# Chapter 1: Introduction

*Nov 2007, Mendocino, CA*

## Contents


**1.1 Analogy** . . . . . . . . . . **1**

**1.2 Goal** . . . . . . . . . . **2**

**1.3 Background** . . . . . . . . . . **3**

1.3.1 The Ideal Router . . . . . . . . . . 3

1.3.2 Why is it Hard to Build a High-speed Ideal Router? . . . . . . . . . . 5

1.3.3 Why is Memory Access Time a Hard Problem? . . . . . . . . . . 7

1.3.4 Historical Solutions to Alleviate the Memory Access Time Problem . . 9

**1.4 Mandating Deterministic Guarantees for High-Speed Routers** . . **12**

1.4.1 Work Conservation to Minimize Average Packet Delay . . . . . . . . . . 13

1.4.2 Delay Guarantees to Bound Worst-Case Packet Delay . . . . . . . . . . 13

**1.5 Approach** . . . . . . . . . . **16**

1.5.1 Analyze Many Different Routers that Have Parallelism . . . . . . . . . . 16

1.5.2 Load-Balancing Algorithms . . . . . . . . . . 17

1.5.3 Caching Algorithms . . . . . . . . . . 18

1.5.4 Use Emulation to Mandate that Our Routers Behave Like Ideal Routers 19

1.5.5 Use Memory as a Basis for Comparison . . . . . . . . . . 20

**1.6 Scope of Thesis** . . . . . . . . . . **21**

**1.7 Organization** . . . . . . . . . . **23**

**1.8 Application of Techniques** . . . . . . . . . . **27**

**1.9 Industry Impact** . . . . . . . . . . **28**

1.9.1 Consequences . . . . . . . . . . 29

1.9.2 Current Usage . . . . . . . . . . 30

**1.10 Primary Consequences** . . . . . . . . . . **30**

**1.11 Summary of Results** . . . . . . . . . . **31**

# A Note to the Reader

The *Naga Jolokia*, a chili pepper with a Scoville rating of 1 million units that grows in northeastern India, Sri Lanka, and Bangladesh, is the hottest chili in the world, as confirmed by the Guinness World Records [1]. To understand just how hot it is, consider that to cool its blow torch-like flame, you would have to dilute it in sugar water over a million times![3] A milder way to put your taste buds to the test would be to stop by an Indian, Chinese, or Thai restaurant and order a *hot soup*. However, your request would be ambiguous; the context isn't helpful, and you might confuse the waiter — that's because the English language, among its many nuances and oddities, doesn't have unique terms to distinguish "temperature hot" and "spicy-hot".

Fortunately or unfortunately (depending on your perspective), high-speed routers offer no such confusion. They are definitely hot (and we are not referring to the colloquial use of the term by undergrads). They consume tremendous power — sometimes more than 5000 watts! — and dissipate that energy as heat. This means that packet processing ASICs on these high-speed routers are routinely built to withstand temperatures as high as 115°C— temperatures so high that even the smallest contact can blister your skin. And so, if you want to understand the workings of a high-speed router, I do not recommend that you touch it. Instead, here are some guidelines that I believe will help make your study of high-speed routers pleasant and safe.

- **Introduction:** The introductory chapter is written in a self-contained manner, with terms defined inline. It defines the general background and motivation for solving the memory access time problem for high-speed routers. Appendix A has a short tutorial on memory technology.
- **Independent Chapters:** The chapters in this thesis are written so that, for most part, they can be read independently. Section dependencies and additional readings are listed at the beginning of each chapter.
- **Organization:** Examples and Observations are categorized separately. The key idea in each chapter is specified in an *idea box*. Gray-shaded boxes contain supplementary information. A summary is available at the end of each chapter, and an index is provided at the end of the thesis.

[3] In comparison, a standard pepper spray (shame on you!) may register just over 2 million units — mighty hot, yet quickly neutralized in flowing water.

*"You are rewarding a teacher poorly*
*if you remain always a pupil".*

— Friedrich Nietzsche[†]

# 1 Introduction

## 1.1 Analogy

**Guru:** Consider a set of pigeon holes, each of which is capable of holding several pigeons. Up to $N$ pigeons may arrive simultaneously, and each pigeon must have immediate access to a pigeon hole. In the same interval in which the pigeons arrive, up to $N$ pigeons must leave their holes. How many pigeon holes are required to guarantee that the departing pigeons can leave their holes, and that the arriving pigeons can enter a hole?

**Shisya[1]:** It would require just one pigeon hole with an entrance sufficiently wide to allow $N$ pigeons to arrive and $N$ pigeons to depart simultaneously.

**Guru:** That is the ideal. But what if no pigeon hole has an entrance sufficiently wide?

**Shisya:** In that case, I would create $N$ pigeon holes. Each arriving pigeon would be directed to a separate pigeon hole. At any time, each pigeon hole must be wide enough to receive one pigeon and allow up to $N$ pigeons to depart.

**Guru:** Suppose I constrain the pigeon hole so that at any time it can either allow at most one pigeon to arrive, or allow at most one pigeon to depart? Furthermore, no pigeon may enter a pigeon hole while another is departing ...

[†] Friedrich Nietzsche (1844-1900), German Philosopher.
[1] Sanskrit: "student".

The question posed in the imaginary conversation is directly applicable to the design of Internet routers. The analogy is that an Internet router receives $N$ arriving packets (pigeons) in a time slot, one packet from each of its $N$ inputs. Each packet can be written and stored temporarily in one of several memories (analogous to pigeons arriving in a pigeon hole). Packets can later be read from a memory (pigeons departing from a hole). No more than $N$ packets (destined to each of $N$ outputs) can depart the router during any time slot.

In defining the problem in this manner, we make several assumptions and overlook some potential constraints. For example, (1) the available memory may in fact be several times slower than the rate at which packets arrive, (2) the memory may not be able to hold many packets, (3) the interconnect used to access the memories may not support the inputs (outputs) writing to (reading from) any arbitrary combination of memories at the same time, (4) when a packet arrives, its time of future departure may be unknown.

We can infer from the above dialogue that if the memory was faster, or if the packets arrived and departed at a slower rate, or if there were fewer constraints, it would be easier to ensure that packets can leave when they need to. We will see that the answer to the question in the conversation is fundamental to our understanding of the design of high-speed Internet routers.

## 1.2 Goal

Our goal in this thesis is to design high-speed Internet routers for which we can say something predictable and deterministic (we will formalize this shortly) regarding their performance. There are four primary reasons for doing this:

1. Internet end-users are concerned primarily about the performance of their applications and their individual packets.
2. Suites of applications (such as VoIP, videoconferencing, remote login, Netmeeting, and storage networking) are very sensitive to delay, jitter, and packet

drop characteristics, and do not tolerate non-deterministic performance (*e.g.*, unpredictable delays) very well.

3. Hackers or viruses can easily exploit non-deterministic and known deficiencies in router performance and run traffic-adversarial patterns to bring down routers and the Internet.
4. If the individual routers that form the underlying infrastructure of the Internet can be made to deliver deterministic guarantees, we can hope to say something deterministic about the performance of the network as a whole. The network would then potentially be able to provision for different classes of applications, cater to delay-sensitive traffic (*e.g.*, voice, Netmeeting, *etc.*), and provide end-to-end guarantees (*e.g.*, bandwidth, delay, *etc.*) when necessary.

