a sequential design. The arbitration time of the Ping-Pong Arbitration (PPA) scheme [14] for an *N*-input switch is proportional to $\log_2\lceil N/2 \rceil$. The arbitration time is only 11 gate delays for a 256x256 switch. The Parallel RR Arbiter (PRRA) using a binary tree structure is presented in [124]. As with PPA it has $O(\log_2(N))$ gate delays, but it resolves an unfairness problem in PPA. Arbitration time of the tree-structured RR arbiters described in [38], [103], [122] is also $O(\log_2(N))$.

Pipelined arbiter designs are used in several switch architectures [43], [82], [105]. Round-robin greedy scheduling (RRGS) [105] scales to a large switch because the

amount of information transferred among function modules is small. However, the scheduling delay increases in proportion to the number of switch ports, and scheduling can be unfair. The Group-Pipeline Scheduler (GPS) [82] improves on RRGS by dividing N nodes into K groups (N / K nodes per a group), and assigns an RRGS function module to each group. It has a smaller arbitration time and better fairness than RRGS. An FPGA implementation of RR scheduling using a Pipelined Priority Encoder and Barrel Shifter is presented in [43]. Encoded bits are divided and inputted into multiple smaller priority encoder units. Outputs from these units are merged and inputted into another priority encoder to obtain the final encoded result. An arbitration time of $O(\log_2(N))$ can be obtained at the cost of using more FPGA space.

Some pipelined arbiters are not of a two-stage design. One example is the pipeline-based concurrent round-robin dispatching scheme using multiple subschedulers [89]. Each subscheduler provides a dispatching result in every P scheduling cycle, where P is the number of subschedulers. The arbitration time required for scheduling an IQ switch can be relaxed by using a four-stage (cell arrival/departure, request transmission, arbitration, and grant transmission) pipelined operation [47].

The scheduling delay of existing RR arbiter designs is a function of N. For some values of N and T_c the design becomes insufficient to achieve 100% throughput. An RR arbitration scheme that can schedule and achieve 100% throughput independent of N and T_c is needed.

6.3.2 Overlapped RR arbiter

To improve the scalability of RR polling, an Overlapped Round Robin (ORR) arbiter is proposed that fully overlaps polling and cell scheduling [123]. In a system of N queues, each queue has one control input (*select*) and one control output (*arrival*). Figure 6.10 shows a cell queue and Figure 6.11 shows the system of N queues with an (a) RR polling unit and (b) cell scheduling unit. The polling algorithm is shown in Figure 6.12 and the scheduling algorithm in Figure 6.13. A counter $C1_i$ is incremented on cell arrivals to Q_i and decremented on scheduled cell departures. The *arrival* output causes the increment of $C1_i$. The decrement of $C1_i$ is caused by the scheduling algorithm. The counter $C1_i$ represents the number of cells currently queued in Q_i. A counter $C2_i$ is decremented on cell departures from Q_i and is increased in the polling algorithm shown in Figure 6.11. The counter $C2_i$ represents the number of cells in a queue "marked" for forwarding. At all times, $C1_i - C2_i \geq 0$. The input *select* is used to select a queue for forwarding cells. Only one *select* line can be active on any given cell slot. A single counter $C3$ representing the number of cells permitted to be forwarded in the scheduling

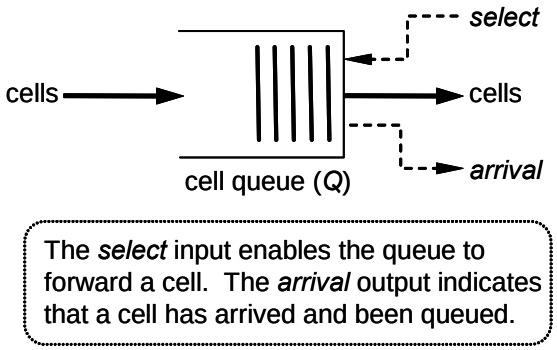

Figure 6.10 – Queue with control and data lines

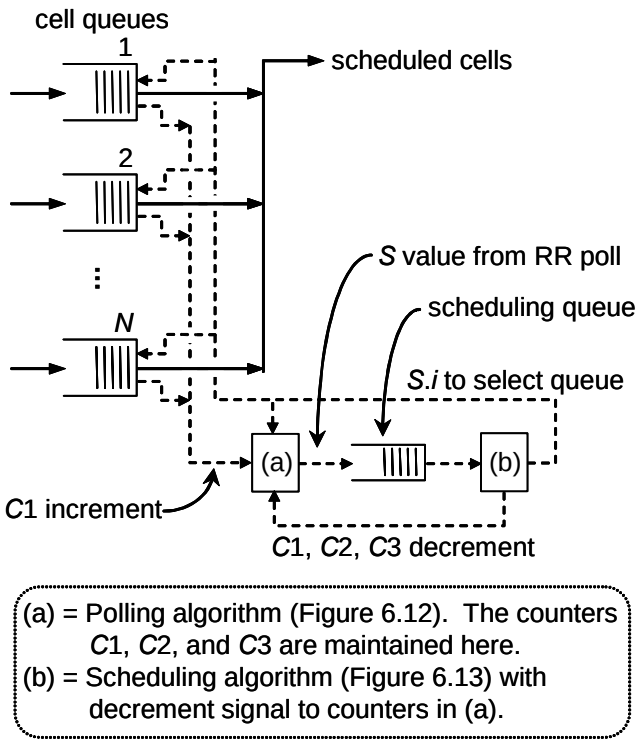

Figure 6.11 – Cell queues and scheduling queue

```
1.  do forever
2.      i = mod(i, N) + 1
3.      while (C3 > K) wait
4.      mark = min(K, C1_i – C2_i)
5.      if (mark > 0)
6.          C2_i = C2_i + mark
7.          C3 = C3 + mark
8.          S.i = i
9.          S.m = mark
10.         queue S to the scheduling queue
```

Figure 6.12 – Polling algorithm

queue is also maintained. All counters are stored in the polling unit. A constant value, K, is used in the polling unit. The counter $C3$ and the setting of K are described later.

The polling algorithm (Figure 6.12) "visits" each queue by testing whether $C1_i - C2_i > 0$. (line 4). If this holds, then there are unmarked cells in the queue. When a queue is visited and the *mark* value (line 4) is non-zero, the counters $C2_i$ and $C3$ are updated and a scheduling value, *S*, comprising the queue index, *i*, concatenated with the number of cells marked in this visit, *m* $(1 \le m \le K)$, is queued in a special scheduling queue. The polling time T_p is incurred in lines 2 to 10 of the polling algorithm and in the time to increment $C1_i$ For this study, the notation *S.i* and *S.m* are used to mean the index value and marked cell count, respectively, for a given value of *S*. The value *S* is of size $\log_2(N) + \log_2(K)$ bits. Line 3 in the polling algorithm stops the polling if the value of $C3$ exceeds *K*. The counter $C3$ contains the sum of *S.m* currently queued in the service queue. The poll stopping in line 3 is essential to improving long-term fairness, and its properties are discussed later in this section.

```
1.  do forever
2.      if (the scheduling queue is non-empty)
3.          S = dequeue from scheduling queue
4.          set select for queue S.i
5.          do for j = 1 to S.m
6.              wait for a cell to finish forwarding
7.              decrement C1_S.i, C2_S.i, and C3
8.          reset select for queue number S.i
```

Figure 6.13 – Scheduling algorithm

The scheduling algorithm (Figure 6.13) dequeues *S* from the scheduling queue when all currently scheduled cells have been forwarded. For example, if the currently dequeued scheduling value has *S.m* equal to 3, then after 3 cell forwarding times, the next

queued scheduling value will be dequeued. The index *S.i* is the queue to be issued a *select* for forwarding of *S.m* cells. The polling and scheduling algorithms run concurrently. The value of *K* is set so that work conservation is achieved for all possible cases of queued cells in the *N* queues. The value of *K* also binds the maximum delay a cell arriving to an empty queue will experience.

Lemma. The smallest integer K needed for the ORR scheduling to achieve work conservation for all possible cases of queued cells in the N queues can be derived as

$$K = \left\lceil N \frac{T_p}{T_c} \right\rceil. \tag{6.1}$$

Proof. A time to poll all *N* nodes; NT_p, divided by T_c is the total cell forwarding time required to poll all *N* nodes. That is, NT_p / T_c cells are forwarded during one RR scheduling cycle. If this RR scheduling time is less than the cell forwarding rate, at least one cell is scheduled during one cell forwarding time. Thus the system becomes work-conserving for any $K > NT_p / T_c$. The least integer greater than or equal to NT_p / T_c is a ceiling of NT_p / T_c.

Theorem. A new HOL cell at any queue of the ORR arbiter can be forwarded in less than $K \cdot (N-1) + 2K + 1$ *cell forwarding time.*

Proof. By definition, the ORR poller visits any of *N* queues in every *N* polling time where each queue marks up to *K* cells per polling time. Thus, a HOL cell at any queue has to wait, at most, $K(N-1)$ cell forwarding time if the scheduled queue was empty. Since the sum of *S.m* in the scheduling queue can be as large as $2K$, a new HOL cell at any queue has to wait, at, most $K(N-1) + 2K$ cell forwarding time.

6.3.3 The ORR arbiter in the CICQ switch

The ORR arbiter can be used to implement the RR arbitration in the CICQ switch. The input ports require knowledge of CP occupancy from each cross point to prevent the overflow of the CP. Each of N CPs, CP_i where $i = 1,2,...,\ N$ associated with VOQ_i sends one bit of CP status, F_i, to its input port. For a CP buffer size of $3 \cdot K$ cells, F_i of 0 is sent if the occupancy at CP_i is below K. An F_i of 1 is sent if the occupancy at CP_i is at or above K (at most $2 \cdot K$ cells destined to CP_i may be queued in the scheduling queue of the ORR arbiter). Thus, F_i controls the operation of the line 5 of the polling algorithm in Figure 6.12 as if $((mark > 0)$ and $(F_i == 0))$. The RR arbiters in the crossbar can be replaced with the ORR arbiters in the same fashion.

6.3.4 Evaluation of the ORR arbiter

In this study, the performance of the ORR arbiter was compared with that of a two-stage RR arbiter and an ideal sequential RR arbiter with zero polling time. For the two-stage RR arbiter, the scheduling time is assumed to be exactly T_c. The ideal RR arbiter can not be implemented, but serves as a useful comparative lower bound. For all experiments, $K = N$, $T_p = T_c$, and a simulation time of 1 million cell times was used, unless otherwise stated. Cell arrivals were Bernoulli distributed, unless otherwise stated.

