Chapter 4: Analyzing CIOQ Routers with Localized Memories

Feb 2008, Bombay, India

Contents

List of Dependencies

- **Background:** The memory access time problem for routers is described in Chapter 1. Section 1.5.2 describes the use of load balancing techniques to alleviate memory access time problems for routers.

Additional Readings

- **Related Chapters:** The general load balancing technique used to analyze the Combined Input Output Queued (CIOQ) router is described in Section 2.3. The technique is also used to analyze other router architectures in Chapters 2, 3, 5, 6, and 10.

Table: *List of Symbols.*

B	Leaky Bucket Size
c, C	Cell
DT	Departure Time
k	Maximum Transfer Delay
N	Number of Ports of a Router
R	Line Rate
S	Speedup
T	Time Slot
t_f	First Free Time Index

Table: *List of Abbreviations.*

CIOQ	Combined Input Output Queued Router
FCFS	First Come First Serve (Same as FIFO)
i.i.d	Independently and Identically Distributed
LAN	Local Area Network
MWM	Maximum Weight Matching
OQ	Output Queued
PIFO	Push In First Out
SB	Single-buffered
SOHO	Small Office Home Office
QoS	Quality of Service
VOQ	Virtual Output Queue
WAN	Wide Area Network
WFQ	Weighted Fair Queueing

"Whoever does the proposing gets a better deal".

— Social Commentary by Harry Mairson[†]

4 Analyzing CIOQ Routers with Localized Memories

4.1 Introduction

In this chapter, we will analyze the combined input output queued (CIOQ) router. A CIOQ router (unlike the single-buffered (SB) routers that we introduced in Chapter 2), has two stages of buffering. In a CIOQ router, a packet is buffered *twice* - once when it arrives and once before it leaves. The first buffer is on the local line card it arrives on. The second buffer is on the line card that it departs from. Intuitively, the two buffers make the job easier for a central switch scheduler based on the following idea —

Idea. *"If the switch fabric is not ready to transfer packets as soon as they arrive, it can hold them in the input line card. Similarly, if the scheduler has the opportunity to send a packet to the output, it doesn't need to wait until the output line is free; the output line card will buffer it until the line is free".*

Our goal is to scale the performance of CIOQ routers and find the conditions under which they can give deterministic performance guarantees, as described in Section 1.4.

[†]Harry Mairson, The Stable Marriage Problem, Brandeis Review, vol 12(1), '92.

Localized Buffering is a consequence of product requirements. Most routers today are built with backplane chassis that allow line cards to be added independently. Each line card may have different features, support different protocols, and process cells at different line rates. This allows customers to purchase line cards incrementally, based on their budget, bandwidth needs, and feature requirements. Because each line card may process cells differently, it becomes necessary to process and store cells locally on the arriving line card.

✎**Example 4.1.** Figure 4.1 shows a router with multiple line cards. The line cards perform different functions. The router in the figure has three gigabit Ethernet line cards for local area network (LAN) switching, and two wide area network (WAN) line cards that provide an uplink to the Internet. The backup WAN line card, has a large congestion buffer due to its slow uplink. A 100GE LAN card that provides for faster connectivity is also shown. Finally, a fiber channel [61] based line card (which implements a separate protocol), provides connectivity to storage LANs. The five different kinds of line cards in the router might all have different architectures — *i.e.*, different packet processing ASICs, features, and memory architectures.

Because of the costs involved in deploying high-speed routers, customers usually deploy new line cards separately over time, on an as-needed basis. A CIOQ router allows this, and so can be upgraded gradually. As a consequence, it has become the most common architecture among high-speed routers today.[1] Cisco Systems, currently the world's largest manufacturer of Ethernet switches and IP routers, deploys the CIOQ architecture in Enterprise [4], Metro [62], and Internet core routers. [5].

[1]In contrast, low-speed Ethernet switches and routers, *e.g.*, wireless access points, small office home office (SOHO) routers, *etc.*, are sold in fixed-function units (colloquially referred to as "pizza-boxes") which are not built for easy upgrades.