In addition there are various secondary reasons why we want deterministic performance guarantees from our high-speed routers. For example, unlike computer systems, where occasional performance loss is acceptable and even unavoidable (*e.g.*, due to cache or page misses) router designers do not like non-deterministic performance in their ASICs. This is because they cause instantaneous stalls in their deep pipelines, as well as loss of performance and eventually even unacceptable packet loss. Also, many routers are compared in "bake-off" tests for their ability to withstand and provide guaranteed performance under even worst-case traffic conditions.

## 1.3 Background

### 1.3.1 The Ideal Router

Which factors prevent us from building an ideal router that can give deterministic performance guarantees? Consider a router with $N$ input and $N$ output ports. We will denote $R$ (usually denoted in Gb/s) to be the rate at which packets[2] arrive at every input and depart from every output. We normalize time to the arrival time

[2] Although packets arriving to the router may have variable length, for the purposes of this thesis we will assume that they are segmented and processed internally as fixed-length "cells" of size $C$. This is common practice in high-performance routers – variable-length packets are segmented into cells as they arrive, carried across the router as cells, and reassembled back into packets before they depart.

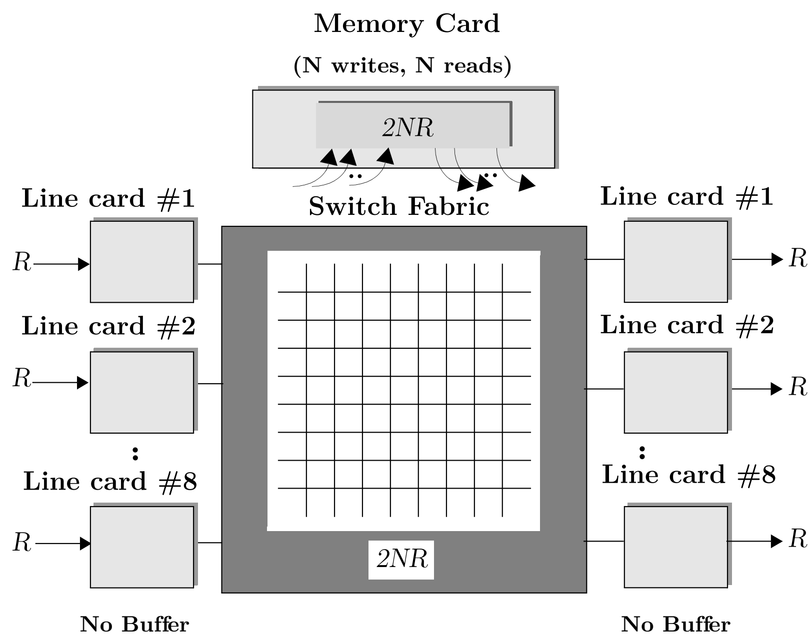



**Figure 1.1:** *The architecture of a centralized shared memory router. The total memory bandwidth and interconnect bandwidth are* $2NR$*.*

between cells ($C/R$) at any input, and refer to it as a *time slot*.[3] Assume that a single shared memory can store all $N$ arriving packets and service $N$ departing packets. This architecture is called the centralized shared memory router, and an example with $N = 8$ ports and $R = 10$ Gb/s (these numbers are typical of mid-range routers typically deployed by smaller local Internet Service Providers) is shown in Figure 1.1. The shared memory router would need a memory that would meet the following requirements:

1. **Bandwidth:** The memory would need to store all $N$ packets arriving to it and allow up to $N$ packets to depart from it simultaneously, requiring a total memory bandwidth of $2NR$ Gb/s.
2. **Access Time:** If we assume that arriving packets are split into fixed-size cells of size $C$, then the memory would have to be accessed every $A_t = C/2NR$ seconds.

[3] In later chapters we will re-define this term as necessary so as to normalize time for the router architecture under consideration.

For example, if $C = 64$ bytes, the memory would need to be accessed[4] every 3.2 ns.[5]

3. **Capacity:** Routers with faster line rates require a large shared memory. As a rule of thumb, the buffers in a router are sized to hold approximately $RTT \times R$ bits of data during times of congestion (where $RTT$ is the round-trip time for flows passing through the router[6]), for those occasions when the router is the bottleneck for TCP flows passing through it. If we assume an Internet $RTT$ of approximately 0.25 seconds, a 10 Gb/s interface requires 2.5 Gb of memory. With 8 ports, we would need to buffer up to 20 Gb.

If the memory meets these three requirements, and the switching interconnect has a bandwidth of $2NR$ to transport all $N$ arriving packets and all $N$ departing packets from (to) the central memory, then the packets face no constraints whatever on how and when they arrive and depart; and the router has minimum delay. Also, the memory is shared across all ports of the router, minimizing the amount of memory required for congestion buffering. This router behaves ideally and is capable of delivering the predictable performance that we need.

### 1.3.2 Why is it Hard to Build a High-speed Ideal Router?

What happens if we design this high-speed ideal router using available memory technology? Of course, we will need to use *commodity* memory technology. By a commodity part, we mean a device that does not stress the frontiers of technology, is widely used, is (ideally) available from a number of suppliers, and has the economic benefits of large demand and supply.[7] Two main memory technologies are available in

[4] This is referred to in the memory industry as the random cycle time, $T_{RC}$.

[5] The memory bandwidth is the ratio of the width of the memory access and the random access time. For example, a 32-bit-wide memory with 50 ns random access time has a memory bandwidth of 640 Mb/s.

[6] The word router in this thesis also refers to Ethernet, ATM, and Frame Relay "switches". We use these words interchangeably.

[7] In reality there is a fifth requirement. Many high-speed routers have product cycles that last five or more years, so router vendors require that the supply of these memories is assured for long periods.

the market today: SRAM and DRAM. Both offer comparable memory bandwidth; however, SRAMs offer lower (faster) access time, but lower capacity than DRAMs.

✎**Example 1.1.** At the time of writing, with today's CMOS technology, the largest available commodity SRAM [2] is approximately 72 Mbits, has a bandwidth of 72 Gb/s, an access time of 2 ns, and costs $70.[8]

To meet the capacity requirement, our router would need more than 275 SRAMs, and the memories alone would cost over $19K – greatly exceeding the selling price of an Enterprise router today! Although possible, the cost would make this approach impractical.

✎**Example 1.2.** The largest commodity DRAM [3] available today has a capacity of 1Gb, a bandwidth of 36 Gb/s, an access time of 50 ns, and costs $5.

We cannot use DRAMs because the *access rate*[9] is an order of magnitude above what is required. If our router has more than $N = 16$ ports, the access time requirement would make even the fastest SRAMs inapplicable.

The centralized shared memory router is an example of a class of routers called *output queued (OQ)* routers. In an OQ router (as shown in Figure 1.2), arriving packets are placed immediately in queues at the output, where they contend with other packets destined to the same output. Since packets from different outputs do not contend with each other, OQ routers minimize delay and behave ideally. An OQ router can have $N$ memories, one memory per output. Each memory must be able to simultaneously accept (write) $N$ new cells (one potentially from every input) and read one cell per time slot. Thus, the memory must have an access rate proportional to $N+1$ times the line rate. While this approximately halves the access time requirement compared to the centralized shared memory router, this approach is still extremely impractical at high speeds.