6.3.5 Simulation experiments

Experiments were performed to evaluate work conservation and fairness, and to characterize the output process. The experiments were as follows:

1) *Work conservation experiment:* For $N = 4$, M queues are saturated with 1000 cells each. The drain time of the queues is measured. The value of M ranges from 1 to 4, and K ranges from 1 to 8.

2) *Fairness experiment #1:* For $N = 4$ the arrival rates are set to 0.1λ for queue 1, 0.2λ for queue 2, 0.3λ for queue 3, and 0.4λ for queue 4, with λ ranging from 0.50 to 0.99. The mean queueing delay is measured for each of the four queues. Instability was also tested, by checking for any queue length exceeding 5000 cells in 100 million cell times. (Evaluating stability in this experimental manner was also done in [32].)

3) *Fairness experiment #2:* For $N = 16$, 1 to 15 queues are saturated, and tagged cells arrive at a low rate (in randomly chosen time slots) to the empty queue. The scheduling delay of the tagged cells is measured. This experiment demonstrates the head-of-line cell forwarding time bound derived in the theorem of Section 3.

4) *Output characterization experiment:* For $N = 16$ the arrival rates are set to λ / N for each queue, with λ ranging from 0.60 to 0.90. The value of K ranged from 2 to 16. The coefficient of variation of the output interdeparture times was measured for a simulation run consisting of a 10 million cell times.

6.3.6 Experiment results

Figure 6.14 shows the results for the work conservation experiment. For the ORR, all queues are entirely drained in $1000 \cdot M$ cell times for $K \geq 4$ for all M measured. This supports the K derived in Eq. (1). For the two-stage RR and the ideal RR, all queues are also entirely drained in $1000 \cdot M$ cell times.

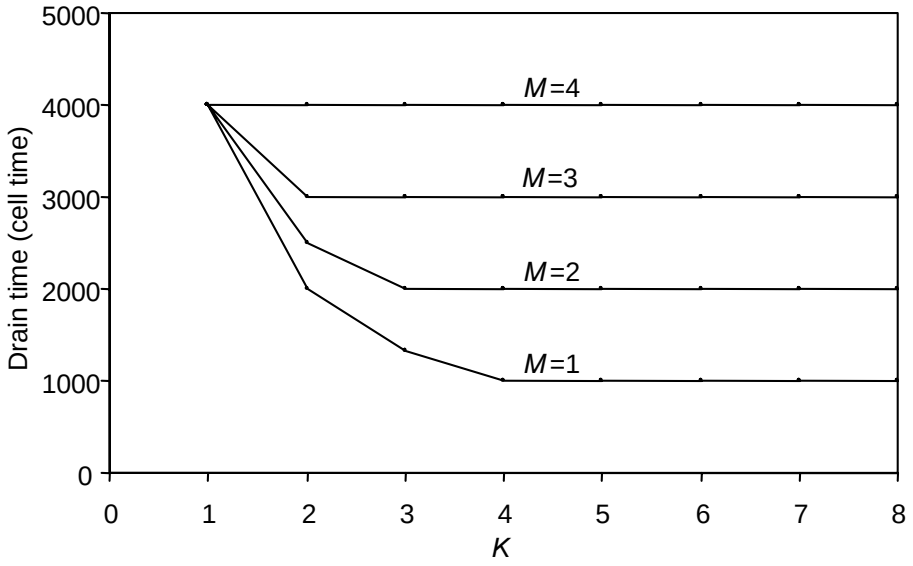

Figure 6.14 – Results for the work conservation experiment

Only the results for the fairness experiment #1 with Q_1 and Q_4 are shown in Figure 6.15. Results show that the mean queueing delay for each queue scheduled by the ORR is greater than the mean queueing delay scheduled by the two-stage RR arbiter and the ideal RR arbiter for all λ measured. The ORR, two-stage RR, and ideal RR are all stable and thus also long-term fair. Fairness experiment #2 resulted in a maximum measured scheduling delay of 240 cell times. The maximum, according to the theorem of Section 6.3.2, is 273 cell times.

Figure 6.16 shows the results for the output characterization experiment. The coefficient of variation of the output interdeparture times for the two-stage RR and the ideal RR is smaller than the coefficient of variation of interarrival time for all λ measured. The coefficient of variation of interdeparture time for the ORR, with $K = 8$ and 16, is greater than that of Bernoulli traffic for all λ measured. The coefficient of variation of interdeparture time for the ORR is smaller than that of

Bernoulli traffic for all λ measured only with $K = 2$. These results show that the value of *K* used for the ORR arbiter influences the output traffic characteristics.

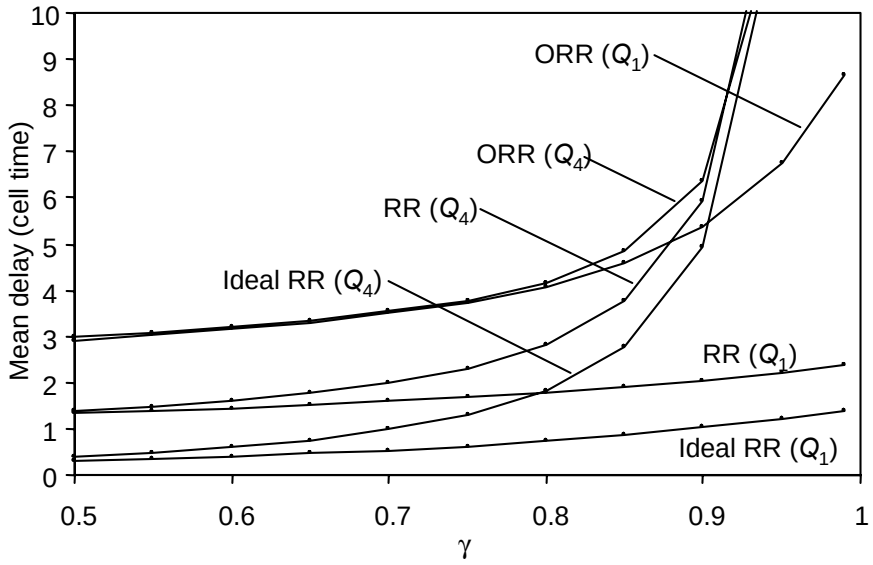

Figure 6.15 – Results for fairness experiment #1

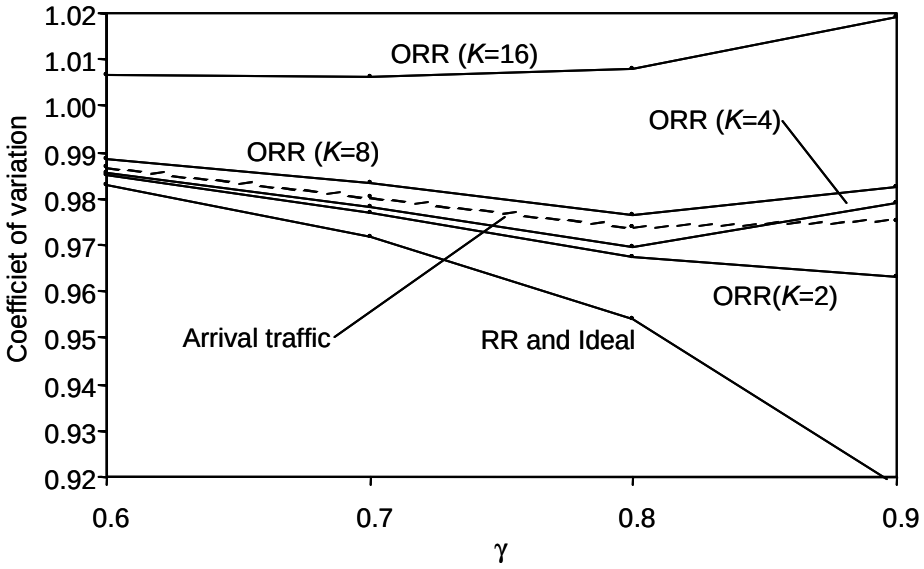

Figure 6.16 – Results for output characterization experiment

Chapter 7: Scalable CICQ Switches

This chapter investigates the scalability of future CICQ switches. Problems incurred with CICQ switches for ever-increasing link data rates (40 to 160 Gbps) are described and resolved using a distributed rate controller. It is proven that the rate controller will significantly reduce the CP buffer requirement. Simulation evaluation demonstrates that the delay overhead due to the distributed rate controller is acceptable.

7.1 Scalability of existing packet switch

Until recently, a packet switch was built in a single cabinet as shown in Figure 7.1(a). To accommodate the growth of the Internet traffic, current packet switches handle OC 768 (40-Gbps) link data rates and will need to handle 3072 (160-Gbps) and higher link data rates in the near future. These future link data rates are likely to be an aggregate of a large number of OC192 link data rates, due to the limitations of the SRAM clock cycle [80]. Consequently, a large number of line cards are required per packet switch [80]. This results in the following:

1) Increases in physical space
2) Increases in power consumption

These phenomena consequently required that a new packet switch architecture be distributed: line cards and switch fabrics are distributed in multiple cabinets as shown in

figure 7.2(b). New routers, including Alcatel 7670 RSP [2], Avici TSR [5], and Juniper TX8/T640 [49], are all designed in this manner. Thus, the new packet switch architecture requires the interconnection between the line cards and the switch fabric to be 10s meters apart [80]. Consequently, the round-trip time (RTT) delay internal to the switch must be taken into account in designing the next generation of CICQ switches.

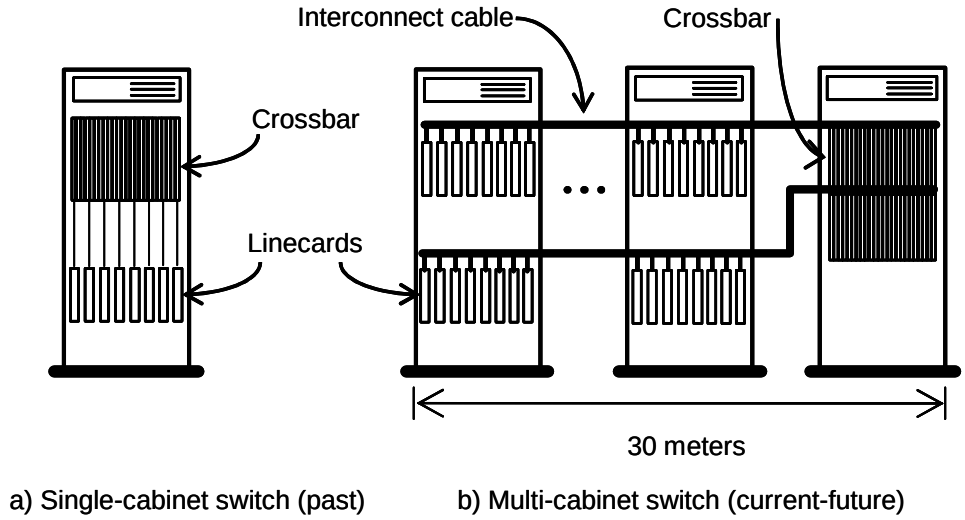

Figure 7.1 – Trend in switch design

In 1993, Link level credit-based flow control (FC) for ATM networks was studied [57]. In [113], it was shown that RTT delay significantly increases contention on output ports for VOQ IQ switches based on parallel and iterative scheduling algorithms [113]. The performance evaluation of CICQ switches with RTT delay is studied in [1]; and the backpressure in IQ and OQ switches is studied in [100]. ATLAS I (ATm multi-LAne backpressure Switch One) is a single-chip gigabit ATM switch with optional credit-based flow control that returns two credits per packet cycle [52], [62]. Credits are returned per input using an FIFO discipline, and the credit memory must be large enough to hold all

credits that are allowed to circulate per adapter/switch input pair [52]. Thus, the FIFO size is proportional to the number of ports N and the memory size M assigned per switch input/output pair, and has the complexity O(MN). Furthermore, the FIFO access speed must be fast enough to handle N writes per packet cycle to account for N packet departures from the same row in parallel.