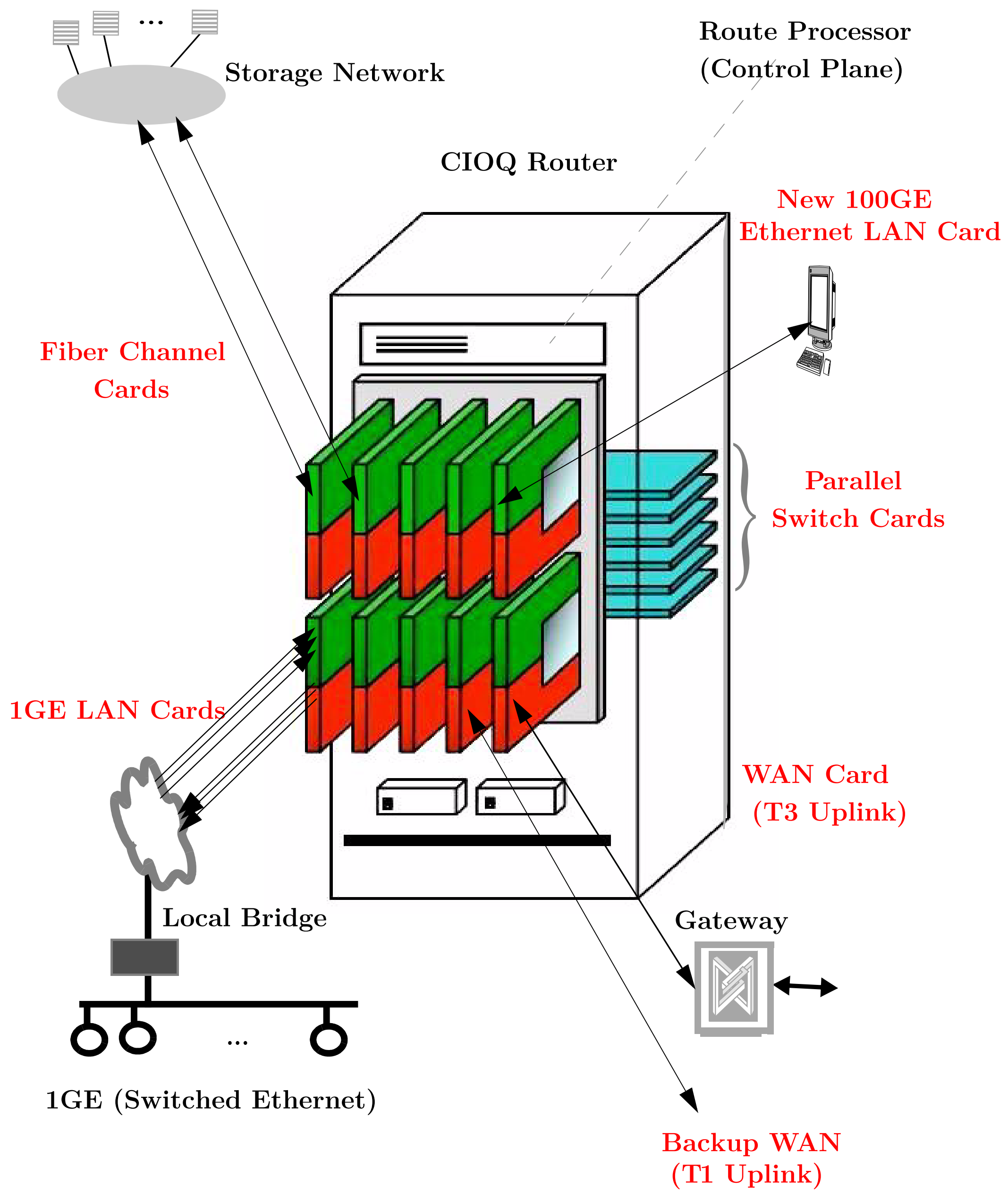

Figure 4.1: *A router with different line cards.*

4.1.1 A Note to the Reader

This chapter pertains to analytical work done over several years of research on CIOQ routers by multiple researchers. Many CIOQ scheduling algorithms have been proposed to obtain statistical guarantees (such as 100% throughput), as well as deterministic performance guarantees such as work-conservation (to minimize average delay) and emulation of an OQ router (to minimize the worst-case delay faced by packets). In this chapter, we only focus on the emulation of an OQ router, and we consider both FCFS and PIFO queuing policies. If a CIOQ router emulates an OQ router, then it automatically guarantees 100% throughput and is also work-conserving.

We make several fine points in this chapter and attempt to succinctly capture the underlying theory behind the emulation of OQ routers. We want to do three things — (1) present the existing theory in a very simple manner, (2) derive more intuitive algorithms that emulate an OQ router using the pigeonhole principle, and (3) re-state previously known results in terms of the pigeonhole principle.

In order to do this, we present the material in an order which is *not* chronological, but which builds up the theory step by step. We present only the results that, in hindsight, are insightful and clarifying to the reader.[2]

4.1.2 Methodology

We will use our constraint set technique (first described in Chapter 2) which is a formal way to apply the pigeonhole principle and analyze the CIOQ router. We will consider the crossbar-based CIOQ router, since this is the most common switch fabric deployed today. However, as we will see, this basic application of constraint sets has two problems when it is applied to this non-single-buffered router — (1) It requires us to make certain assumptions about the arrival traffic, and (2) It is only useful in analyzing an FCFS CIOQ router; we cannot analyze a PIFO-CIOQ router.

As we will see, the main reason why we cannot analyze a PIFO-CIOQ router is that the basic constraint set technique attempts to specify the transfer time of a packet

[2]A chronological order of the related work is described in Section 4.10.

to the output line card as soon as the packet arrives. It does not take full advantage of the architecture of a CIOQ router — *i.e.*, the switch fabric does not immediately need to commit to transferring packets that are waiting in the input line card, especially if there are more urgent packets destined to the same output.

Therefore, in the later part of this chapter, we will extend the constraint set technique to analyze and take advantage of the architecture of the (non-single-buffered) CIOQ router. We will show that this *extended constraint set* technique allows us to analyze both FCFS and PIFO CIOQ routers without making any assumptions about the arrival traffic. As we will see, this requires us to use a class of algorithms called *stable matching algorithms* [63] to schedule the crossbar, so as to meet the conditions mandated by the extended constraint set technique.

4.2 Architecture of a Crossbar-based CIOQ Router

Figure 4.2 shows an example of a crossbar-based CIOQ router. We denote the CIOQ router to have a speedup of S, for $S \in \{1, 2, 3, \ldots, N\}$ if it can remove up to S cells from each input and transfer at most S cells to each output in a time slot. Note that the CIOQ router can be considered to be a generalization of the IQ router and the OQ router. At one extreme, in an IQ router (*i.e.*, $S = 1$) only one cell can reach the output from every input in a time slot. At the other extreme, in an OQ router (*i.e.*, $S = N$), all the arriving N cells can be transferred to a specific output in a given time slot.

The crossbar-based CIOQ router shown in the figure has a speedup of $S = 2$, and has $N = 8$ ports. As described in Chapter 3, a crossbar fabric is restricted to perform a matching (*i.e.*, a permutation) between inputs and outputs in every time slot. If the crossbar has a speedup S, then it can perform two independent matchings in every time slot, as shown in Figure 4.2.

We will assume that cells are logically arranged on each line card as N virtual output queues (VOQs). VOQs allow an input to group cells for each output separately,

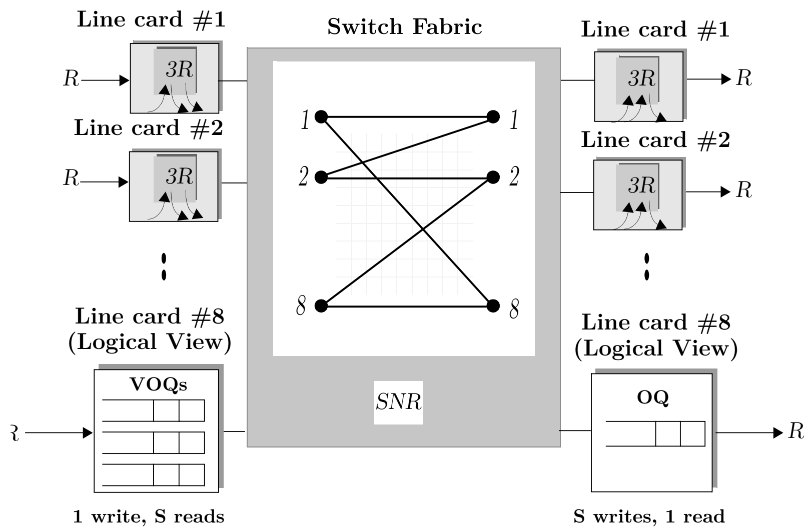

Figure 4.2: *The physical and logical view of a CIOQ router with crossbar speedup $SNR = 2NR$.*

and so a total of N^2 VOQs are maintained in the router. VOQs are needed to prevent *head of line blocking* [64]. Head of line blocking occurs when the head-of-line cell c, queued at some input i, destined to some output j, prevents other cells at input i from being scheduled. This can occur if output j is receiving cells from some other input in the same time slot. Hence cell c cannot be scheduled (because its output j is busy), while cells queued behind c (destined to other outputs) cannot be scheduled because they are not at the head of the input queue.

4.3 Background

In [65], Charny proved that a CIOQ router can emulate an FCFS-OQ router under restricted arrival traffic. Her algorithm was motivated by the following nice idea, *i.e.*,

Idea. *"We can use* maximal matching[a] *scheduling algorithms to emulate an FCFS-OQ router".*

[a]In a maximal matching, by definition, if there is a packet waiting at input i destined to output j, then either input i or output j is part of the matching.

We will first present her proof, and then derive a simpler and more intuitive proof based on the pigeonhole principle.