[8]As noted above, this does not preclude the existence of higher-speed and/or larger-capacity SRAMs, which are not as yet *commodity* parts.

[9]The access rate of a memory is the inverse of the access time, and is the number of times that a memory can be uniquely accessed in a time slot.

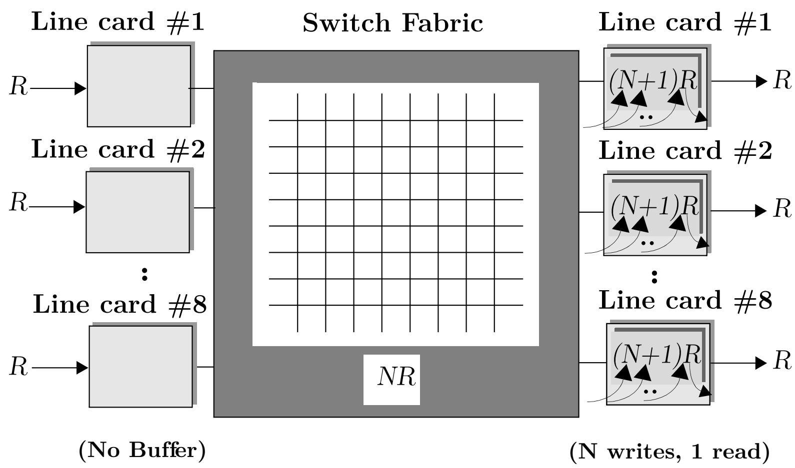



**Figure 1.2:** *The architecture of an output queued router. The memory bandwidth on any individual memory is* $(N+1)R$.

In summary, although OQ routers are ideal and have attractive performance, they are not scalable due to the memory access time limitations.

### 1.3.3 Why is Memory Access Time a Hard Problem?

If memory bandwidth or capacity were a problem, we could simply using more memories in parallel! For example, many Ethernet and Enterprise line cards [4][10] use up to 8 memories in parallel to meet the bandwidth requirements. Similarly, the CRS-1 [5] core router, one of the highest-speed Internet routers available today, uses up to 32 memories to meet its large buffer capacity requirement. While this approach has its limits (having a large number of memories on a line card is unwieldy, and the number of parts that can be used in parallel is limited by the die area of the chips, board size, and cost constraints), it offers a simple way to achieve higher memory capacity and bandwidth.

☞**Observation 1.1.** However, memory access time is a more fundamental limitation. While packets can easily be spread and written over multiple parallel

[10] For simplicity, we will use the terms line card and port interchangeably, and assume that a line card terminates exactly one port. In reality, a line card can receive traffic from many ports.

memories, it is difficult to predict how the data will be read out later. If we want to read from our memory subsystem, say, every 3.2 ns, and all the consecutive packets reside in a DRAM with a much slower access time, then having other parallel memories won't help.

Although the access time[11] will be reduced over time, the rate of improvement is much slower than Moore's Law [6]. Note that even newer DRAMs with fast I/O pins – such as DDR, DDRII, and Rambus DRAMS [7] – have very similar access times. While the I/O pins are faster for transferring large blocks, the access time to a random location in memory is still approximately 50 ns. This is because high-volume DRAMs are designed for the computer industry, which favors capacity over access time. Also, the access time of a DRAM is determined by the physical dimensions of the memory array (and therefore line capacitance), which stays constant from generation to generation.

Commercial DRAM manufacturers have recently developed fast DRAMs (RL-DRAM [8] and FCRAM [9]) for the networking industry. These reduce the physical dimensions of each array by breaking the memory into several banks. This worked well for 10 Gb/s line rates, as it meant these fast DRAMs with 20 ns access times could be used. But this approach has a limited future, for two reasons: (1) as the line rate increases, the memory must split into more and more banks, which leads to an unacceptable overhead per bank,[12] and (2) even though all Ethernet switches and Internet routers have packet buffers, the total number of memory devices needed is a small fraction of the total DRAM market, making it unlikely that commercial DRAM manufacturers will continue to supply them.[13]

☞**Observation 1.2.** Fundamentally, the problem is that networking requires extremely small-size data accesses, equal to the size of the smallest-size 64-byte

[11]The random access time should not be confused with memory latency, which is the time taken to receive data back after it has been issued by the requester.

[12]For this reason, the third-generation parts are planned to have a 20 ns access time, just like the second generation.

[13]At the time of writing, there is only one publicly announced source for future RLDRAM devices, and no manufacturers for future FCRAMs.

packet. By 1997, this was identified as a fundamental upcoming problem.

As line rates increase (usually at the rate of Moore's Law), the time it takes these small packets to arrive grows linearly smaller. In contrast, the random access time of commercial DRAMs has decreased by only 1.1 times every 18 months (slower than Moore's Law) [10]. And so the problem only becomes harder. If we want to design high-speed routers whose capacity can scale, and that can give predictable performance, then we need to alleviate the memory access time problem.

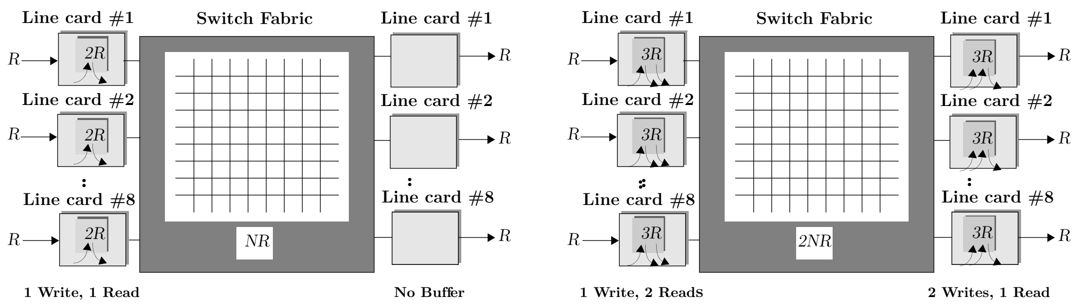
☞**Observation 1.3.** By 2005, routers had to be built to support the next-generation 40 Gb/s line cards. By then, this had become a pressing problem in immediate need of a solution.

### 1.3.4 Historical Solutions to Alleviate the Memory Access Time Problem

Since the first routers were introduced, the capacity of commercial routers[14] has increased by about 2.2 times every 18 months (slightly faster than Moore's Law). By the mid-1990s, router capacity had grown to the point where the centralized shared memory architecture (or output queuing) could no longer be used, and it became popular to use input queueing instead.

In an input queued router, arriving packets are buffered in the arriving line cards as shown in Figure 1.3(a). The line cards were connected to a non-blocking crossbar switch which was configured by a centralized scheduling algorithm. From a practical point of view, input queueing allows the memory to be distributed to each line card, where it can be added incrementally. The switching interconnect needs to carry up to $N$ packets from the inputs to the respective outputs and needs a bandwidth of

[14]We define the capacity of a router to be the sum of the maximum data rates of its line cards, $NR$. For example, we will say that a router with 16 OC192c line cards has a capacity of approximately 160 Gb/s.




**(a)** Input Queued Router

**(b)** CIOQ Router

**Figure 1.3:** *The input-queued and CIOQ router architectures.*

$2NR$. More important, each memory only needs to run at a rate $2R$ (instead of $2NR$), enabling higher-capacity routers to be built.