The BNR/Harvard switch uses a link-level FC protocol based on absolute credits, which requires $N \cdot \log(MV)$ bits of credit information to be transmitted per packet cycle where V is the total number of connections shared in a single link [58]. The ATLAS and DEC ANS switches [91] reduce the total amount of credit information to be transmitted per packet cycle to $N \cdot \log(MV)$ bits. A reception scheduler is used to reduce the credit feedback rate to a one-per-cell transfer cycle without a noticeable performance reduction in [33].

Several studies address the issue of CP buffer size. A reduction of CP buffer for CICQ switches with multiple-priority traffic is investigated in [63], where CP buffer size is reduced from N^2PT_{RTT} to $N^2T_{RTT} + P - 1$ where T_{RTT} is RTT delay in cell time. A two-lane buffered crossbar design was proposed to handle more than two levels of priority traffic using only two queues per CP [19] and [18]. It was observed that the CICQ switch with a CP buffer size that can hold 60% of back-to-back cells in transit between the line card and the CP buffer has an acceptable performance [33]. Simulation evaluations of the effect of RTT and CP buffer size for variable-length packets were carried out [53]. As link data rates and internal cable lengths increase, the minimum number of feedback credits needed to maintain work conservation of the switch increases. Consequently, the switch fabric will no longer be able to implement CP

buffers sufficient to maintain work conservation of the switch. CICQ switches that scale independently of the growth of the RTT value are needed.

7.2 Distributed rate controlled CICQ switches

A new distributed rate-controlled CICQ switch is proposed to reduce the CP buffer size of the CICQ switch. The goal is to implement a switch that is fully scalable and independent of the RTT value. Figure 7.2 shows an overview of the distributed rate-controlled CICQ switch architecture. The distributed rate-controlled CICQ switch consists of inter-connected line card cabinets and a crossbar cabinet. The interconnection of line card cabinets forms a ring where scheduling information is circulated among line cards. No communication is needed between the line card and crossbar for scheduling purposes, thereby reducing the I/O requirement at crossbar chips. The formal CICQ switch architecture uses credit-based flow control via CP buffer occupancy status. Thus, it requires a communication link between the crossbar fabric and line cards for the

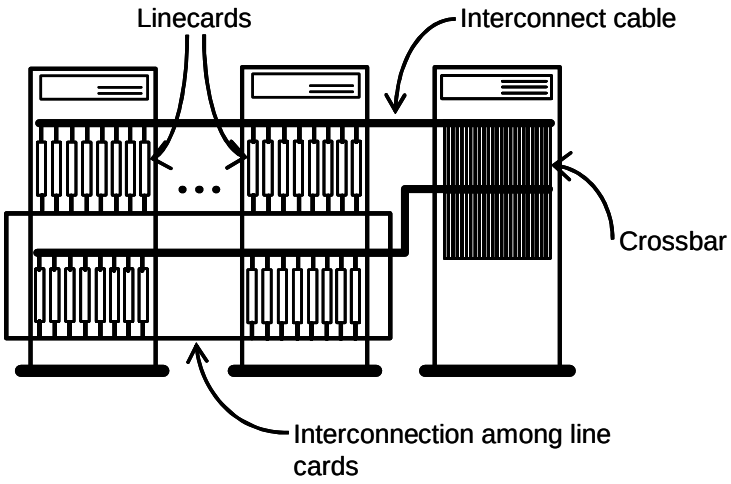

Figure 7.2 – Distributed rate controller (overview)

transferring of CP buffer occupancy status in addition to an interconnected link for transferring packet payload. This configuration is undesirable for a switch with a large number of ports, since crossbar chips have a limited number of pins for I/Os [21].

The distributed rate-controlled CICQ switch uses overlapped rate allocation and VOQ scheduling phases in a time cycle, called a *frame*, that consists of multiple cell scheduling cycles: the rate allocated in $frame_i$ is used by VOQ scheduling in $frame_{i+1}$ (see Figure 7.3). It allocates a rate to each VOQ so that over allocation of the CP buffer is prevented.

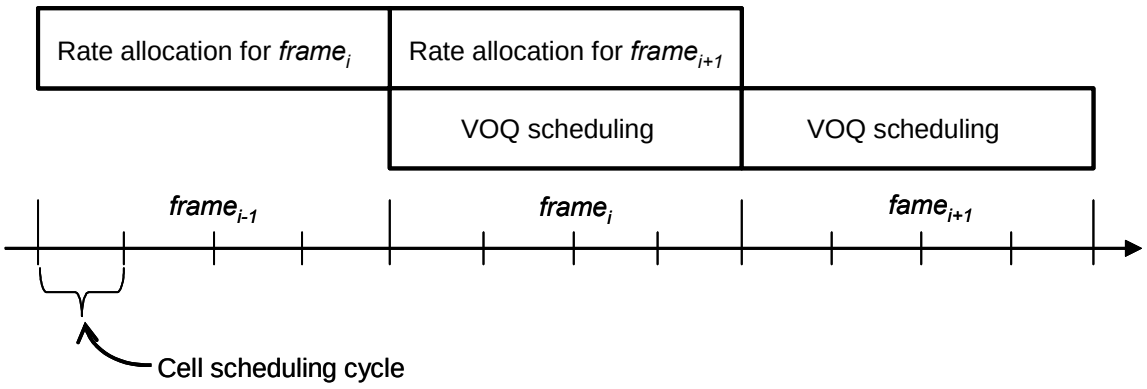

Figure 7.3 – Overlapped rate allocation and VOQ scheduling phases

Figure 7.4 shows the pseudocode for the rate allocation. Each input port first generates an N-bit vector, *VOQ_empty*[1..N], with 0 and 1 bits indicating non-empty VOQ and empty VOQ, respectively (step 1). Another vector to store the current VOQ occupancy status, *VOQ_current*[1..N], is also generated (step 1). The allocation of rates to VOQs is achieved by distributing the VOQ occupancy status of each input port through a ring of interconnected line cards (step 2). Each input port monitors the arrival of the VOQ occupancy status and keeps track of how may VOQs that share a common output

destination are non-empty; it does this via N counters, *cnt*[1…*N*] (one for each set of VOQs with a common output port destination) (step 3). Once the VOQ occupancy statuses are all distributed, each input port updates a set of N threshold values, *threshold*[1...*N*], and generates a masking vector, *VOQ_msk[1...N]* (step 5).

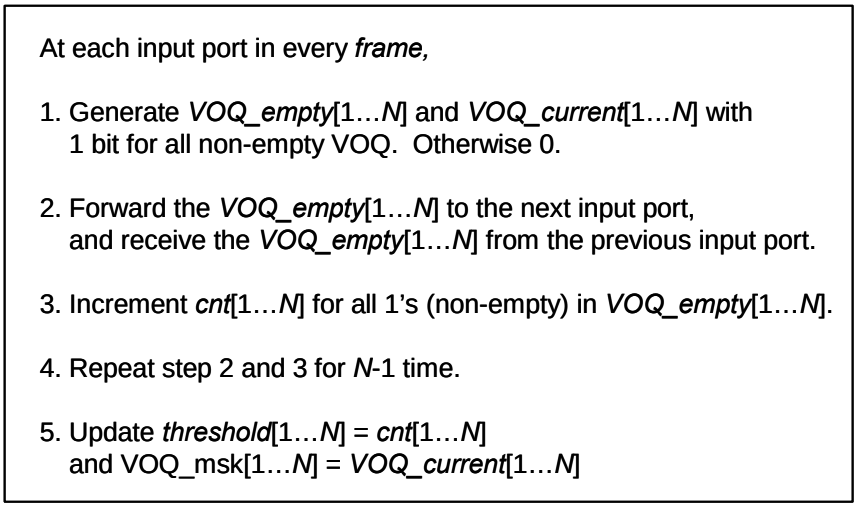

Figure 7.4 – Rate control

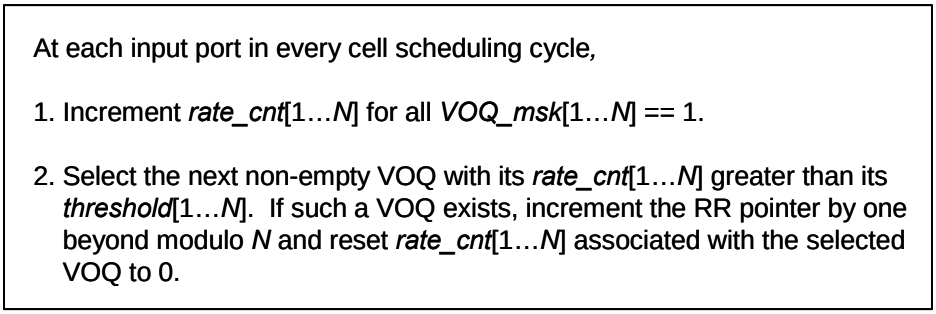

Figure 7.5 – VOQ scheduling

In the VOQ scheduling phase (Figure 7.5), rate counters, *rate_cnt*[1…*N*] are incremented by 1 in every cell scheduling cycle. A cell may be scheduled once the HOL cell has waited for a period of time equal to or greater than *threshold*[1...*N*].