4.3.1 Charny's Proof

Charny [65] placed the following restrictions on the arrival traffic when she analyzed the CIOQ router:

1. **Outputs can only be finitely congested:** Charny constrained the arriving traffic to be *single leaky bucket constrained.* The arriving traffic is said to be single leaky bucket constrained if for every output j, the number of cells that arrive destined to j in the time interval (t_1, t_2) is given by $N(t_1, t_2) \leqslant \lambda_j(t_2 - t_1) + B_j$, where B_j is some constant, and $0 \leqslant \lambda_j < 1$ for the traffic to be admissible.[3] Note that if $\lambda_j > 1$, then output j receives more traffic than it can handle in the long term – this means that we need a router that supports a faster line rate! We can see that the above traffic constraint is equivalent to assuming that the shadow OQ router has a finite buffer size B_j for every output, and that no output is over-subscribed. If any output becomes congested and has more than B_j cells destined to it temporarily, then such cells are dropped.
2. **Inputs can only receive one cell per time slot:** Charny restricted the arrival traffic model such that no more than one cell can arrive at each input of the CIOQ router, in any given time slot.

In practice, both the above restrictions are always true:

[3]Refer to Appendix B for an exact definition.

1. Router buffers are finite and their size is determined during design. If we let $B = \max\{\forall j, B_j\}$, then the output buffer size is bounded by B.
2. Cells arrive on physical links, and even though there may be multiple input channels that can send cells in parallel, there is a cumulative maximum rate at which a single cell can arrive.

We are now ready to present the main theorem in Charny's work [65]:

Theorem 4.1. *(Sufficiency, by Reference) Any maximal algorithm with a speedup* $S > 2$*, which gives preference to cells that arrive earlier,*[4] *ensures that any cell arriving at time* t *will be delivered to its output at a time no greater than* $t + \lceil B/(S-2) \rceil$*, if the traffic is single leaky bucket* B *constrained.*

Proof. This is proved in section 2.3, Theorem 5 of [65].[5] We provide the outline of the proof here. Charny uses a maximal matching algorithm (called oldest cell first) that gives priority to cells that arrive earlier to the CIOQ router, and uses the fact that for any maximal algorithm, if there is a cell waiting at input i destined to output j, then either input i is matched or output j is matched (or both).[6] The proof counts all cells (called competing cells) that can prevent a particular cell from being transferred, and classifies the competing cells into two types – cells at input i, or cells destined to output j. It is shown that after a cell arrives, it cannot be prevented from being transferred to output j for more than $\lceil B/(S-2) \rceil$ time slots. ❒

4.3.2 Why Is Charny's Proof Complex?

The argument presented in Charny's proof above is somewhat complex, for two reasons. Consider any cell c:

1. Cell c can be repeatedly prevented – by competing cells arriving to the same input later than it – from being transferred to its output over multiple time slots. *It would be nice if we can prevent this from happening.*

[4] This is defined in section 2.3 in [65].

[5] Charny uses a dual leaky bucket traffic model. The result in [65] has been restated here for the single leaky bucket model to facilitate comparison between Theorem 4.2 and 4.1.

[6] A similar analysis was used in [22].

2. Cell c can be overtaken – by competing cells arriving later at different inputs but destined to the same output as cell c – and potentially prevented from getting serviced by the output. *It would be nice if we can somehow ensure that this does not affect when cell c gets serviced by the output.*

In what follows, we solve both the above problems [66] by fixing the time at which a cell is transferred from the input to the output ("transfer time"), as soon as it arrives. In this way, the time at which a cell is serviced by the output can't be affected by cells arriving later than it.

☞**Observation 4.1.** Note that fixing a cell's transfer time does not prevent the cell from being overtaken by other cells arriving later at different inputs, but destined to the same output. It merely ensures that these other cells do not affect the time at which cell c gets serviced by the output.

4.4 Analyzing FCFS-OQ Routers Using the Pigeonhole Principle

First consider the physical structure of the crossbar CIOQ router. If a cell c arrives at input i destined for output j, the router is constrained to transfer the cell only when input i and output j are both free. So we can think of cells contending with cell c (because they are from input i or destined to output j) as *pigeons*, contending for a transfer time (*pigeonholes*). As described in Chapter 2, constraint sets are a convenient accounting method to maintain and update this information. We are now ready to analyze the conditions under which the CIOQ router will emulate an FCFS-OQ router. We will use the algorithm described below, which is a direct consequence of the pigeonhole principle. More formally, the algorithm is an example of a *time reservation algorithm* [67, 68, 69, 70], since it reserves a future time for the arriving cell to be transferred from the input to the output, immediately on arrival of a cell.

Algorithm 4.1: Constraint set-based time reservation algorithm.

```
1 input  : Arrival and departure times of each cell.
2 output : A transfer time for each cell.
3 for each cell $c$ do
4     When a cell arrives at input $i$ destined to output $j$ with FCFS-OQ departure time
      $DT$, the cell is scheduled to depart at the first time in the future (larger than $DT$)
      when both the input $i$ and output $j$ are free to participate in a matching.
```

We start by describing the algorithm when speedup $S = 1$, before generalizing to larger speedup values:

1. **Maintaining constraint sets**: All inputs and outputs maintain a constraint set. Each entry in the constraint set represents an opportunity to transmit a cell in the future, one entry for each future time slot. For each future time slot that an input is busy, the corresponding entry in its constraint set represents a cell that it will transmit across the switch fabric to an output. Similarly, for each future time slot that an output is busy, the entry represents a cell that it will receive from one of the inputs. If, at some time in the future, there is no cell to be transferred from an input (or to an output), then the corresponding entry is free and may be used to schedule newly arriving cells.

2. **Negotiating a constraint-free time to transfer**: When a cell arrives at input i destined to output j, input i communicates its input constraint set to output j and requests a time in the future for it to transmit that cell. Output j then picks the first time in the future t_f in the interval $(DT, DT + k)$ (where k is a constant which we will determine shortly) for which both input i and output j are free to transmit and receive a cell, *i.e.*, time index t_f is free in the constraint sets of input i and output j. Output j grants input i the time slot t_f in future for transmitting the cell.

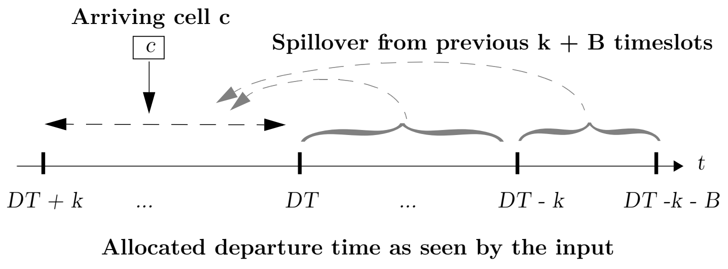
Arriving cell c
c
Spillover from previous k + B timeslots
DT + k
...
DT
...
DT - k
DT -k - B
t
Allocated departure time as seen by the input

Figure 4.3: *The constraint set as maintained by the input.*

3. **Updating constraint sets**: Both input i and output j update their respective constraint sets to note the fact that time t_f in the future is reserved for transmitting the cell from input i to output j in the CIOQ router.

Note that, for any crossbar speedup S, an entry in the input constraint set is said to be free in a particular time slot if the input is scheduled to send fewer than S cells. Likewise, an entry in the output constraint set is said to be free if the output is scheduled to receive fewer than S cells (from any input) in the corresponding time slot.