A router is said to give *100% throughput* if it is able to fully utilize its output links.[15] Theoretical results have showed that with a queueing structure called virtual output queues (VOQs), and a maximum weight matching scheduling algorithm, an input queued router can achieve 100% throughput [11, 12]. However, in an input queued router, it is known that a cell can be held at an input queue even though its output is idle. This can happen for an indefinitely long time. So there are known simple traffic patterns [13] that show that an input queued router cannot behave identically to an OQ router.

Another popular router architecture is the combined input-output queued (CIOQ) router, shown in Figure 1.3(b). A CIOQ router buffers packets twice – once at the input, and again at the output. This router can behave identically[16] to an output queued router if the memory on each line card runs at rate $3R$, the switching interconnect runs at rate $2NR$, and it implements a complex scheduling algorithm [13].

Table 1.1 summarizes some well-known results for the above router architectures. While the results in Table 1.1 may appeal to the router architect, the algorithms

[15] For a formal definition, see Appendix B.

[16] We define this formally later and refer to it as *emulate*.

required by the theoretical results are not practical at high speed because of the complexity of the scheduling algorithms. For these reasons, the theoretical results have not made much difference in the way routers are built. Instead, most routers use a heuristic scheduling algorithm such as iSLIP [14] or WFA [15], and a memory access rate between $2R$ and $3R$.[17] Performance studies are limited to simulations that suggest most of the queueing takes place at the output, potentially allowing it to behave similarly to an output queued router. While this may be a sensible engineering compromise, the resulting system has unpredictable performance, cannot give throughput guarantees, and the worst case is not known.

Other multi-stage and multi-path router architectures have been used. They include Benes, Batcher-banyan [16], Clos, and Hypercube switch fabrics. However, they either require large switching bandwidth, require multiple stages of buffering, or have high communication complexity between the different stages. The implementation complexity and hardware costs involved have restricted their popular use.

In comparison to CIOQ routers, other router architectures tend to hit their hardware limits earlier. This affects their scalability, and so, primarily as a compromise, CIOQ routers have emerged as a common router architecture, even though the performance of practical CIOQ routers is difficult to predict. This is not very satisfactory, given that CIOQ routers make up such a large fraction of the Internet infrastructure. While the architectures described above are an improvement over OQ routers, these routers require that the access rate of the memories is at least as fast as the line rate $R$. Even an input queued router needs a memory that can be accessed at least twice (once to read and once to write a packet) during every time slot. As line rates increase, this may become impossible even with SRAMs.

[17] This refers to a *"speedup"* between one and two. We will avoid the use of the metric commonly called "speedup" in our discussion of memory. The term speedup is used differently by different authors; there is no accepted standard definition. Instead, we will compare different router memory subsystems based on their memory access rate and memory bandwidth. We will, however, define "speedup" and use it as appropriate for other operations related to memory, *e.g.*, when we refer to the speed at which switching interconnects operate, updates are done, or copies are made.

**Table 1.1:** *Comparison of router architectures.*

| Router Type | # mem | Mem. Access Rate.[18] | Total mem. BW | Interconnect BW | Comment |
|---|---|---|---|---|---|
| Centralized Shared Memory | 1 | $\equiv \Theta(2NR)$ | $2NR$ | $2NR$ | Ideal Router (Shared Memory) |
| Output Queued | $N$ | $\equiv \Theta(N+1)R$ | $N(N+1)R$ | $NR$ | Ideal Router (Distributed Memory) |
| Input Queued | $N$ | $\equiv \Theta(2R)$ | $2NR$ | $NR$ | Cannot emulate an ideal router |
| CIOQ | $2N$ | $\equiv \Theta(3R)$ | $6NR$ | $2NR$ | Can emulate an ideal router [13] |

✎**Example 1.3.** For example, with the advent of the new 100Gb/s line cards, even an input queued router would need an access time of 2.56 ns to buffer 64-byte packets. This is already out of reach of most commodity SRAMs today.

## 1.4 Mandating Deterministic Guarantees for High-Speed Routers

Our goal is to build high-speed routers than can give performance guarantees. As we saw, building a slow-speed router with performance guarantees is easy. But the problem is hard when designing high-speed routers. In fact, today, there are no high-speed routers with bandwidths greater than 10 Gb/s that give deterministic guarantees. There are typically two types of performance guarantees: statistical (the most common example being 100% throughput) and deterministic (the most common examples being work-conservation and delay guarantees).

Even statistical guarantees are hard to achieve. In fact, no commercial high-speed router today can guarantee 100% throughput. If we buy a router with less than 100% throughput, we do not know what its true capacity is. This makes it particularly hard for network operators to plan a network and predict network performance. In normal times (when the utilization of the network is quite low), this does not matter, but

[18] The $\Theta(\cdot)$ notation is used to denote an asymptotically tight bound in the analysis of algorithms. In the context of this thesis it denotes that the actual value is within a constant multiple of the value given.

during times of congestion (usually due to link failures), when network performance really matters, this is unacceptable. We want to design routers that give 100% throughput.

However, we will specifically demand that our high-speed routers give deterministic performance guarantees on individual packets, since this will enable us to meet the goals outlined in Section 1.2. Also, if routers give deterministic guarantees, they automatically give statistical guarantees. We are interested in two classes of deterministic guarantees, both of which have relevance for the end-user.

### 1.4.1 Work Conservation to Minimize Average Packet Delay

ℵ**Definition 1.1. Work-conserving:** A router is said to be work-conserving if an output will always serve a packet when a packet is destined for it in the system.

We want our routers to be work-conserving. If a router is work-conserving, then it has 100% throughput, because the outputs cannot carry a higher workload. It also minimizes the expected packet delay, since, on average, packets leave earlier in a work-conserving router than in any other router.[19] To achieve work conservation, we will use first come first served (FCFS) as the main service policy in our analysis of routers. This means that packets to a given output depart in the order of their arrival. However, the techniques that we use to analyze router architectures allow us to extend our results to any work-conserving router if the departure time[20] is known on arrival.

### 1.4.2 Delay Guarantees to Bound Worst-Case Packet Delay

A standard way that routers can provide delay guarantees is to introduce harder constraints by shaping or streamlining the arrival traffic; for example, the leaky-bucket

[19] Strictly speaking, this is only true if all packets have the same size. When packets have different sizes, a router could re-order packets based on increasing size and service the smallest packets first to minimize average delay. This is unimportant because the order of departure of packets cannot be arbitrarily changed by a router.

[20] The departure time for an arriving cell for an FCFS queuing service policy can be easily calculated. It is the first time that the server (output) is free and able to send the newly arriving cell.

constraint.[21] The constrained arriving streams are then serviced by a scheduler (such as, say, a weighted fair queueing [17] scheduler) that provides service guarantees. There are two ways to achieve this —

**Maintaining separate FIFO queues:** Every arrival stream is constrained and stored in a separate queue. A scheduler works in round robin order by looking at these queues and taking the head-of-line packets from the queues. The packets are serviced in the order of their departure time[22] and leave the output in that sorted order. Figure 1.4(a) shows an example of the input traffic, $A(t)$, constrained and split into its three constituent streams, $A(1)$, $A(2)$, and $A(3)$, and stored in their three respective queues. The three queues are serviced by a scheduler in the ratio $2:1:1$. From well-known results on weighted fair queueing [18], it is then possible to define bounds on the worst-case delay faced by a packet transiting the router.