7.3 Properties of distributed rate controlled CICQ switches

In the distributed rate-controlled CICQ switch, the rate may be overallocated to a VOQ with a small number of cells, resulting in under utilization of the link between the line card and crossbar. Figure 7.6 illustrates a simple scenario of a non-work conserving system, where marked slots represent the transfer of cells. For $N = 3$, *frame* = 6 cell scheduling times, and assuming VOQ_{11} and VOQ_{21} are saturated, suppose a cell arrives to an empty VOQ_{31} during $frame_{i-1}$; this cell will not be "visible" to the VOQ scheduler in $frame_{i-1,}$ or $frame_i$. During $frame_i$, cells from VOQ_{11} and VOQ_{21} depart with a total rate of 6 cells/frame. During the rate allocation phase in $frame_i$, a rate of 2 cells/frame is

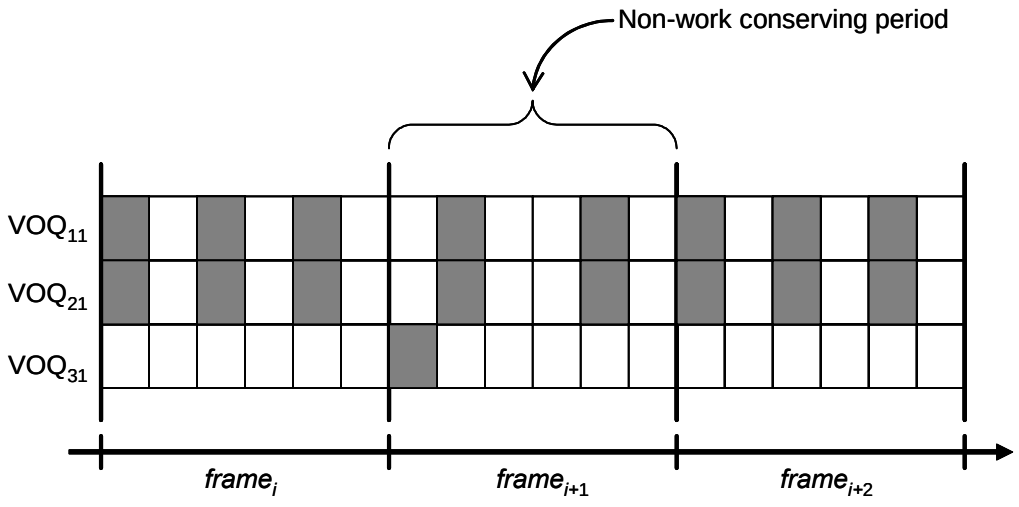

Figure 7.6 – Underallocation of rate

allocated to all three VOQs; however, only one cell can be transferred from VOQ_{31} during the VOQ scheduling phase in $frame_{i+1}$. The condition could repeat indefinitely, resulting in an overflow of cells at VOQ_{11} and VOQ_{21}.

To make the distributed rate-controlled CICQ switch work-conserving, a comparison of VOQ length with threshold is introduced. In step 1 of the rate allocation phase (Figure 7.4), *VOQ_empty*[1…*N*] is set to 1 only if the length of the associated VOQ exceeds a predefined threshold value. A threshold value greater or equal to *frame* is sufficient to make the switch work conserving. To avoid a starvation of VOQ, a timer is set to drain cells at a VOQ less than the threshold value. A similar approach is taken in [16].

It can be shown that the switch size, *N*, is the bounding factor of the CP buffer size for the distributed rate-controlled CICQ switch.

Lemma. Total number of cells existing in a set of CP buffers that have a common output destination never exceeds N for any RTT value.

Proof. The maximum number of cell arrivals to a set of CP buffers with a common output destination in any *N* cell transfer interval is *N* (by the definition of the distributed rate-controlled CICQ switch). As many as *N* cells can be transferred from a set of CP buffers sharing a common output destination, to an output link, as long as a cell exists in any one of the CP buffers (by definition of RR arbitration). Thus, the total number of cells buffered in a set of CP buffers sharing a common output destination is at most *N*.

Theorem. A CP buffer size equivalent to N cells is sufficient for implementing an internally loss-less distributed rate-controlled CICQ switch for any RTT *value.*

Proof. Lemma directly implies that any CP buffer will never exceed *N* cells at any given time.

The maximum queue length of N in a CP buffer occurs in the following scenario: Suppose all VOQs with common outputs, say VOQ_{i0} for $\forall i\, 1 \le i \le N$, have a queued cell(s). In this scenario, it is possible for all HOL cells at the VOQs to be dequeued at the same cell scheduling cycle. It is also possible for this event to occur at the last cell scheduling cycle of $frame_k$. In $frame_{k+1,}$ a rate allocation phase in $frame_k$ may be such that the set of VOQs with the common output port 0 may allocate the entire rate to a single VOQ, say VOQ_{00}. If no rate is allocated for all VOQ_{0j} $\forall i\, 1 \le i \le N$, a CP_{00} will receive N cells from $VOQ_{00,}$ while the CP arbiter for output port 0 transfers the N previously queued cells, one cell from each CP_{i0} $\forall i\, 1 \le i \le N$.

7.4 Evaluation of the distributed rate controlled CICQ switches

This section evaluates the performance of the distributed rate controlled CICQ switch. Simulation models were developed for the CICQ switch and the distributed rate controlled CICQ switch with a CP buffer size = N x 64-byte cells. For all experiments, the response variable is switching delay. Control variables are N, RTT, *frame*, and offered load.

7.4.1 Traffic models

Both Bernoulli and IBP traffic are used to evaluate the performance of the distributed rate-controlled CICQ switch.

1) *Bernoulli traffic:* See Chapter 3.2.1 for the description.

2) *IBP traffic:* See Chapter 3.2.1 for the description.

7.4.2 Simulation experiments

Four simulation experiments are designed to evaluate the performance of the CICQ switch with the distributed rate controller. Output port destination configurations based on high-degree balanced, low-degree balanced, and low-degree unbalanced probability density functions [32] are examined using Bernoulli and IBP. These experiments evaluate how packet switches perform with different output port destination configurations. The same values are used for RTT delay and *frame* that is varied from 16 to 256 cell times for all experiments unless noted otherwise. For all experiments, simulation is terminated after 10 million cell times.

1) *High-degree balanced experiments:* Both Bernoulli and IBP arrival of cells are used, and each of the 16 input ports chooses an output port with a uniform distribution over the 16 output ports. All input ports and output ports have an identical offered load ranging from 80-99% and 60%-90%, for Bernoulli arrival of cells and IBP arrival of cells, respectively.
2) *Low-degree balanced experiments:* Both Bernoulli and IBP arrival of cells are used, and each of the 16 input ports chooses an output port with a uniform distribution over k output ports (where $k < 16$). All input ports and output ports have an identical offered load; k random permutations are used to assign input and output pairs; k is set to 4, and the offered load ranged from 80-99% and 60% to 90%, for Bernoulli arrival of cells and IBP arrival of cells, respectively.
3) *Low-degree unbalanced experiments:* Both Bernoulli and IBP arrival of cells are used, and half of the input ports chooses an output port with a uniform distribution over k ports; the other half chooses an output port with a uniform

distribution over $2k$ output ports. Each flow rate is identical. If half the input ports are loaded at λ., the other half of the input ports are loaded at 2λ. Each output port receives either k or $2k$ flows. $2k$ random permutations are used to assign input and output pairs; k is set to 4 and the offered load ranged from 80-99% and 60% to 90%, for Bernoulli arrival of cells and IBP arrival of cells, respectively.

7.4.3 Experiment results

Figure 7.7 shows the results for the high-degree balanced experiment with Bernoulli arrival of a cell. The distributed rate controlled CICQ switch has a higher delay than the CICQ switch for all RTT and offered loads being measured. The credit-based CICQ

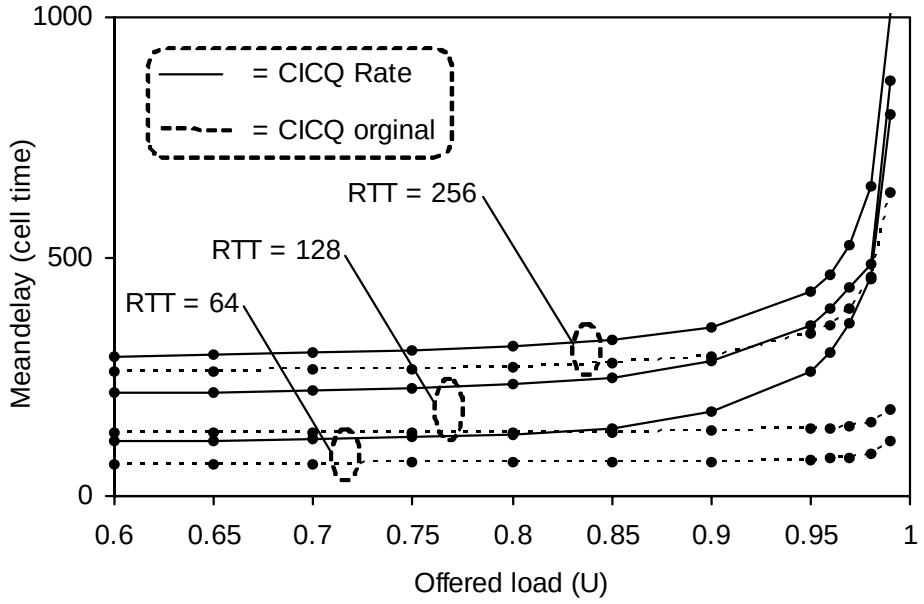

Figure 7.7 – Results for high-degree balanced (Bernoulli) experiment

switch performs well in this experiment because traffic is not bursty and is uniformly distributed to each output port. These two conditions help prevent saturations of the limited CP buffer. RTT is the main element of the switching delay for both switch architectures.

This is not the case in the results for the high-degree balanced experiment with IBP arrival of cells (Figure 7.8). Although traffic is uniformly distributed to all 16 ports, the IBP arrival of cells creates bursty arrival. With an RTT of 256, the CICQ switch has a higher delay than the distributed rate-controlled CICQ switch at a high offered load. RTT no longer dominates the overall switching delay for both switch architectures measured.