4.5 A Simple Proof to Emulate an FCFS CIOQ Router

We now use constraint sets to find the value of k for which every packet in the CIOQ router is transferred from its input to its output within k time slots of its FCFS-OQ departure time, *i.e.*, $t_f \leqslant DT + k$ or $t_f \in (DT, DT + k)$ (where t is the arrival time of a cell and k is a constant). The larger the speedup, the smaller the value of k.

Lemma 4.1. *The number of time slots available in the input constraint set (ICS) for any input i at any given time is greater than*

$$\boxed{\lceil k - \lfloor (k + B)/S \rfloor \rceil. \qquad (4.1)}$$

Proof. Consider a cell that arrives to input i at time t, destined for output j with FCFS-OQ departure time DT. The cell is scheduled to be transferred from input i to output j in the CIOQ router in the interval $(DT, DT + k)$, as shown in Figure 4.3. Since the traffic is single leaky bucket (B) constrained, no cell that arrived before time $DT - B$ at input i has an FCFS-OQ departure time in the interval $(DT, DT + k)$. Hence, no cell that arrived before time $DT - (B + k)$ at input i is allocated to be transferred from input i in the CIOQ router in the interval $(DT, DT + k)$. If the speedup is S, then the number of time slots available in the input constraint set for the newly arriving cell is at least $[k - \lfloor (k + B)/S \rfloor]$. ❐

✎**Example 4.2.** Consider a CIOQ router with $B = 100$ and $k = 150$. Assume that until now all cells have been given a transfer time within $k = 150$ time slots of its FCFS-OQ departure time, DT. Consider a cell c that arrives at time $t = 1000$. If it came to the shadow OQ router, it can never see more than $B = 100$ cells waiting at its output. Its FCFS-OQ departure time, DT (which is allocated by the shadow OQ router based on the number of cells already at the output) will be in the time interval $[1000, 1100]$. Say, $DT = 1055$. We will attempt to give cell c a transfer time within $k = 150$ time slots of DT, *i.e.*, in the interval $[1055, 1205]$. How many cells could have already requested a transfer time between $[1055, 1205]$ at the time of arrival? Definitely, no cells that arrived before time $1055 - 100 - 150 = 805$. For example, the cell that arrived at time 804, must have had a departure time less than 904 and hence a transfer time lesser than 1054, and will not interfere with our cell c. If not, it would violate the fact that until now all cells have been given a transfer time which is within $k = 150$ time slots of its FCFS-OQ departure time.

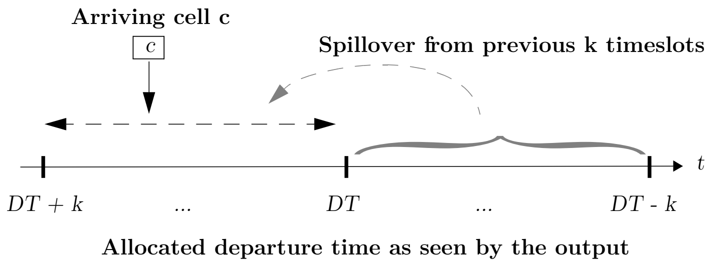
Arriving cell c
c
Spillover from previous k timeslots
t
DT + k
...
DT
...
DT - k
Allocated departure time as seen by the output

Figure 4.4: *The constraint set as maintained by the output.*

Lemma 4.2. *The number of time slots available in the output constraint set (OCS) for any output at any given time is greater than*

$$\boxed{[k - \lfloor (k/S) \rfloor]. \qquad (4.2)}$$

Proof. Consider a cell that arrives at input i at time t, destined for output j with FCFS-OQ departure time DT. The cell is scheduled to be transferred from input i to output j in the CIOQ router in the interval $(DT, DT + k)$, as shown in Figure 4.4. Since all cells are scheduled to be transferred in the CIOQ router within k time slots of their FCFS-OQ departure time, no more than k cells that have FCFS-OQ departure times in the interval $(DT - k, DT - 1)$ can already have been allocated to be transferred to output j in the CIOQ router in the interval $(DT, DT + k)$. Thus, if the speedup is S, then the number of time slots available in the output constraint set for the newly arriving cell is at least $[k - \lfloor (k/S) \rfloor$.[7] ❐

Lemma 4.3. *(Sufficiency) With a speedup $S > 2$, the algorithm ensures that each cell in the CIOQ router is delivered to its output within $[B/(S - 2)]$ time slots of its FCFS-OQ departure time, if the traffic is single leaky bucket B constrained.*

[7]We do not consider cells that have an FCFS-OQ departure time in the interval $(DT + 1, DT + k)$, since the output policy is FCFS and these cells will be considered only after cell c is allocated a time $t_f \in (DT + 1, DT + k)$ for it to be transferred from input to output in the CIOQ router.

Proof. (Using constraint sets). Consider a cell that arrives at time t. It should be allocated a time slot t_f for departure such that $t_f \in ICS \cap OCS$. A sufficient condition to satisfy this is that $[k - \lfloor (k+B)/S \rfloor] > 0$, $[k - \lfloor k/S \rfloor] > 0$, and $[k - \lfloor (k+B)/S \rfloor + k - \lfloor k/S \rfloor] > k$. This is always true if we choose $k > B/(S-2)$. ❐

Theorem 4.2. *(Sufficiency) With a speedup $S > 2$, a crossbar can emulate an FCFS-OQ router if the traffic is single leaky bucket B constrained.*

Proof. This follows from Lemma 4.3. ❐

By showing that the FCFS-CIOQ router emulates an FCFS-OQ router, it immediately follows from Theorem 4.2 that the router has bounded delay, and 100% throughput.

4.5.1 Interesting Consequences of the Time Reservation Algorithm

The time reservation algorithm, which is a consequence of using constraint sets, has three interesting consequences:

1. The application of the constraint set technique leads to an intuitive understanding of the scheduling constraints of the crossbar, and leads to a simpler scheduling algorithm. It simplifies Charny's well-known result (Theorem 4.1).
2. A more general result was later proved by Dai and Prabhakar (using fluid models) [12], and later by Leonardi et al. [71] using Lyapunov functions. However, unlike the work in [12] and [71], constraint sets lead to a hard bound on the worst-case delay faced by a packet in the CIOQ router. In other words, when subject to the same leaky bucket constrained arrival patterns, cells depart from the CIOQ router and the FCFS-OQ router at the same time, or at least within a fixed bound of each other.
3. The algorithm does not distinguish arriving cells based on their output destination. It only requires arriving cells to be stored in an input queue sorted based on their allocated departure time, irrespective of the outputs that they

are destined to. This means that line cards do not need VOQs, and it leads to different queuing architectures.