**Maintaining a single logical queue:** Another way to do this is to move the sorting operation into a "logical" queue known as the push-in first-out (PIFO) queue. A PIFO queue has a number of key features:

1. Arriving packets are "pushed-in" to an arbitrary location in the departure queue, based on their finishing time.
2. Packets depart from the head of line (HoL).
3. Once the packet is inserted, the relative ordering between packets in the queue does not change.

Therefore, once a packet arrives to a PIFO queue, another packet that arrives later may be pushed in before or after it, but they never actually switch places. This is the characteristic that defines a PIFO queue. PIFO includes a number of queueing policies, including weighted fair queueing [17] and its variants such as GPS [18], Virtual Clock [19], and DRR [20]. It also includes strict priority queueing.[23]

[21] This is defined in Section 4.3.1.

[22] Also referred to in literature as finishing time.

[23] Note that some QoS scheduling algorithms such as $WF^2Q$ [21] do not use PIFO queuing.

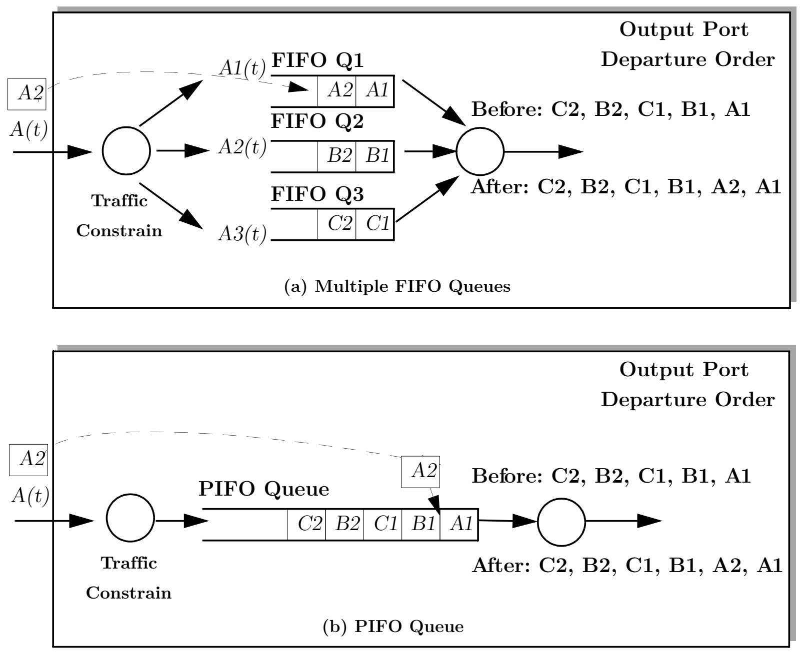



**Figure 1.4:** *Achieving delay guarantees in a router. (a) Multiple FIFO queues, (b) A "logical" push in first out queue.*

✎**Example 1.4.** As Figure 1.4(b) shows, the PIFO queue always maintains packets in a sorted order. When packets come in, they are inserted in the sorted order of their departure times. In Figure 1.4(a), arriving cell $A2$ is inserted at the tail of FIFO $Q1$ and awaits departure after cell $A1$ but before cell $B1$. In Figure 1.4(b), it is inserted directly into the sorted PIFO queue after cell $A1$ but before cell $B1$.

Depending on the architecture being used, we choose either the set of logical FIFO queues, or the sorted PIFO queue to make our analysis easier.

## 1.5 Approach

### 1.5.1 Analyze Many Different Routers that Have Parallelism

While it is difficult to predict the course of technology and the growth of the Internet, it seems certain that in the years ahead routers will be required with: (1) increased switching capacity, (2) support for higher line rates, and (3) support for differentiated qualities of service. Each of these three requirements presents unique challenges. For example, higher-capacity routers may require new architectures and new ways of thinking about router design; higher line rates will probably exceed the capabilities of commercially available memories (in some cases this transition has already happened), making it impractical to buffer packets as they arrive; and the need for differentiated qualities of service will mandate deterministic performance guarantees comparable to the ideal router.

☞**Observation 1.4.** There will probably be no single solution that will meet all needs. Each router architecture will have unique tradeoffs. Very-high-speed interconnect technologies may make certain new router architectures possible, or possibly simplify existing ones. Memory access speeds may continue to lag router requirements to such a degree that architectures with memories with much slower access rates may become mandatory. If memory capacity lags router buffer sizing requirements, architectures with shared memory will become important.

We won't concern ourselves with whether a particular technique is currently implementable, nor will we prefer one router architecture over another, since the cost and technology tradeoffs will be driven by future technology. We will also be agnostic to trends in future technology. The only assumption we will make is that memory access time will continue to be a bottleneck in one form or another. In order to alleviate this bottleneck, we will consider router architectures that have parallelism.

Our approach is to analyze different types of parallel router architectures, work within the technology constraints on them, and ask what it will take for these routers to give deterministic performance guarantees. We will not attempt to analyze all router architectures exhaustively; in fact, future technologies may lead to completely novel architectures. Instead, we will focus on architectural and algorithmic solutions that exploit the characteristic of memory requirements on routers, so that they can be applied to a broad class of router architectures. We will focus on two key ideas: *load balancing* and *memory caching*,[24] that alleviate the load on memory.

### 1.5.2 Load-Balancing Algorithms

A load balancing algorithm is motivated by the following idea —

> **Idea.** *"We could intelligently distribute load among parallel memories.[a] such that no memory needs to be accessed faster than the rate that it can support".*
>
> [a]Ideally, there will be no limit as to how slow these parallel memories can operate.

In order to realize the above idea, a load balancing algorithm usually needs to be aware *a priori* of the time when a memory will be accessed. We introduce a new mathematical technique called *constraint sets* which will help us analyze load balancing algorithms for routers. We have found that memory accesses in routers are fundamentally clashes between writers (*e.g.*, arriving packets that need to be buffered) and readers (*e.g.*, packets that need to depart at a particular future time) that need to simultaneously access a memory that cannot keep up with both requirements at the same time. Constraint sets are a formal method to mathematically capture these constraints. While the constraint set technique is agnostic to the specifics of the router architecture, the actual size and definition of these sets vary from router to router. This helps us separately analyze the load balancing algorithm constraints for each individual router, and:

[24]These are standard techniques in use in the fields of distributed systems and computer architecture.

1. identify the memory access time, memory, and switching bandwidth required for a specific router architecture to be work-conserving,
2. define a switching algorithm for the router,
3. identify the design tradeoffs between memory access time, memory bandwidth, switch bandwidth, and switching algorithms, and
4. perhaps surprisingly, exactly characterize the cost of supporting delay guarantees in a router.

### 1.5.3 Caching Algorithms

The second approach we use is building a memory hierarchy, and is motivated by the following idea:

> **Idea.** *"We can use a fast on-chip cache memory that can be accessed at high rates, and a slow[a] off-chip main memory that can be accessed at low rates".*
>
> [a]The slow memory can potentially be off-chip and have a large memory capacity.