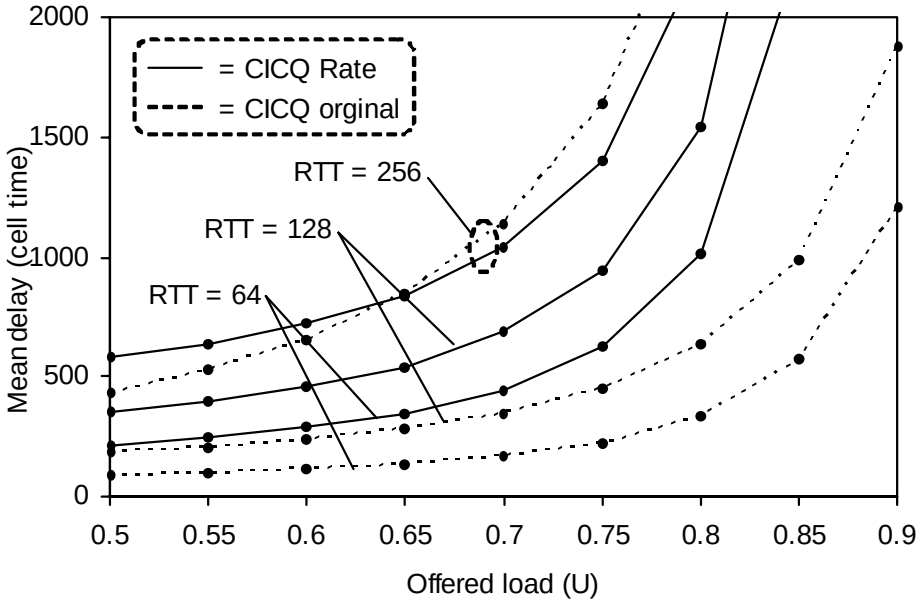

Figure 7.8 – Results for high-degree balanced (IBP) experiment

Figure 7.9 shows the results for the low-degree balanced experiment with a Bernoulli arrival of cells. The mean delay of the distributed rate-controlled CICQ switch is higher than that of the CICQ switch with RTT = 64; however, the CICQ switch with RTT = 128 and 256 are unstable for all offered loads measured. This is expected. Twenty-five percent of the aggregated traffic to any output port comes from a single input port, since each input port selects 4 output ports in this experimental setting. For RTT = 128, at least 32 (128 cells / 4 ports) cells per CP are needed to achieve stability for a theoretical 100% offered load with four output port destinations used in the experiment. The CICQ switch used in the experiment has limited buffering inside the crossbar sufficient to hold 16 cells per CP. *This buffer size is not sufficient to achieve stability, even with only a 50% offered load in this configuration.* The same conclusion is reached concerning the results for the low-degree balanced experiment with IBP arrival of cell (Figure 10).

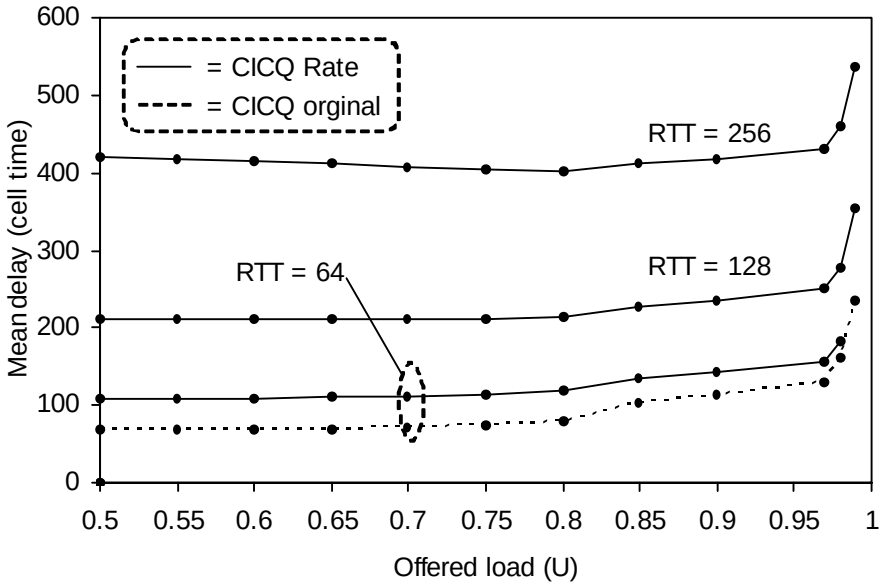

Figure 7.9 – Results for low-degree balanced (Bernoulli) experiment

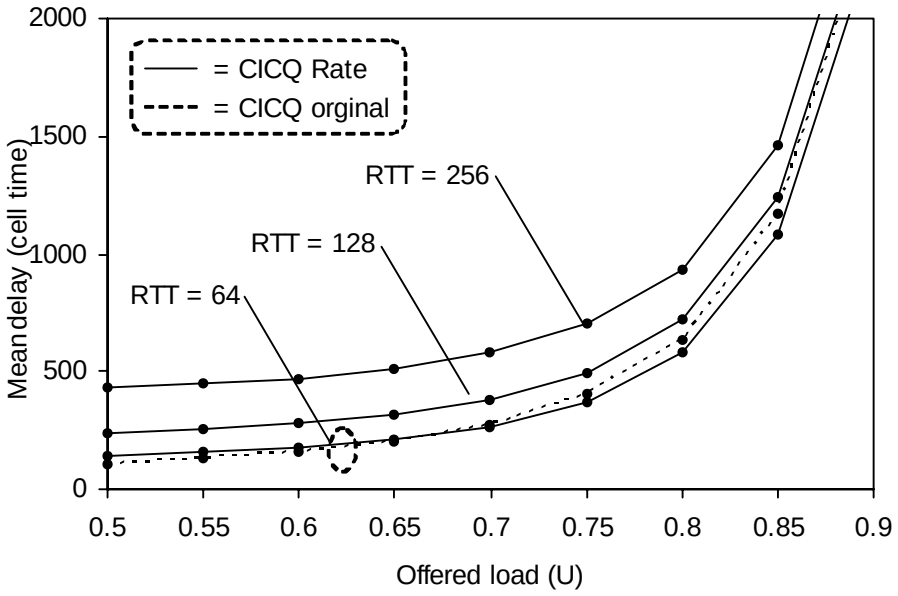

Figure 7.10 – Results for low-degree balanced (IBP) experiment

Figures 7.11 and 7.12 show the results for the low-degree unbalanced experiment with Bernoulli and IBP arrival of cells, respectively. These results indicate that an unbalanced output port configuration further increases the mean switching delay for both switch architectures. In particular, the mean delay of the distributed rate-controlled CICQ switch for the IBP arrival of cells is significantly larger than that measured in the low-degree balanced experiment.

In summary, the distributed rate control switch exhibits greater delay than the credit-based (original) CICQ switch. However, the distributed rate control switch is stable in cases (such as the low-degree balanced and unbalanced traffic, which is typical of real traffic) where the credit-based switch is unstable. This ability to maintain stability is significant and demonstrates that internal rate control of input ports is a good solution for scaling the CICQ switch to multi-cabinet implementations.

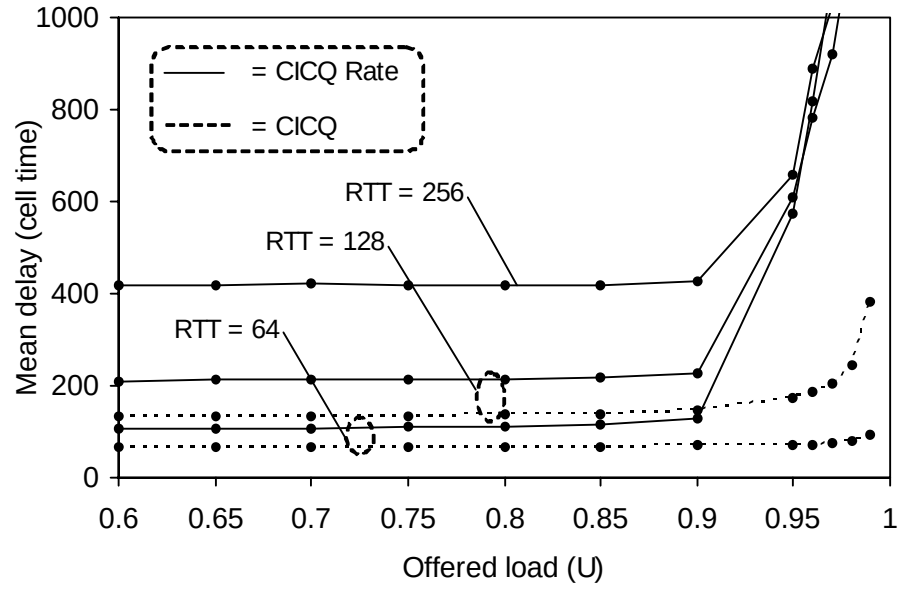

Figure 7.11 – Results for low-degree unbalanced (Bernoulli) experiment

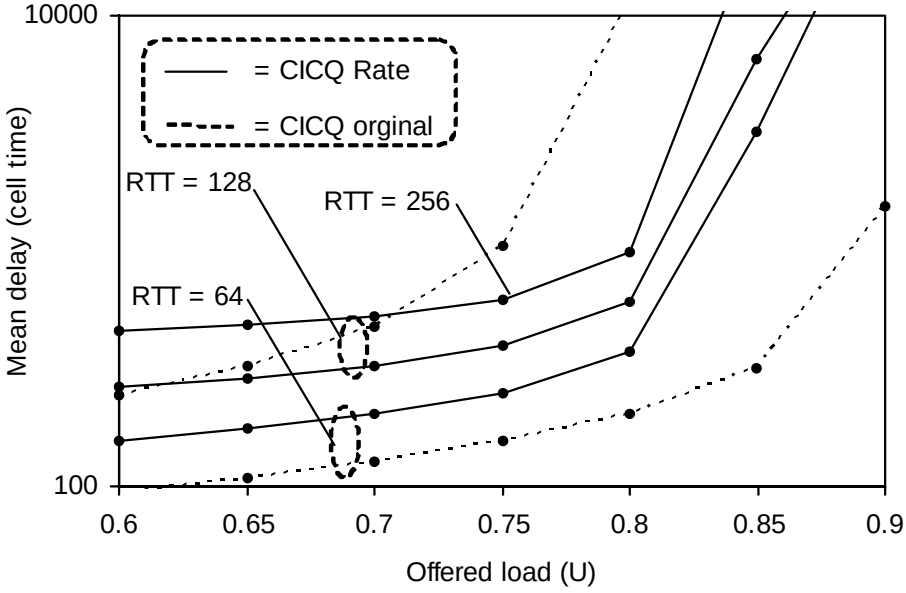

Figure 7.12 – Results for low-degree unbalanced (IBP) experiment

Chapter 8: Summary and Directions for Future Research

The combined input and cross point queued (CICQ) switch architecture has buffering at each input port and crossbar cross point. This architecture is now feasible due to an increase in VLSI density. In this dissertation, the evolution of IQ switches to combined input and crossbar queued (CICQ) switches has been studied. This dissertation has shown that CICQ switches can have simpler and faster schedulers, result in a lower delay, and scale better than IQ switches.

Key switch architectures, including output queued (OQ) and input queued (IQ), were modeled using discrete event simulation techniques. The CICQ switch was modeled and a performance evaluation comparing the CICQ switch to existing switch architectures was completed. It was shown that the CICQ switch has a lower delay at high offered loads than the IQ switch.