4.6 The Problem with Time Reservation Algorithms

There are two problems with the analysis described above — (1) We had to assume that the buffers were finite, and (2) We have no simple way to extend the analysis to a PIFO CIOQ router. While the former is a practical assumption to make,[8] it is difficult to see how we can extend the time reservation algorithm to support a PIFO queueing policy. This is because we fixed the time at which a cell is transferred at the time of its arrival. If later cells that were of higher priority came to the same input, they would not be able to use the time slots that were allocated to the lower priority cells (which may be destined to the same or even different outputs) that arrived earlier. If we look at Figure 4.3 and Figure 4.4, the number of cells that compete for the same interval $(DT, DT + k)$ can be unbounded since cells arriving later can be of higher priority, and this can happen indefinitely. This is a problem with any time reservation algorithm.

In summary, the main problem with time reservation algorithms is that every arriving cell c *immediately* contends for a time slot to be transferred to the output, even though it may not be scheduled for departure until some time in the future. This can happen because there are other cells already at the output that depart before cell c, or that it have a much lower priority than other cells that are already destined to cell c's output. So we need some way to *delay* such a cell c from immediately contending for a time slot. More generally, we need some way to indicate to the scheduler that cell c has a lower priority for transfer, so that the scheduler may choose to schedule some other higher-priority cell instead of cell c.

[8]In what follows, we will see that we can eliminate this assumption too.

4.7 A Preferred Marriage Algorithm for Emulating PIFO-OQ Routers

Chuang et al [13] were the first to show that if the priority of cells could be conveyed via a preference list by each input and output to a scheduler (without scheduling cells immediately on arrival), then it is possible for a CIOQ router to emulate a PIFO-OQ router. Their main idea was that a central scheduler would compute a matching (marriage) between the inputs and outputs based on their preferences for each other. The *preferred marriages* that are computed have the property that they are stable. (See Box 4.1.)

They introduce a counting technique and monitor the progress of every cell[9] to ensure that it reaches its output on time, as compared to a PIFO-OQ router.

In what follows, our goal is to show that the counting technique introduced in [13] can be viewed as a re-statement of the pigeonhole principle. We will see that in order to do this, we will have to:

1. Indicate the priority of cells to the scheduler (Section 4.8.1),
2. Extend the application of the pigeonhole principle to make it aware of cell priorities (Section 4.8.2), and,
3. Find the conditions under which this extended pigeonhole principle is satisfied (Section 4.8.3).

[9]The authors do this via the use of a variable called *slackness*.

✂Box 4.1: The Stable Marriage Problem✂

Consider the following succinct description of stable marriages, defined in [72] —

Stable Marriages: *Given N men and N women, where each person has ranked all members of the opposite sex with a unique number between 1 and N in order of preference, marry the men and women off such that there are no two people of opposite sex who would both rather have each other than their current partners. If there are no such people, the marriages are "stable".*

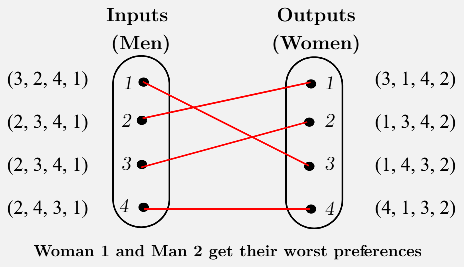

Figure 4.5: *A stable marriage*

An easier way to understand this is captured in the following quote [73] which presupposes a married woman who professes interest in marrying another man. If she receives the reply, *"Madam, I am flattered by your attention, but I am married to someone I love more than you, so I am not interested"*, and this happens to any woman who wants to switch (or vice versa) then the set of marriages is said to be stable.

☞**Observation 4.2.** The algorithm was first applied for pairing medical students to hospital jobs, and an example is shown in Figure 4.5. Gale and Shapely [63] proved that there is *always* a set of stable marriages, irrespective of the preference lists. The latter fact is key to analyzing crossbar routers.

In a CIOQ router, inputs and outputs maintain a priority list as described in Section 4.8.1. The output prefers inputs based on the position of cells in its output priority queue, *i.e.*, based on the departure order of cells destined to that output. Similarly, inputs prefer outputs based on the position of the cells in the input priority queues. In the context of routers, a set of marriages is a matching. Since there is *always* a stable marriage, a stable matching ensures that for every cell c waiting at an input queue:

1. Cell c is part of the matching.
2. A cell that is ahead of c in its input priority list is part of the matching.
3. A cell that is ahead of c in its output priority list is part of the matching.

So we can say that for every cell that is not transferred to its output in a scheduling phase, the number of contentions will decrease by one in every matching. With speedup two, this is enough to satisfy Property 4.1, which is required to emulate an OQ router.

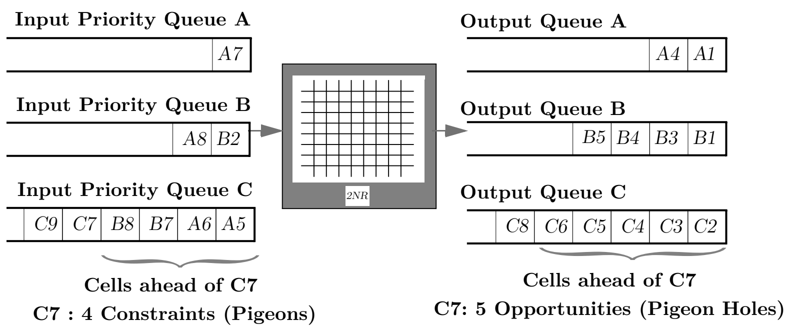

Figure 4.6: *Indicating the priority of cells to the scheduler.*

4.8 A Re-statement of the Proof

4.8.1 Indicating the Priority of Cells to the Scheduler

In order to convey the priority of cells to a scheduler, we will need a *priority queue* to indicate the departure order of cells to the scheduler. A priority queue (as shown in Figure 4.6), gives the scheduler flexibility in servicing higher-priority cells that are still at their inputs, irrespective of their arrival time. The queue has the following features:[10]

1. Arriving packets are "pushed-in" to an arbitrary location in the queue, based on their departure order.
2. The position of the packets in the queue determines their priority of departure.
3. Once the packet is inserted, the relative ordering between packets in the queue does not change.
4. A packet can depart from an arbitrary location in the queue if packets ahead of it in the queue are unable to depart.

[10]Since a cell can leave from an arbitrary location in a priority queue, it has been referred to as a push in random out (PIRO) queue in literature [13].

✎**Example 4.3.** The example in the figure shows a 3x3 CIOQ router. The ports are labelled A, B, and C. The input priority queue for ports A, B, C, and the three output queues for ports A, B, and C are shown. In the example shown, cells $A5$, $A6$, $B7$, $B8$, $C7$, and $C9$ are at the input priority queue for port C. None of the cells in the input priority queue are as yet scheduled. The input indicates to the scheduler the priority of the cells via its input priority queue. For example, cell $C7$ has lower priority than cells $A5$, $A6$, $B7$, and $B8$; however, it has higher priority than cell $C9$. In a given time slot, if the scheduler matches output ports A and B to some other inputs, then cell $C7$ can be scheduled to leave before cells $A5$, $A6$, $B7$, and $B8$.

Why does cell $C7$ have a lower priority than, say, cell $B8$? We can see from the figure that cell $C7$ has five cells — $C2$, $C3$, $C4$, $C5$, and $C6$ — that are ahead of it at output port C. This means that cell $C7$ does not have to be scheduled in the next five time slots. However, cell $B8$ has only four cells — $B1$, $B3$, $B4$ and $B5$ — ahead of it in output port C. So, in a sense, cell $B8$ has to more urgently reach its output than cell $C7$.