The idea is similar to that used in most processors and computing systems. Packets or data that are likely to be written or read soon are held in fast cache memory, while the rest of the data is held in slower main memory. If we have prior knowledge about how the memory will be accessed (based on knowledge of the data structures required to be maintained by routers), then we can (1) write data in larger blocks at a slower access rate, and (2) pre-fetch large blocks of data from main memory at slower access rates and keep it ready before it needs to be read.

**Observation 1.5.** But unlike a computer system, where is acceptable for a cache to have a miss rate, such behavior is unacceptable in networking, since it can lead to loss of throughput and packet drops.

Therefore, our cache design should be able to guarantee a 100% hit rate under all conditions. We will see that, within bounds, such caching algorithms exist for certain applications that are widely used on most routers. As an example, packets are maintained in queues; the good thing is that we know which data will be needed soon – it's sitting at the head of the queue. We will borrow some existing mathematical techniques to propose and analyze caching algorithms, such as *difference equations* and *Lyapunov functions*, which have been used extensively in queueing analysis.

### 1.5.4 Use Emulation to Mandate that Our Routers Behave Like Ideal Routers

We want to ensure that any router that we design will give us deterministic performance guarantees. We therefore compare our router to the ideal OQ router. To do this, we assume that there exists an OQ router, called the *"shadow OQ router"*, with the same number of input and output ports as our router. The ports on the shadow OQ router receive identical input traffic and operate at the same line rate as our router. This is shown in Figure 1.5. Consider any cell that arrives to our router. We denote $DT$, the departure time of that cell from the shadow OQ router. We say that our router *mimics* [13, 22, 23] the ideal router if under identical inputs, the cells also depart our router at time $DT$. If we want work conservation, we will compare our router to a shadow OQ router performing an FCFS policy (called an FCFS-OQ router). If we want delay guarantees, our comparison will be with a PIFO-OQ shadow router.

Note that the router we compare may have some fixed propagation delays that we wish to ignore (perhaps the data transits a slower interconnect, or is buffered in a slower memory, *etc.*). In particular, we are only interested in knowing whether our cell faces a *relative queueing delay*, *i.e.*, an increased queueing delay (if any) relative to the delay it receives in the shadow OQ router. We are now ready to formally compare our router with an ideal router.

ℵ**Definition 1.2. Emulate:** A router is said to emulate an ideal shadow OQ router if departing cells have no relative queueing delay, compared to the shadow OQ router.

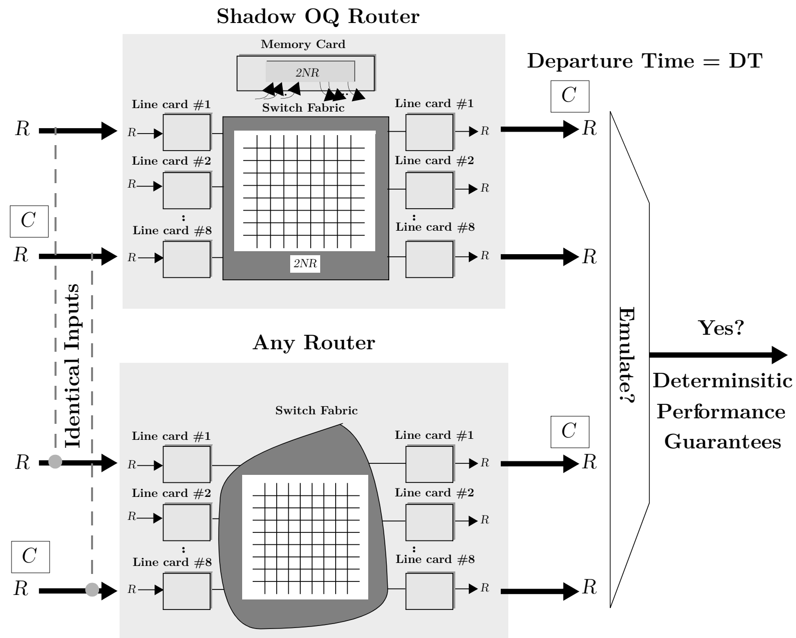



**Figure 1.5:** *Emulating an ideal output queued router.*

Note that the definition of emulation makes no assumptions on the packet sizes, the bursty nature of packet arrivals, or the destination characteristics of the packets, and requires that the cells face no relative delay irrespective of the arrival pattern. An important consequence of this definition is as follows — the ideal output queued router is safe from adversarial traffic patterns (such as can be created by a hacker or virus) that exploit potential performance bottlenecks in a router. If our router can emulate an OQ router, then our router is also safe from adversarial performance attacks!

### 1.5.5 Use Memory as a Basis for Comparison

Throughout this thesis, we will use memory for comparison. We will look at architectures that alleviate memory access time requirements. And we will see that doing so requires tradeoffs on other components; for example, we may need more total memory bandwidth, more interconnect bandwidth, an on-chip cache, *etc.* We believe that memory serves as a good metric, for three reasons:

1. Routers are, and will continue to be, limited by the access rate and memory bandwidth of commercially available memories in the foreseeable future. All else being equal, a router with smaller overall access rate and memory bandwidth requirements can have a higher capacity.
2. Memory components alone contribute approximately 20% of the cost of materials, on average. In certain high-speed market segments, almost a third of the cost comes from memory [24]. Cisco Systems, the leading Internet router vendor, alone spends roughly $800M p.a. on memory components. Routers with lower memory bandwidth will need less memory, and will be more cost-efficient.
3. A router with higher memory bandwidth will, in general, consume more power. Routers are frequently limited by the power that they consume (because they are backed up by batteries) and dissipate (because they must use forced-air cooling). The total memory bandwidth indicates the total bandwidth of the high-speed I/Os that connect the memories to control logic. As an example, high-speed memory I/O contributes to approximately 33% of the overall power on Ethernet switches and Enterprise routers [25].

Similar to computing systems, memories are (and in the foreseeable future will continue to be) the main bottleneck to scaling the capacity and meeting the cost and power budgets of high-speed routers.

## 1.6 Scope of Thesis

This thesis is about the router data-plane, which processes every incoming packet, and needs to scale with increasing line rates.[25] The data-plane needs to perform many *computational tasks*, such as parsing arriving packets, checking their integrity, extracting and manipulating fields in the packet, and adding or deleting protocol headers, as shown in Figure 1.6. In addition it performs several *algorithmic tasks*, such as forwarding lookups to determine the correct packet destination, packet and flow

[25] We do not consider problems in the router control plane, which usually runs much slower than line rate and is involved with management, configuration, and maintenance of the router and network connectivity.