IQ switches, including the CICQ switch, are unstable for an unbalanced (schedulable) traffic load to two ports of a switch. This unstable region occurs when asymmetric arrivals occur at any two input ports in a switch. The burst stabilization protocol was proposed as a solution to this instability region. The new protocol uses a queue length threshold in the switch VOQ buffers. It aggressively serves resources with a queue length above a threshold value and prevents queues from growing without bound. The burst stabilization protocol is shown to provide stability for both IQ and CICQ switches.

The significance of the burst stabilization protocol is that a costly internal speed-up of the switch is not needed, while all existing methods to achieve stability require speed-up.

Variable length packets dominate network traffic (e.g., IP packets in Ethernet frames). Switching variable length packets in IQ switches rationally requires an internal switch speed-up, and the segmentation of packets into cells. A method of cell-merging, where header bytes of the arriving packet are merged with trailer bytes from the previous packet, is proposed and evaluated. This cell merging method reduces the required speed-up; no changes to switch-matrix scheduling algorithms are needed. Simulation with a packet trace shows a reduction in the needed speed-up for an iSLIP scheduled input buffered switch.

Native switching of variable length packets in CICQ switches results in unfairness between ports. A block transfer mechanism is proposed to resolve the unfairness caused by the variability of packet lengths in CICQ switches. The block transfer mechanism transfers up to a predefined number of bytes of packet data from a selected VOQ. The CICQ switch with the block transfer mechanism can handle switching of variable-length packets better than existing IQ switches, which use speed-up for various types of traffic, including traced packets-based traffic.

The feasibility of the CICQ switch architecture for 24 ports and a 10-Gbps link data rate is demonstrated with an FPGA-based design. Scalability of the CICQ switch is dependent on the speed of round robin (RR) polling and the delay of internal switch feedback. Two new RR arbiters were proposed, modeled, and evaluated. A priority encoder based RR poller that uses feedback masking was proposed. This design has a lower delay than any known design for an FPGA implementation. Second, an overlapped

RR (ORR) arbiter design that fully overlaps RR polling and scheduling was proposed. The ORR arbiter is fully scalable, independent of the size of the inputs, and has been proven to be work-conserving.

A distributed rated control of input ports was investigated to enable the CICQ switch to scale to multi-cabinet implementation while bounding the size of the CP buffers. It was shown that switch stability could be achieved independent of RTT (between line cards and crossbar) for all types of traffic. The original credit-based CICQ switch, was unable for realistic low-degree balanced and unbalanced traffic.

8.1 Specific contributions of this research

This research has addressed new methods for improving the performance, stability, and scalability of the CICQ switch. The five key contributions of the research presented in this dissertation are as follows:

1) The performance of CICQ switches was evaluated.
2) A new method of achieving switch stability without an internal speed-up for unbalanced (schedulable) traffic was proposed and investigated.
3) Reduced complexity and reduced speed-up in the scheduling of variable length packets was achieved for VOQ IQ switches, and the unfairness caused by the variability of packet lengths in CICQ switches was resolved using a block transfer mechanism.
4) The feasibility of the implementation of a CICQ switch was investigated: the hardware design of a 10-Gps 24 port CICQ switch was performed using FPGA technology, and faster and scalable RR arbiters were designed.

5) A future scalable distributed CICQ switch scheduler was proposed and investigated.

In summary, the evaluations performed in this research show that CICQ switches have a lower delay, are simpler to implement, and scale to an ever-increasing link data rate better than existing VOQ IQ switches. Because it is able to efficiently support variable-length packets, the CICQ switch architecture could be the next generation of a single-stage crossbar switch architecture that will enable the future growth of the Internet.

8.2 Directions for future research

This dissertation has addressed the stability, variable-length packet handling capabilities, and scalability of the CICQ switch. Further scalability can be achieved if the CP buffer size is further reduced. One possibility for achieving this requirement is by combining a small amount of the dedicated CP buffer, and a reasonable amount of the expensive shared memory internal to the switch fabric. Shared memory, if properly configured, can be flexibly allocated to prevent the temporal overload of any CP buffer. Implementation of shared buffers per column or per row of a crossbar is worth exploring in the future.

Only recently, QoS work was addressed in the context of CICQ switch architectures. In [83], the cell arrival time at the input port is forwarded to the CP buffer along with the cell, and is used at the CP arbiter. The amount of bandwidth needed to allocate to the flow in the switch to guarantee delay was determined in [78]. WFQ for a CICQ switch was implemented and its fairness properties were studied in [17]. It was shown that a CICQ switch with speed-up of two can emulate an OQ switch [64]; and 3x speed-up is

sufficient to guarantee a delay in the CICQ switch [21]. Further investigation is needed into QoS in the CICQ switch architecture.

References

[1] F. Abel, C. Minkenberg, R. Luijten, M. Gusat, and I. Hiadis, "A Four-Terabit Single-Stage packet Switch with Large Round-Trip Time Support," *IBM Research Report RZ 3430 (#93609)*, July 2002.

[2] Alcatel, "Alcatel 7670 Routing Switch Platform." URL: *http://www.alcatel.com/products/productsummary.jhtml?repositoryID=/x/opgproduct/Alcatel_7670_RSP.jhtml*.

[3] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-Speed Switch Scheduling for Local-Area Networks," *ACM Transactions on Computer Systems* 11, no. 4 (November 1993): 319-352.

[4] AVNET Inc., "Designing High-Performance Systems with Virtex-II Pro FPGA Devices," Presentation given on September 10, 2002 for the Florida Suncoast IEEE chapter.

[5] Avici Systems, "The Avici TSR: The World First Scalable Router." URL: *http://www.avici.com/documentation /datasheets/Avici_TSR.pdf*.

[6] R. Bakka and M. Dieudonne, "Switching Circuit for Digital Packet Switching Network," *United States Patent 4,314,367*, February 1982.

[7] Bell Labs, "Network Equipment Building Systems Statndards," URL: *http://www.nebs-faq.com.*

[8] G. Bloch, S. Greiner, H. de Meer, and S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications.* New York: John Wiley & Sons, Inc., 1998.

[9] E. Brockmeyer, H. Halstrom and A. Jensen, "The Life and Works of A. K. Erlang," Copenhagen: The Copenhagen Telephone Company, 1948.

[10] W. Bux, "Local-Area Subnetworks: A Performance Comparison," *IEEE Transactions on Communications* 29, no. 10 (October 1981): 1465-1473.

[11] C. Chang, W. Chen, and H. Juang, "On Service Guarantees for Input Buffered Crossbar Switches: A Capacity Decomposition Approach by Birkhoff and von Neumann," *Proceedings of IEEE IWQoS*, 1999, pp. 79-86.

[12] J. Chao and J. Park, "Centralized Contention Resolution Schemes for a Large-Capacity Optical ATM Switch," *Proceedings of IEEE ATM Workshop,* May 1999, pp. 79-86.

[13] J. Chao, "Saturn: A Terabit Packet Switch Using Dual Round-Robin," *IEEE Communication Magazine* 38, no. 12, (December 2000): 78-84.

[14] J. Chao, C. Lam, and X. Guo, "A Fast Arbitration Scheme for Terabit Packet Switches," *Proceedings of IEEE GLOBECOM*, December 1999, pp. 1236-1243.

[15] K. Christensen, Home Page for Kenneth J. Christensen, 2000. URL: *http://www.csee.usf.edu/~christen*.

[16] K. Christensen, K. Yoshigoe, A. Roginsky, and N. Gunther, "Performance of Packet-to-Cell Segmentation Schemes in Input Buffered Packet Switches," *Proceedings of the IEEE ICC*, June 2004, pp.1097-1102.

[17] N. Chrysos and M. Katevenis, "Weighted Fairness in Buffered Crossbar Scheduling," *Proceedings of IEEE HPSR*, June 2003, pp. 17-22.

[18] N. Chrysos: "Design Issues of Variable-Packet-Size, Multiple-Priority Buffered Crossbars", *Technical Report FORTH-ICS/TR*-325, October 2003.

[19] N. Chrysos and M. Katevenis, "Multiple Priorities in a Two-Lane Buffered Crossbar," *Technical Report FORTH-ICS/TR-328*, November 2003.

[20] S. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching Output Queueing with a Combined Input/Output-Queued Switch," *IEEE Journal of Selected Areas in Communications* 17, no. 6 (June 1999): 1030-1039.

[21] S. Chuang, S. Iyer, and N. McKeown, "Practical Algorithms for Performance Guarantees in Buffered Crossbars," *Stanford HPNG Technical Report TR03-HPNG-061501*, 2003.

[22] Cisco Systems, "Data sheet: Cisco 12416 Internet Router." URL: *http://www.cisco.com/warp/public/ cc/pd/rt/12000/12416/prodlit/itro_ds.htm.*

[23] CNET, 2002. URL: *http://shopper.cnet.com/Cisco_12416___router/4014-3334_9-30108507.html*.

[24] S. Crocker, "Host Software," RFC 001, April 1969. URL: *http://www.funet.fi/index/FUNET/history/internet/en/rfc1.txt*.

[25] W. Cui, H. Ko, and S. An, "A Threshold Based Scheduling Algorithm for Input Queue Switch, " *Proceedings of ICIN*, February 2001, pp. 207-212.

[26] J. Dai and B. Prabhakar, "The Throughput of Data Switches with and without Speedup," *Proceedings of IEEE INFOCOM*, March 2000, pp. 556-564.

[27] J. Delgado-Frias and J. Nyathi, "A VLSI High-Performance Encoder with Priority Lookahead," *Proceedings of the 8th Great Lakes Symposium on VLSI,* February 1998, pp.59-64.

[28] A. Demers, S. Keshav, and S. Shenkar, "Analysis and Simulation of a Fair Queueing Algorithm," *Proceedings of SIGCOMM,* September 1989, pp. 1-12.

[29] Y. Doi and N. Yamanaka, "A High-Speed ATM Switch with input and Cross-Point Buffers," *IEICE Transactions on Communications* E76-B, no. 3 (March 1993): 310-314.

[30] G. Gilder, *Telecosm: The World After Bandwidth Abundance*, Touchstone, 2002.

[31] P. Goli and V. Kumar, "Performance of Crosspoint Buffered ATM Switch Fabric," *Proceedings of IEEE INFOCOM*, May 1992, pp. 426-435.

[32] M. Goudreau, S. Kolliopoulos, and S. Rao, "Scheduling Algorithms for Input-Queued Switches: Randomized Techniques and Experimental Evaluation," *Proceedings of IEEE INFOCOM*, March 2000, pp. 1634-1643.