☞**Observation 4.3.** Note that the introduction of a priority queue does not prevent cells from being queued in VOQs. In fact, there are two choices for implementation — (1) The cells may be queued in VOQs, or segregated further into queues based on differentiated classes of service for an output. An input priority queue is maintained separately to describe the order of priority between these cells, or (2) We can do away with VOQs altogether; the cells can be queued directly in an input priority queue.

4.8.2 Extending the Pigeonhole Principle

We can state the observations from Example 4.3 in terms of pigeons and pigeonholes.

1. **Pigeons:** For every cell c, the cells that are ahead of it in the input priority queue contend with it for a time slot, so that they can be transferred to their respective outputs. These cells are similar to contending pigeons.
2. **Pigeonholes:** Every cell that is already at cell c's output and has an earlier departure time than cell c allows the cell more time to reach its output. These cells represent distinct opportunities in time for cell c to be transferred to the output. We will use pigeonholes to represent these opportunities.

In order to ensure that cells waiting in the input priority queue reach the output on time, we will mandate the following inequality: *the number of opportunities (pigeonholes) for a cell is always equal to or greater than the number of cells that contend with it (pigeons)*. This inequality must be satisfied for every cell, as long as it resides in the input priority queue.

Let us contrast this to our analysis of SB routers. In an SB router, a cell gets allocated to a memory on arrival. This memory is available to the output when the cell needs to be read at its departure time. So, we can forget about the cell once it has been allocated to a memory. In contrast, in a CIOQ router, the cell may remain in the input priority queue for some time before it is transferred to the output. So we have to continually monitor the cell to ensure that it can reach the output on time for it to depart from the router. This is formalized in Algorithm 4.2.

We maintain two constraint sets to track the evolution of every cell — (1) *An opportunity set* (pigeonholes) that constrains the number of time slots that are available for a cell to be transferred, and (2) *A contention set* (pigeons) to track the cells that contend with a cell and prevent it from being transferred to its output.

✎**Example 4.4.** Table 4.1 shows an example of the opportunity and contention sets for each cell in the input priority queue for port C of Figure 4.6. Both these sets are easily derived from the position of the cell in its input priority queue and the state of the output queue as shown in Figure 4.6. Note that for every cell in Table 4.1, the number of opportunities (column 5) is currently greater than the number of contentions (column 3).

Algorithm 4.2: Extended constraint sets for emulation of OQ routers.

```
1 input  : CIOQ Router Architecture.
2 output: A bound on the number of memories, total memory, and switching
          bandwidth required to emulate an OQ router.
3 for each cell $c$ which arrives at time $T$ and departs at time $DT$ do
4     for $t \leftarrow\in \{T, T+1, \ldots, DT\}$ do
5         Opportunity Set (pigeonholes) ← cells ahead of $c$, already at its output
6         Contention Set (pigeons) ← cells ahead of $c$ at its input
7         if cell $c$ is still at the input then
8             Ensure: |Pigeonholes| ≥ |Pigeons|.
```

Table 4.1: *An example of the extended pigeonhole principle.*

Cell	Cells Ahead in Input	#Contentions (Pigeonholes)	Cells Ahead in Output	#Opportunities (Pigeons)
$A5$	-	0	$A1$,$A4$	2
$A6$	$A5$	1	$A1$,$A4$	2
$B7$	$A5$,$A6$	2	$B1$,$B3$,$B4$,$B5$	4
$B8$	$A5$,$A6$,$B7$	3	$B1$,$B3$,$B4$,$B5$	4
$C7$	$A5$,$A6$,$B7$,$B8$	4	$C2$,$C3$,$C4$,$C5$,$C6$	5
$C9$	$A5$,$A6$,$B7$,$B8$,$C8$	5	$C2$,$C3$,$C4$,$C5$,$C6$,$C8$	6

4.8.3 Using Induction to Enforce the Pigeonhole Principle

How do we ensure that the conditions described in Algorithm 4.2 are met? To do this, we use *simple induction* to track the change in size of the contention and opportunity set in every time slot that a cell waits at the input. But we have two problems — (1) In every time slot, for every cell, it is possible that the contention set increases in size by one, due to newly arriving cells that may have higher priority than it. Also, the opportunity set will decrease in size by one, because a cell will depart from the output. The former increases the number of pigeons, while the latter decreases the number of pigeonholes available for the cell. Both of these are detrimental from the cell's point of view, and (2) When a cell arrives, it is inserted into the input priority queue. At the time of insertion, it must at least have equal or more opportunities as the number of contenting cells. If not, it may never make it to the output on time.

So, in order to meet the induction step (between consecutive time slots that the cell waits at the input) and the induction basis case (at the time slot when a cell arrives to the input), we mandate the following properties:

✓**Property 4.1.** **Induction Step:** In every time slot, the switch scheduler does the following at least *twice* — it either decrements the size of the contention set or increments the size of the opportunity set for every cell waiting at the input, *i.e.*, *it either decrements the number of pigeons or increments the number of pigeonholes for every cell.*

✓**Property 4.2.** **Induction Basis Case:** On arrival, a cell is inserted into a position in the input priority queue such that it has equal or more opportunities to leave than the number of contentions, *i.e.*, *it has equal or more pigeonholes than the number of pigeons.*

4.9 Emulating FCFS and PIFO-OQ Routers Using the Extended Pigeonhole Principle

We have shown that the method of proof introduced in [13] is simply a re-statement of the pigeonhole principle, applied to every cell for every time slot that it waits at the input. We are now ready to re-state the results for the preferred marriage algorithm in [13] as follows — If Property 4.1 and Property 4.2 are met, the conditions for the extended constraint set technique (Algorithm 4.2) are satisfied. It follows that all cells reach their output on time, and a CIOQ router can emulate a PIFO-OQ router.

In order to satisfy Property 4.1, the authors [13] ensure that the preferred marriages are stable [63], and operate the crossbar at speedup $S = 2$. Also, in order to satisfy Property 4.2, the authors insert an arriving cell as far ahead in the input priority queue as required so that it has more opportunities than the number of contending cells at the time of insertion.[11] In an extreme case, the arriving cell may have to be

[11]Note that there are many insertion policies that satisfy Property 4.2, since we only need to insert an arriving cell at any position where it has more opportunities than the number of contending cells.

inserted at the head of line. Note that the memories in the input line cards need to run at rate $3R$ (to support one write for the arriving cell, and up to two reads into the crossbar). The memories in the output line cards also need to run at $3R$ (to support two writes for cells arriving from the crossbar, and one read to send a cell to the output line). This requires a total memory bandwidth of $6NR$. We can now state the following theorem:

Theorem 4.3. *(Sufficiency, by Citation) A crossbar CIOQ router can emulate a PIFO-OQ router with a crossbar bandwidth of* $2NR$ *and a total memory bandwidth of* $6NR$.