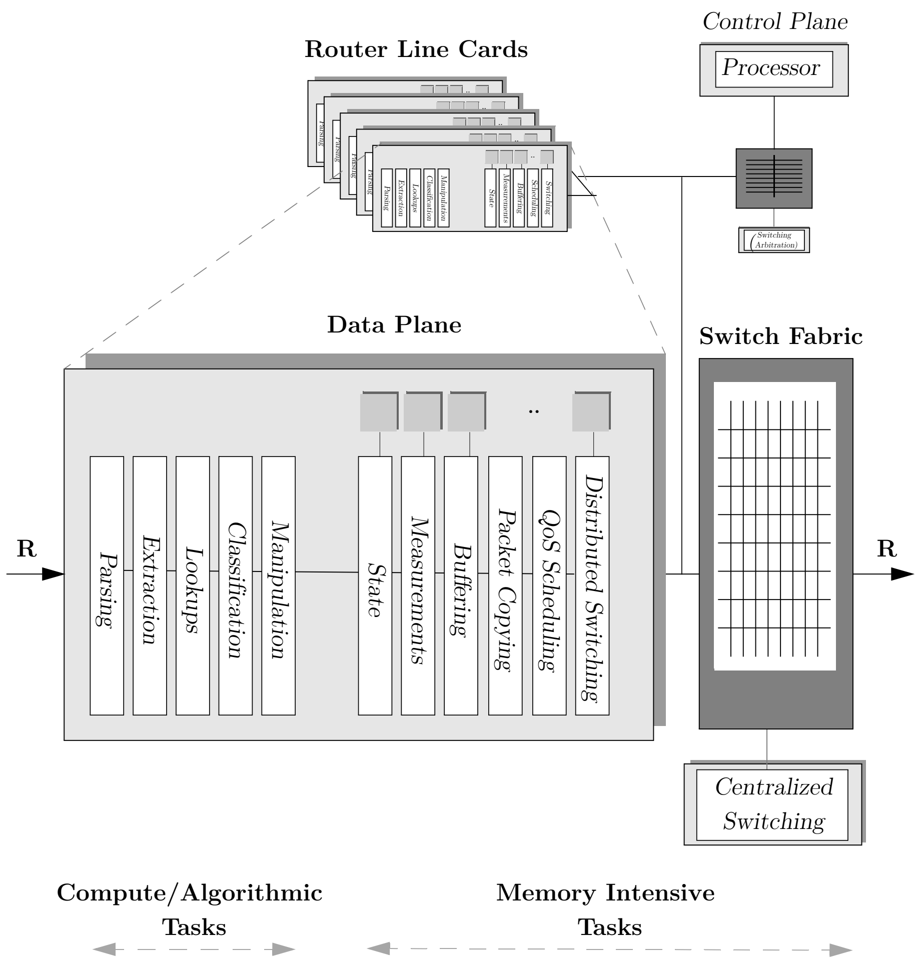



**Figure 1.6:** *The data-plane of an Internet router.*

classification (*e.g.*, for security, billing and other applications), deep packet inspection (*e.g.*, for virus filtering), *etc.* A number of algorithms and specialized hardware (such as CAMs, hardware assisted classifiers, complex parsers, *etc.*) have been proposed to make these tasks efficient. We do not address any of these computational or algorithmic tasks in this thesis. Instead, we focus on the tasks in the router data-path for which memory is a bottleneck, as shown in Figure 1.6.

At a minimum, a router needs to buffer and hold packets in memory during times of congestion. It needs to switch packets across a switching fabric and move these packets to their correct output line cards (where it may again be buffered temporarily). As shown in the figure, switching can be performed centrally, or by each line card in a distributed manner. In addition, routers perform measurements and keep statistics in memory, for purposes of billing, metering, and policing traffic. Statistics are also useful for planning, operating, and managing a network. Routers perform packet scheduling as described in Section 1.4.2 to provide various qualities of service. In some cases, specialized application-aware routers maintain state for flows to perform complex tasks such as application proxies, network address translation, *etc.*, as well as maintain policing, shaping, and metering information. In some cases, routers need to support packet multicasting, where packets may need to be copied multiple times. As we shall see, each of the above tasks requires high memory access rates. This thesis studies the nature of the data structures needed by the above tasks, and takes advantage of their associated memory accesses to alleviate their memory bottlenecks.

## 1.7 Organization

In Section 1.3.1, we saw that an ideal output queued router requires a memory access rate that is proportional to the product of the number of ports $N$ and the line rate of each port $R$. In the remainder of the thesis, we will look at ways to reduce the access rate of the memory on high-speed routers. This is done by trading off memory access time with memory bandwidth, capacity or a cache as summarized in Table 1.2. We divide the rest of this thesis into two parts:

### Part I: Load Balancing and Caching Algorithms for Router Architecture

In the first part of the thesis, we will consider how we can scale the performance of the router as a whole. Thus, in Chapter 2, we first define a new class of router architectures, called *"single-buffered"* routers that are of particular interest. We also answer the question posed in the conversation in the Introduction, since it sheds light on the analysis of all single-buffered routers. Then, in each chapter, we ask a question about the technology and the architectural constraints that the memory on a router

**Table 1.2:** *Organization of thesis.*

| Chpt. | Analysis | Memory Tradeoff |
|---|---|---|
| 1 | Output Queued Router | Ideal router |
| 2 | Parallel Shared Mem. Router | More Memory bandwidth |
| 3-(a) | Distributed Shared Mem. Router | More Memory bandwidth |
| 3-(b) | Parallel Distributed Shared Mem. Router | More Memory bandwidth |
| 4 | CIOQ Router | More Memory bandwidth |
| 5 | Buffered CIOQ Router | Line rate cache |
| 6-(a) | Parallel Packet Switch | More Memory bandwidth |
| 6-(b) | Buffered Parallel Packet Switch | Line rate cache |
| 7-(a) | Packet Buffer Cache | Line rate cache |
| 7-(b) | Pipelined Packet Buffer Cache | Line rate pipelined cache |
| 8 | Packet Scheduler Cache | Line rate cache |
| 9 | Statistics Counter Cache | Line rate cache |
| 10 | Parallel Packet State | More memory bandwidth, capacity |
| App. J | Parallel Packet Copy | More memory bandwidth |

might face, as it pertains to its overall architecture:

**Chpt. 2: What if the memories must be shared?** Memory capacity could become a problem, and shared memory architectures could become mandatory. In Chapter 2, we look at a parallel shared memory router architecture that allows the access rate on each memory to be arbitrarily small, while allowing all of the memories to be shared among all ports of the router.

**Chpt. 3-(a): What if memories must be distributed?** Memory power could become a problem, forcing routers to distribute memories across several line cards to achieve better cooling, *etc.* Also, product constraints sometimes require that additional router bandwidth (and the extra memory it requires) be added incrementally. This mandates the use of distributed memories. In Chapter 3, we analyze a distributed shared memory architecture.

**Chpt. 3-(b): What if the memories must be limited?** If memory access time lags router system requirements, then a large number of memories that operate in parallel are needed. If there are too many memories, it may become infeasible to manage, and this may become a bottleneck. In Chapter 3, we will consider a parallel distributed shared memory architecture that limits the number of memories that need to be managed, and alleviates this problem.

**Chpt. 4: What if memories must be localized?** Many routers have line cards that have different port speeds and features, and support different protocols. Because the line cards have different processing and memory capabilities, such a system cannot use distributed processing or distributed memory. In Chapter 4, we look at a fairly common CIOQ architecture that has localized buffering.

**Chpt. 5: What if on-chip memory capacity is plentiful?** New memory technologies such as embedded DRAM [26] allow the storage of large amounts of on-chip memory. In Chapter 5, we will consider the addition of on-chip cache memory to the commonly used crossbar switching fabric of a CIOQ router. We will show that the resulting buffered CIOQ router can greatly simplify and make practical the CIOQ architecture introduced in Chapter 4.

**Chpt. 6-(a,b): What if memories run really slow?** If line rates become much faster than the memory access time, we will need to look at massively parallel memory architectures. In Chapter 6, we analyze the parallel packet switch (PPS). The PPS is a high-speed router that can be built from a network of slower-speed routers. We consider two variants of the PPS — (1) an unbuffered version, and (2) a buffered PPS (which has a small cache).