[33] F. Gramsamer, M. Gusat, and R. Luijten, "Flow Control Scheduling," *Microprocessors and Microsystems* 27 (2003): 233-241.

[34] V. Guffens, G. Bastin, and H. Mounier, "Using Token Leaky Bucket with Feedback Control for Guaranteed Boundedness of Buffer Queue," *preprint submitted to 2003 European Control Conference*, October 2002.

[35] N. Gunther and J. Shaw, "Path Integral Evaluation of ALOHA Network Transients," *Information Processing Letters* 33, no. 6 (1990): 289-295.

[36] N. Gunther, K. Christensen, and K. Yoshigoe, "Characterization of the Burst Stabilization Protocol for the RR/RR CICQ Switch," *Proceedings of the IEEE Conference on Local Computer Networks*, October 2003, pp. 260-269.

[37] A. Gupta and N. Georganas, "Analysis of Packet Switch with Input and Output Buffers and Speed Constraints," *Proceedings of IEEE INFOCOM*, April 1991, pp.694-700.

[38] P. Gupta and N. McKeown, "Design and Implementation of a Fast Crossbar Scheduler," *IEEE Micro* 19, no. 1 (January/February 1999): 20-28.

[39] A. Gupta, L. Barbosa, and N. Georganas, "16x16 Limited Intermediate Buffer Switch Module for ATM Networks," *Proceedings of IEEE GLOBECOM*, December 1991, pp. 939-943.

[40] A. Gupta, L. Barbosa, and N. Georganas, "Limited Intermediate Buffer Switch Modules and Their Interconnection Networks for B-ISDN," *Proceedings of IEEE ICC*, June 1992, pp. 1646-1650.

[41] M. Han, D. Kwak, and B. Kim, "Desynchronized Input Buffered Switch with Buffered Crossbar," *IEICE Transactions on Communications* E86-B, no. 7 (July 2003): 2216-2219.

[42] J. Hopcroft and R. Karp, "An $N^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs," *Society for Industrial and Applied Mathematics Journal for Computation* 2, (1973): 225-231.

[43] S. Huajin, G. Deyuan, Z. Shengbing, and W. Danghui, "Design Fast Round Robin Scheduler in FPGA," *Proceedings of IEEE Communications, Circuits and Systems and West Sino Expositions*," 2002, pp. 1257-1261.

[44] IEEE "IEEE Standards 802.3," 1983.

[45] Intel Corp., "Intel Accelerates 10-Gigabit Communications in Enterprise Data Centers with New XPAK Optical Transceiver," *Press Release*. URL: *http://www.intel.com/pressroom/archive/releases/20020827net.htm*.

[46] T. Javadi, R. Magill, and T. Hrabik, "A High-Throughput Scheduling Algorithm for a Buffered Crossbar Switch Fabric," *Proceedings of IEEE ICC*, June 2001, pp. 1581-1591.

[47] G. Jeong, J. Lee, and B. Lee, "An Advanced Input-Queued ATM Switch with a Pipelined Approach to Arbitration," *Proceedings of IEEE ICC*, April 2002, pp. 2416-2420.

[48] Y. Jiang and M. Hamdi, "A 2-stage Matching Scheduler for a VOQ Packet Switch Architecture," *Proceedings of IEEE ICC,* April-May 2002, pp. 26-33.

[49] Juniper Networks, “The Essential Core: Juniper Networks T640 Internet Routing Node with Matrix Technology.” URL: *http://www.juniper.net/solutions/literature/solutionbriefs/351006.pdf*.

[50] M. Karol and M. Hluchyj, “Queueing in High-performance Packet-Switching,” *IEEE Journal on Selected Areas in Communications* 6, (December 1988): 1587-1597.

[51] M. Karol, M. Hluchyj, and S. Morgan, “Input versus Output Queueing on a Space Division Packet Switch,” *IEEE Transactions on Communications* 35, no. 12 (December 1987): 1347-1356.

[52] K. Katevenis, D. Serpanos, and P. Vatsolaki, “ATLAS I: A General-purpose, Single-chip ATM Switch with Credit-Based Flow Control,” *Proceedings of IEEE Hot Interconnects Symposium*, August 1996, pp. 63-73.

[53] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos, “Variable Packet Size Buffered Crossbar (CICQ) switches,” *Proceedings of IEEE ICC*, June 2004, pp.1090-1096.

[54] Y. Kato, T. Shimoe, K. Hajikano, and K. Murakami, “Experimental Broadband ATM Switching System,” *Proceedings of IEEE GLOBECOM*, December 1988, pp.1288-1292.

[55] H. Kim, J. Son, and K. Kim, “A Packet-Based Scheduling Algorithm for High-Speed Switches,” *International Conference on Electrical and Electronic Technology,* August 2001, pp. 117-121.

[56] P. Krishna, N. Patel, A. Charny and R. Simcoe, “On the Speedup Required for Work-Conserving Crossbar Switches,” *Proceedings of IEEE/IFIP IWQoS*, May 1998, pp. 225-234.

[57] H. Kung and A. Chapmann, “The FCVC (Flow-Controlled Virtual Channels) Proposal for ATM Networks: A Summary,” *Proceedings of International Conference on Network Protocols*, October 1993, pp. 116-127.

[58] H. Kung, T. Blackwell, and A. Chapmann, "Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Allocation, and Statistical Multiplexing, *Proceedings of ACM SIGCOMM* , 1994, pp. 101-104.

[59] G. Leonidas and W. Szpankowski, "Stability of Token Passing Rings," *Queueing Systems: Theory and Applications*, Vol. 11, pp. 7-33, February 1992.

[60] S. Li and M. Lee, "A Study of Traffic Imbalances in a Fast Packet Switch," *Proceeding of IEEE INFOCOM*, April 1989, pp. 538-547.

[61] R. LoMaire, and D. Serpanos, "Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues," *IEEE/ACM Transactions on Networking* 2, no. 5 (October 1994): 471-482.

[62] G. Lornaros, D. Pnevmatiakos, P.Vatsolaki, G. Kalokerinos, C. Xanthaki, D. Mavroidis, D. Serpanos, and M. Katevenis, "Implementation of ATLAS I: A Single-Chip ATM Switch with Backpressure," *Proceedings of IEEE Hot Interconnects Symposium*, August 1998, pp. 85-96.

[63] R. Luijten, C. Minkenberg, and M. Gusat, "Reducing Memory Size in Buffered Crossbars with Large Internal Flow Control Latency," *Proceedings of IEEE GLOBECOM* 7 (December 2003): pp. 3683-3687.

[64] R. Magill C. Rohrs, R. Stevenson, "Output Queued Switch Emulation by Fabrics with Limited Memory," *IEEE Journal of Selected Areas in Communications* 21, no. 4, (May 2003): 606-615.

[65] M. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet Scheduling in Input-Queued Cell-Based Switches," *Proceedings of IEEE INFOCOM*, 2001, pp. 1085-1094.

[66] M. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet-Mode Scheduling in Input-Queued Cell-Based Switches," *IEEE/ACM Transactions on Networking* 10, no. 5, (October 2002): 666-678.

[67] M. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet Scheduling in Input-Queued Cell-Based Switches," *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, 2001, pp. 1085-1094.

[68] M. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Scheduling in Input-Queued Cell-Based Packet Switches," *Proceedings of IEEE GLOBECOM*, December 1999, pp. 1227-1235.

[69] N. McKeown and T. Anderson, "A Quantitative Comparison of Iterative Scheduling Algorithms for Input-Queued Switches," *Computer Networks and ISDN Systems* 30, no. 24, (December 1998): 2309-2326.

[70] N. McKeown, "Fast Switched Backplane for a Gigabit Switched Router," Cisco Systems white paper, URL: *http://www.cisco.com/warp/public/cc/cisco/mkt/core/12000/tech/fasts_ws.pdf.*

[71] N. McKeown, "Scheduling algorithms for Input-QueuedSwitches," Ph.D. Thesis, University of California at Berkley, 1995.

[72] N. McKeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Transactions on Networking* 7, no. 2 (April 1999): 188-201.

[73] N. McKeown, B. Prabhakar, and M. Zhu, "Matching Output Queueing with Combined Input and Output Queueing," *Proceedings of the 35th Allerton Conference on Communication, Control and Computing*, September 1997, pp.595-603.

[74] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "The Tiny Tera: A packet Switch Core," *IEEE Micro Magazine* (Jan.-Feb. 1997): 26-33.

[75] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *Proceedings of IEEE INFOCOM*, March 1996, pp. 296-302.

[76] A. Mekkittikul and N. McKeown, "A Starvation-free Algorithm for Achieving 100% Throughput in an Input-Queued Switch," *Proceedings of IEEE ICCCN*, October 1996, pp. 226-231.

[77] A. Mekkittikul and N. McKeown, "Achieving 100% Throughput in an Input-Queued Switch," *IEEE Transactions on Communications* 47, no. 8 (August 1999): 1260-1267.

[78] L. Mhamdi and M. Hamdi, "MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches," *IEEE Communications Letters*, pp. 451-453, Vol. 7, (9), September 2003.

[79] L. Mhamdi and M. Hamdi, "Practical Scheduling Algorithms for High-Performance Packet Switches, *Proceedings of IEEE ICC*, May 2003, pp. 1659-1663.

[80] C. Minkenberg, R. Luijte, F. Abel, W. Denzel, and M. Gusat, "Current Issues in Packet Switch Design," *Proceedings of ACM SIGCOMM*, January 2003, p.119-124.

[81] S. Moon and D. Sung, "High-Performance Variable-Length Packet Scheduling Algorithm for IP Traffic," *Proceedings of IEEE GLOBECOM*, November 2001, pp. 2666-2670.

[82] A. Motoki, S. Kamiya, R. Ikematsu, and H. Ozaki, "Group-Pipeline Scheduler for Input-Buffer Switch," *Proceedings of IEEE International Conference on ATM and High Speed Intelligent Internet Symposium,* April 2001, pp. 158-162.

[83] S. Motoyama and M. Arantes, "IP Switch with Distributed Scheduling," *IEE Electronics Letters*, pp. 392-393, April 2002.

[84] M. Nabeshima, "Performance Evaluation of a Combined Input- and Crosspoint-Queued Switch," *IEICE Transactions on Communications* E83-B, no. 3 (March 2000): 737-741.

[85] S. Nojima, E. Tsutio, H. Fukuda, and M. Hashimoto, "Integrated Services Packet Network Using Bus Matrix Switch," *IEEE Journal of Selected Areas in Communications* 5, no. 8, (October 1987): 1284-1292.

[86] G. Nong, M. Hamdi, and K. Letaief, "Efficient Scheduling of Variable-Length IP Packets on High-Speed Switches," *Proceedings of IEEE GLOBECOM*, December 1999, pp. 1407-1411.