Proof. Refer to [13] for the original proof. ❐

4.10 Related Work

In what follows, we present a brief chronological history of the research in CIOQ routers. Due to the large body of previous work, this section is by no means exhaustive, and we only present some of the key salient results.

4.10.1 Statistical Guarantees

We first consider CIOQ routers with crossbar speedup $S = 1$. Recall that such a CIOQ router is simply an IQ router. IQ routers that maintain a single FIFO buffer at their inputs are known to suffer from head-of-line blocking. Karol et al. [64] showed that the throughput of the IQ router is limited to 58% when the input traffic is independent, identically distributed (i.i.d)[12] Bernoulli, and the output destinations are uniform.

Based on the negative results in [64], several authors [75, 76, 77, 78] did analytical and simulation studies of a CIOQ router (which maintains a single FIFO at each input) for various values of speedup. These studies show that with a speedup of four to five, one can achieve about 99% throughput when arrivals are i.i.d at each input and when the distribution of packet destinations is uniform across the outputs.

[12] Refer to Appendix B for an exact definition.

✂Box 4.2: The Stability of Forced Marriages✂

The algorithm described to emulate an OQ router in Section 4.9 is an example of *preferred marriage*. Inputs and outputs maintain preference lists and communicate their choices to a central scheduler. The central scheduler computes a stable marriage and attempts to satisfy (to the best of its abilities) the preferences of the inputs and the outputs.

Suppose we deny the inputs and outputs choice. We can ask — What happens if the matchings (marriages) that we compute are *forced marriages*?[a] Prabhakar et al [41] describe an algorithm called MUCFA (Most Urgent Cell First Algorithm) whose analogy to forced marriages is as follows:

MUCFA: *When a cell arrives it is given an* urgency, *which is the time remaining for it to leave the OQ router. A cell progressively becomes more urgent in every time slot. In each scheduling phase, an input is forced to prefer outputs that have more urgent cells, and outputs are forced to prefer inputs based on which inputs have more urgent cells. The preferences lists are created simply as a consequence of the urgency of a cell. Inputs and outputs have no say in the matter. In each scheduling phase, a centralized scheduler computes a forced marriage which is stable.*

The stable forced marriage computed by MUCFA gives priority to more urgent cells and has the following property. For every cell c waiting at an input queue:

1. Cell c is part of the matching.
2. A cell that is more urgent than c in its input priority list is part of the matching.
3. A cell that is more urgent than c in its output priority list is part of the matching.

Prabhakar et al. [41] show using a reductio-ad-absurdum argument that MUCFA can emulate an OQ router with speedup four.

☞**Observation 4.4.** It is interesting to apply the pigeonhole principle to analyze MUCFA. It can be easily seen that MUCFA satisfies Property 4.1 with speedup two. But there is no easy way to know whether MUCFA satisfies the induction base case, *i.e.*, Property 4.2 on cell arrival. This is because unlike the *preferred marriage* algorithm, MUCFA does not give *choice* to an input to insert an arriving cell into its preference list.

Coincidentally, similar to marriages in certain traditional societies, the algorithms for forced marriages for CIOQ routers [41] preceded the algorithms for preferred marriages [13]. Note that both sets of marriages can be made stable, and are successful in emulating an OQ router.

It has been shown via counterexample [74] that MUCFA with speedup two cannot emulate an OQ router. However, it is not known whether MUCFA can emulate an OQ router with speedup three, and this remains an interesting open problem.

[a]Not to be confused with *arranged marriage*, a much-maligned and misused term, incorrectly used in the modern world to describe a large class of marriages!

It was later shown that the low throughput of IQ routers is due to head-of-line blocking, and it can be overcome using virtual output queues (VOQs) [79]. This led to a renewed interest in IQ routers. Tassiulas and Ephremides [39] and McKeown et al. [11] proved that an IQ router with VoQs can achieve 100% throughput with a *maximum weight matching* (MWM) algorithm, if the input traffic is i.i.d and admissible; the outputs are allowed to be non-uniformly loaded. Dai and Prabhakar [12] generalized this result and showed that MWM can achieve 100% throughput provided that the input traffic satisfies the strong law of large numbers and is admissible. However, MWM is extremely complex to implement and has a time complexity, $O(N^3 \log N)$.

A different approach was considered by Chang et al. [80] and Altman et al. [81]. They showed (using the results of Birkhoff [82] and von Neumann [83]) that the crossbar can be scheduled by a fixed sequence of matchings (called frames)[13] such that the IQ router can achieve 100% throughput for any admissible arrival pattern. These are similar to time division multiplexing (TDM) techniques in the switching literature [84]. However, their usage of *frame scheduling* requires prior knowledge of the arrival traffic pattern, and the router needs to maintain a potentially long sequence of matchings, and so these techniques have not found usage in practice.

One would expect that the *maximum size matching* (MSM) algorithm, which maximizes the instantaneous bandwidth of the crossbar (the most efficient algorithm has a lower time complexity $O(N^{2.5})$ [85, 86]), would also be able to achieve 100% throughput. However, contrary to intuition, MSM is known to be unfair (if ties are broken randomly), can lead to starvation, and hence cannot achieve 100% throughput [87]. Not all MSM algorithms suffer from loss of throughput. In [88] it is shown that the longest port first (LPF) algorithm (an MSM algorithm that uses weights to break ties) achieves 100% throughput for Bernoulli arrivals.

In [89], Weller and Hajek give a detailed analysis on the stability of online matching algorithms (including MSM) using frame scheduling with a constrained traffic model. Our work in [90] extends some of the results in [89] by alleviating the traffic constraints and considering stochastic (Bernoulli) arrivals. It shows that with a slight modification

[13]This is similar to the idea of frame scheduling described in Chapter 3.

of frame scheduling (called batch scheduling), a class of MSM algorithms (called Critical Maximum Size Matching, or CMSM), can achieve 100% throughput for (uniform or non-uniform) Bernoulli i.i.d. traffic. Also, under the purview of batch scheduling, the unfairness of MSM reported in [88] is eliminated, and *any* MSM algorithm can achieve 100% throughput under Bernoulli i.i.d. uniform load. However, since the batch scheduling MSM algorithms described above can suffer from large worst case delay and have large time complexity, they are not used in practice.

There are two approaches that are more practical toward achieving 100% throughput. They involve speeding up the crossbar or randomizing the MWM scheduler. Dai and Prabhakar [12] in their seminal paper analyzed maximal matching algorithms and proved that any *maximal matching* algorithm can achieve 100% throughput with speedup, $S = 2$ for a wide class of input traffic. Subsequently, Leonardi et al. [71] also proved a similar result. The authors in [36, 91, 92, 93] took a different approach. They describe load balancing algorithms[14] that are simple to implement and achieve 100% throughput. They come at the expense of requiring two switching stages, and also require a higher crossbar speedup, $S = 2$.[15]

Tassiulas [94] showed that a randomized version of MWM (which is easier to implement) can achieve 100% throughput with speedup, $S = 1$. Later, Giaccone et al. [95] described other randomized algorithms that achieve 100% throughput with $S = 1$ and also closely realize the delay performance of the MWM scheduler.