[87] NuHorizons Electronic Corp., "XC2VP Virtex-II Pro FPGA." URL: *http://www.nuhorizons.com/products/NewProducts/POQ13/xilinx.html*.

[88] Y. Oie, M. Murata, K. Kubota and H. Miyahara, "Effect of Speedup in Nonblocking Packet Switch," *Proceedings of IEEE ICC*, June 1989, pp. 410-414.

[89] E. Oki, R. Rojas-Cessa, and J. Chao, "PCRRD: A Pipeline-Based Concurrent Round-Robin Dispatching Scheme for Clos Network Switches," *Proceedings of IEEE ICC*, April 2002, pp. 2121-2125.

[90] E. Oki, R. Rojas-Cessa, and J. Chao, "A Pipeline-Based Approach for Maximal-Sized Matching Scheduling in Input-Buffered Switches," *IEEE Communications Letters*, pp. 263-265, Vol. 5, (6), June 2001.

[91] C. Ozveren, R. Simcoe, and G. Varghese, "Reliable Efficient Hop-by-Hop Flow Control," *IEEE Journal of Selected Areas in Communications* 13, no. 4 (1995): 642-650.

[92] A. Parekh and R. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Service Networks: The Single-node Case," *IEEE/ACM Transaction on Networking* 1, (1993): 344-357.

[93] I. Radusinovic and M. Pejanovic, "Impact of Scheduling Algorithms on Performances of Buffered Crossbar Switch Fabrics," *Proceedings of IEEE ICC*, April 2002, pp. 2416-2420.

[94] E. Rathgeb, T. Theimer, and M. Huber, "Buffering Concepts for ATM switching Networks," *Proceedings of IEEE GLOBECOM*, December 1988, pp.1277-1281.

[95] E. Re and R. Fantacci, "Performance Evaluation of Input and Output Queueing Techniques in ATM Switching Systems," *IEEE Transactions on Communications* 40, no. 10 (Oct. 1993): 1565-1575.

[96] L. Robert, "Beyond Moore's Law: Internet Growth Trends," *IEEE Computer Magazine* 33, issue 1 (January 2000): 117-119.

[97] R. Rojas-Cessa, E. Oki, Z. Jing, and H. Chao, "CIXB-1: Combined Input-One-Cell Crosspoint Buffered Switch," *Proceedings of IEEE Workshop on High Performance Switching and Routing*, May 2001, pp. 324-329.

[98] R. Rojas-Cessa, E. Oki, and H. J. Chao, "CIXOB-*k*: Combined Input-Crosspoint-Output Buffered Packet Switch," *Proceedings of IEEE GLOBECOM*, November 2001, pp. 2654-2660.

[99] R. Rojas-Cessa, "High-Performance Round-Robin Arbitration Schemes for Input-Crosspoint Buffered Switches, *Proceedings of IEEE Workshop on High Performance Switching and Routing*, April 2004, pp. 167-171.

[100] R. Schoenen and A. Dahlhoff, "Closed Loop Credit-Based Flow Control with Internal Backpressure in Input and Output Queued Switches," *Proceedings of IEEE Workshop on High Performance Switching and Routing*, 2000, pp. 195-203.

[101] H. Schwetman, "CSIM18 - The Simulation Engine," *Proceedings of the 1996 Winter Simulation Conference*, December 1996, pp. 517-521. URL: *http://www.mesquite.com*.

[102] D. Serpanos and P. Antoniadis, "FIRM: A Class of Distributed Scheduling Algorithms for High-speed ATM Switches with Multiple Input Queues," *Proceedings of IEEE INFOCOM*, March 2000, pp. 548-555.

[103] E. Shin, V. Mooney III, and G. Riley, "Round-Robin Arbiter Design and Generation*," Proceedings of the 15th International Symposium on System Synthesis*, October 2002, pp. 243-248.

[104] V. Singhal and R. Le, "High-Speed Buffered Crossbar Switch Design Using Virtex-EM Devices," March 14, 2000, http://www.xilinx.com/xapp/xapp240.pdf.

[105] A. Smiljanic, R. Fan, and G. Ramamurthy, "RRGS-Round-Robin Greedy Scheduling for Electronic/Optical Terabit Switches," *Proceedings of IEEE GLOBECOM*, December 1999, pp. 1244-1250.

[106] D. Stephens and H. Zhang, "Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture," *Proceedings of IEEE INFOCOM*, April 1998, pp. 282-290.

[107] D. Stiliadis and A. Varma, "Providing Bandwidth Guarantees in an Input-Buffered Crossbar Switch," *Proceedings of IEEE INFOCOM,* April 1995, pp. 960-968.

[108] I. Stoica and H. Zhang, "Exact Emulation of an Output Queueing Switch by a Combined Input Output Queueing Switch," *Proceedings of IEEE/IFIP International Workshop on Quality of Services*, May 1998, pp.218-224.

[109] H. Takagi, "Queueing Analysis of Polling Systems," *ACM Computing Surveys* 20, no. 1 (1988): 5-28.

[110] Y. Tamir and G. Frazier, "High Performance Multi-Queue Buffers for VLSI Communications Switches," *Proceedings of Computer Architecture*, June 1988, pp. 343-354.

[111] Y. Tamir and H. Chi, "Symmetric Crossbar Arbiters for VLSI Communication Switches," *IEEE Transactions on Parallel and Distributed System* 4, no. 1 (January 1993): 13-27.

[112] F. Tobagi, “Fast Packet Switch Architectures for Broadband Integrated Services Digital Networks,” *Proceedings of IEEE* 78, no.1 (January 1990): 133-167.

[113] F. Tobajas, R. Esper-Chain, V. Armas, J. Lopez, and R. Sarmiento,” Round-Trip Delay Effect on Iterative Request-Grant-Accept Scheduling Algorithms for Virtual Output Queued Switches,” *Proceedings of IEEE GLOBECOM* 2 (November 2002): 1889-1893.

[114] R. van der Mei, “Waiting-Time Distributions in Polling Systems with Simultaneous Batch Arrivals,” *Annals of Operations Research* 47, no. 113 (2002): 157-173.

[115] J. Walrand and P. Varaiya, *High Performance Communication Networks,* Morgan Kaufmann, 1996.

[116] Xilinx Inc., “Xilinx Vertex II Pro Platform FPGA Data Sheet DS083-1 (v2.1)”, September 3, 2002. URL: *http://www.Xilinx.com/publications/products/v2pro/ds_pdf/ds083.htm*.

[117] Xilinx Inc., “10-Gigabit Ethernet MAC with XGMII or XAUI v2.1 DS201 (v2.1),” June 24, 2002. URL: *http://www.xilinx..com/ipcenter/catalog/logicore/docs/ten_gig_eth_mac.pdf*.

[118] Xilinx Inc., “Xilinx WebPACK 4.2.” URL: *http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=ISE+WebPack*.

[119] K. Yoshigoe and K. Christensen, “A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar,” *2001 IEEE Workshop on High Performance Switching and Routing*, May 2001, pp. 271-275.

[120] K. Yoshigoe and K. Christensen, “An Evolution to Crossbar Switches with Virtual Output Queueing and Buffered Cross Points,” *IEEE Network* 17, no. 5 (September-October 2003): 48-56.

[121] K. Yoshigoe and K. Christensen, “A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar,” *Proceedings of IEEE HPSR*, May 2001, pp. 271-275.

[122] K. Yoshigoe, K. Christensen, and A. Jacob, “The RR/RR CICQ Switch: Hardware Design for 10-Gbps Link data rate,” *Proceedings of IEEE International Performance, Computing, and Communications Conference*, April 2003, pp. 481-485.

[123] K. Yoshigoe, K. Christensen, and A. Roginsky, “Design of A High-Speed Overlapped Round Robin (ORR) Arbiter,” *Proceedings of the IEEE Conference on Local Computer Networks*, October 2003, pp. 638-639.

[124] S. Zheng, M. Yang, J. Blanton, P. Golla, and D. Verchere, “A Simple and Fast Parallel Round-Robin Arbiter for High-Speed Switch Control and Scheduling,” *Proceedings of the 45th Midwest Symposium on Circuits and Systems*, August 2002, pp. 671-674.

[125] B. Zhou and M. Atiquzzaman, “Performance of ATM Switch Fabrics Using Cross-Point Buffers, *Proceedings of IEEE INFOCOM*, April 1995, pp. 16-23.

List of Publications

1) K. Christensen, K. Yoshigoe, A. Roginsky, and N. Gunther, "Performance of Packet-to-Cell Segmentation Schemes in Input Buffered Packet Switches," *Proceedings of the IEEE ICC*, June 2004, pp. 1097-1102.

2) K. Yoshigoe, K. Christensen, and A. Roginsky, "Performance Evaluation of New Scheduling Methods for the RR/RR CICQ Switches," submitted to the *Computer Communications*, July 2003.

3) N. Gunther, K. Christensen, and K. Yoshigoe, "Characterization of the Burst Stabilization Protocol for the RR/RR CICQ Switch," *Proceedings of the IEEE Conference on Local Computer Networks*, October 2003, pp. 260-269.

4) K. Yoshigoe, K. Christensen, and A. Roginsky, "Design of A High-Speed Overlapped Round Robin (ORR) Arbiter," *Proceedings of the IEEE Conference on Local Computer Networks*, October 2003, pp. 638-639.

5) K. Yoshigoe and K. Christensen, "An Evolution to Crossbar Switches with Virtual Output Queueing and Buffered Cross Points," *IEEE Network* 17, no. 5 (September-October 2003): 48-56.

6) K. Yoshigoe, K. Christensen, and A. Jacob, "The RR/RR CICQ Switch: Hardware Design for 10-Gbps Link Speed," *Proceedings of the IEEE 2003 International Performance, Computing, and Communications Conference*, April 2003, pp. 481-485.

7) K. Yoshigoe and K. Christensen, "RATE Control for Bandwidth Allocated Services in IEEE 802.3 Ethernet," *IEEE 26th Conference on Local Computer Networks*, November 2001, pp. 446-453.

8) K. Yoshigoe and K. Christensen, “A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar,” *Proceedings of IEEE Workshop on High Performance Switching and Routing*, May 2001, pp. 271-275.

About the Author

Kenji Yoshigoe received his Bachelor's degree in Computer Science from the University of South Florida. He worked as a research assistant in the field of computer networks. His Ph.D. work was funded by the National Science Foundation under Grant No. 9875177 through his advisor Dr. Kenneth J. Christensen. He is the author and co-author of 8 articles on the subject of performance evaluation of computer networks. His professional affiliations include IEEE, ACM, and ASEE.