4.10.2 Deterministic Guarantees

The algorithms described above make no guarantees about the delay of individual packets, and only consider average delay. The approach of mimicking (or emulating) an OQ router was first formulated in [41]. They showed that a CIOQ router with a speedup of four, and an algorithm called MUCFA (most urgent cell first algorithm),

[14]Note that the algorithms we proposed in [36] were first described for a parallel packet switch architecture, which is described in Section 6.7. But the results are immediately applicable to the CIOQ router.

[15]Note that when these algorithms have knowledge of the destination port of the incoming cell, they can emulate an FCFS-OQ router even under adversarial inputs.

✂Box 4.3: Work Conservation without Emulation✂

A work-conserving router gives a deterministic performance guarantee— it always keeps its outputs busy. In Chapter 1, we mandated FCFS-OQ emulation as a condition to achieve work-conservation. This gave us an additional deterministic performance guarantee for each individual packet, *i.e.*, it would leave at the same time as compared to an ideal FCFS-OQ router. We can relax this condition. In what follows, we will show that we can also apply the pigeonhole principle to find the conditions under which a CIOQ router is work-conserving *without* mandating FCFS-OQ emulation.

For a router to be work-conserving, we only need to ensure that its outputs are kept busy. So, in such a router, the order of departures of packets from outputs is irrelevant. It is sufficient that some packet leave the output at all times that the corresponding output in the shadow OQ router is also busy. So, from a cell's perspective, it is sufficient that either (1) some cell ahead of it in the input priority list leave the input, or (2) some cell *(irrespective of its departure time)* make it to the output in every scheduling opportunity. So we can make the following simple modification to the extended constraint set technique in order that our CIOQ router be work-conserving:

Algorithm 4.3: Extended constraint sets for work conservation.

```
1 input  : CIOQ Router Architecture.
2 output : A bound on the number of memories, total memory and switching
           bandwidth required to emulate an OQ router.
3 for each cell $c$ which arrives at time $T$ and departs at time $DT$ do
4     for $t \leftarrow\in \{T, T+1, \ldots, DT\}$ do
5         Opportunity Set (pigeonholes) ← any cells already at its output
6         Contention Set (pigeons) ← cells ahead of $c$ at its input
7         if cell $c$ is still at the input then
8             Ensure: |Pigeonholes| ≥ |Pigeons|.
```

As a consequence, the scheduler which ensures that the preferred marriages described in Section 4.9 are stable (and which was used to ensure that a CIOQ router can emulate a PIFO router) can be simplified. It no longer needs to be aware of the priority between cells destined to the same output. This leads to the following theorem:

Theorem 4.4. *A crossbar CIOQ router is work-conserving with a crossbar bandwidth of* $2NR$ *and a total memory bandwidth of* $6NR$.

can emulate an OQ router for arbitrary input traffic patterns and router sizes. Later, in their seminal paper [13], the authors improved upon the result in [41] and were the first to show that a CIOQ router can emulate an OQ router with speedup two.

In contrast, Charny [65] also considered emulation, but required assumptions on the arrival traffic and only considered FIFO traffic, while Krishna et al. [23] independently showed that a CIOQ router is work-conserving with speedup two. Note that Stoica et al. [96] also considered the emulation of a CIOQ router with speedup two. Their paper had an error, which was later fixed [97] in consultation with the authors in [13]. Recently, Firoozshahian et al. [98] presented a surprising algorithm that (unlike all other previous algorithms) only uses local information to emulate a constrained OQ router that performs "port-ordered" scheduling policies.

4.11 Conclusions

We have extended the work done on providing delay guarantees in [65] and PIFO emulation in [13], as follows:

1. Our work on time reservation algorithms, presented in Section 4.4 for the classical unbuffered crossbar, extends Charny's result [65]. In [65] it was shown that a maximal matching algorithm would lead to the main result (Theorem 4.1). The result relied on a scheduler that examines the contents of the input queues during each time slot to determine which cells to schedule. In contrast, the constraint set technique leads to an almost identical result (Theorem 4.2), using a simpler algorithm that schedules cells as soon as they arrive. While algorithms for IQ and CIOQ routers that schedule cells immediately upon arrival have been proposed before [67, 68, 69, 70], we are not aware of any previous work that shows when such algorithms can achieve 100% throughput, give bounded delay, and emulate an FCFS-OQ router.

2. The analysis of non-SB routers was a hard problem, because cells that arrive to non-SB routers do not get queued for their outputs immediately. They may be held at one or more intermediate points before they are transferred to their

respective outputs. We introduced the *extended constraint set technique*, which follows the path of the cell and enforces constraints on the cell on every time slot before it reaches its output. The extended constraint set technique has given us a powerful tool to analyze non-SB routers. It has clarified our understanding of the results in [13] and brought it under the purview of the pigeonhole principle.

Summary

1. In this chapter, we analyze the combined input output queued (CIOQ) router. In a CIOQ router, a packet is buffered *twice* — once when it arrives, and once before it leaves. The first buffer is on the line card it arrives on. The second buffer is on the line card it departs from.
2. Intuitively, the two buffers make the job easier for a central scheduler — if the switch is not ready to transfer packets as soon as they arrive, it can hold them in the input line card. Similarly, if the scheduler has the opportunity to send a packet to the output, it doesn't need to wait until the output line is free — the output line card will buffer it in the output until the line is free.
3. We use the constraint set technique (introduced in Chapter 2) to analyze the conditions under which a CIOQ router can emulate an OQ router.
4. The use of the constraint set technique implies that our scheduling algorithm attempts to reserve a time slot (either now or in the future) to transfer a cell to the output line card as soon as it arrives. As a consequence, our switching algorithm belongs to a class of algorithms called *"time reservation algorithms"*.
5. We show that a CIOQ router (using a time reservation algorithm) with a crossbar bandwidth greater than $2NR$ and a memory bandwidth greater than $6NR$ is work-conserving (Theorem 4.4) and can emulate an FCFS-OQ router (Theorem 4.2).
6. Our techniques lead to an intuitive understanding of the constraints faced in a CIOQ router, and also simplify the algorithms presented in [65].
7. However, we have to make assumptions (e.g., that the CIOQ router has a finite size buffer) in order to prove the above results. While these assumptions are practical, the constraint set technique unfortunately does not allow us to find the conditions under which a CIOQ router can emulate an OQ router that provides qualities of service.

8. This is because the basic constraint set technique does not take full advantage of the architecture of a CIOQ router as it commits to transferring a packet immediately on arrival.
9. We therefore introduce the extended constraint set technique to analyze the CIOQ router. The extended constraint set technique is a re-statement of the counting principle introduced in [13], and is a method to continuously track the progress of a cell while it waits at the input, in its attempt to be transferred to the output.
10. We re-state the previously known results in [13] in terms of the extended constraint set technique to show that a CIOQ router with a crossbar bandwidth of $2NR$ and a memory bandwidth of $6NR$ can emulate an OQ router that supports multiple qualities of service (Theorem 4.3).
11. The extended constraint set technique has broadened our understanding of routers. It brings under the purview of the pigeonhole principle other router architectures that are not single-buffered.