## Part II: Load Balancing and Caching Algorithms for Router Line Cards

In the second part of the thesis, we consider how we can scale the performance of the memory-intensive tasks on a router line card that were described in Figure 1.6. Again, we consider several questions pertaining to the memory usage of these data-path tasks.

**Chpt. 7-(a): What if the memory departure time cannot be predicted?** The time at which a packet needs to depart from memory is decided by a scheduler. Architectural or hardware design constraints may prevent the buffer from knowing the time at which a packet will depart from memory. This prevents us from using load balancing algorithms that require knowledge of packet departure times. In Chapter 7, we will look at a memory caching algorithm that can overcome this limitation to build high-speed packet buffers. We will show how this caching algorithm can be readily extended for use in other router data-plane tasks such as packet scheduling, page management, *etc.*

**Chpt. 7-(b): What if we can tolerate large memory latency?** Many high-performance routers use deep pipelines to process packets in tens or even hundreds of consecutive stages. These designs can tolerate a large (bit fixed) pipeline latency between when the data is requested and when it is delivered. Whether this is acceptable will depend on the system. In Chapter 7, we will consider optimizations that reduce the size of the cache (for building high-speed packet buffers) to exploit this.

**Chpt. 8: What if we can piggyback on other memory data structures?** Routers perform packet scheduling in order to provide varied qualities of service for different packet flows. This requires the scheduler to be aware of the location and size of every packet in the system, and requires a scheduler database whose memory access rate is as fast as the buffer from which it schedules packets. In Chapter 8, we describe a mechanism to encode, piggyback, and cache the scheduler database along with the packet buffer cache implementation described in Chapter 7. This eliminates the need for a separate scheduler database.

**Chpt. 9: What if we can exploit the memory access pattern?** Many data-path applications maintain statistics counters for measurement, billing, metering, security, and network management. We can exploit the way counters are maintained and build a caching hierarchy that alleviates the need for high-access-rate memory. This is described in Chapter 9.

**Chpt. 10: What if we can't exploit the memory access pattern?** Networking applications such as Netflow, NAT, TCP offload, policing, shaping, scheduling, state maintenance, *etc.*, maintain flow databases. A facet of these databases is that their memory accesses are completely random, and the memory address that they access is dependent completely on the arriving packet, whose pattern of arrival cannot be predicted. These applications need high access rates, but have minimal storage and bandwidth requirements. In many cases, the available memories (especially DRAMs) provide an order of magnitude more capacity than we need, and have a large number of independently accessible memory banks.[26] High-speed serial interconnect technologies [27] have enabled large increases in bandwidth. At the time of writing, a serial interconnect can provide 5-10 times more bandwidth per pin compared to the commonly used parallel interconnects used by commodity memory technology. In Chapter 10, we will look at how we can exploit increased memory capacity and bandwidth to achieve higher memory access rates to benefit the above applications.

**App. J: What if we need large memory bandwidth?** There are a number of new real-time applications such as Telepresence [28], videoconferencing, multi-player games, IPTV [29], *etc.*, that benefit from multicasting. While we cannot predict the usage and deployment of the Internet, it is likely that routers will be called upon to switch multicast packets passing over very-high-speed lines with a guaranteed quality of service. Should this be the case, routers will have to support the extremely large memory bandwidths needed for multicasting. This is because an arriving packet can be destined to multiple destinations, and the router may have to make multiple copies of the packet. We briefly look at methods to efficiently copy packets in Appendix J.[27]

## 1.8 Application of Techniques

We now outline how the two key architectural techniques — load balancing and caching — are applied in this thesis.

[26] It is not uncommon today to have memories with 32 to 64 banks.

[27] Compared to the main body of work in this thesis, our results on multicast require a large memory bandwidth and cache size. They are impractical to implement in the general case (unless certain tradeoffs are made) for providing deterministic performance guarantees. And so, these results are included in the Appendix.

**Table 1.3:** *Application of techniques.*

| Chpt. | Analysis | Load-Balancing Algorithm | Caching Algorithm |
|---|---|---|---|
| 1 | Output Queued Router | None | None |
| 2 | Parallel Shared Memory Router | Memories | None |
| 3-(a) | Distributed Shared Memory Router | Line cards | None |
| 3-(b) | Parallel Distributed Shared Memory Router | Line cards, Memories | None |
| 4 | CIOQ Router | Time | None |
| 5 | Buffered CIOQ Router | Time | VOQ Heads |
| 6-(a) | Parallel Packet Switch | Switches | None |
| 6-(b) | Buffered Parallel Packet Switch | Switches | Arriving, Departing Packets |
| 7-(a) | Packet Buffer Cache | Memories | Queue Head and Tails |
| 7-(b) | Pipelined Packet Buffer Cache | Memories | Queue Head and Tails |
| 8 | Packet Scheduler Cache | Packets | Packet Lengths |
| 9 | Statistics Counter Cache | None | Counter Updates |
| 10 | Parallel Packet State | Memory banks | None |
| App. J | Parallel Packet Copy | Memory banks | None |

Depending on the requirement, we load-balance packets over unique memories, memory banks, line cards, and switches. If there is only one path (or memory) that a packet can access, then the load balancing algorithms distribute their load over time. A special case occurs for packet scheduling, where load balancing of packet lengths is done over packets written to memory. These are summarized in column two of Table 1.3. The advantage with load balancing algorithms is that there is no theoretical limit to how slow the memories can operate. The caveat is that these algorithms usually need more memory bandwidth (than the theoretical minimum), and the algorithms require complete knowledge of all memory accesses.

In contrast, caching algorithms appear simpler to implement, but they need a cache that runs at the line rate ($\equiv \Theta(R)$). We demonstrate caching architectures and algorithms for a number of data-path tasks. The cache may hold heads and tails of queues, arriving or departing packets, packet lengths, or counters, depending on the data-path task, as shown in column three of Table 1.3.

## 1.9 Industry Impact

Throughout this thesis, we present practical examples and details of the current widespread use of these techniques in industry. The current technology constraints have allowed caching algorithms to find wide acceptance; however, we do not mandate one approach over another, as the constraints may change from year to year due to product requirements and available technology.

☞**Observation 1.6.** A problem solved correctly is analogous to a *meme*.[28] A good technology meme solves a *fundamental* (not ephemeral) problem, lays the foundation for solving related problems, spreads rapidly, and can lead to the application of complementary technologies.

### 1.9.1 Consequences

At the time of writing, we have demonstrated, applied, and in some cases modified the techniques presented in this thesis to scale the performance of various networking applications, including packet buffering, measurements, scheduling, VOQ buffering, page allocation, virtual memory management, Netflow, and state-based applications to 40 Gb/s and beyond, using commodity DRAM-based memory technologies. Surprisingly, these techniques (in particular caching), because they use memory in a monolithic manner across all memories, have also resulted in increased router memory reliability, simplified router memory redundancy (*e.g.*, they have enabled $N+1$ memory redundancy similar to RAID-3 and RAID-5 protection for disks [30]), and enabled hot-swappable recovery from router memory failures.

They have helped reduce worst-case memory power (by ∼25-50% [31]), and are currently being modified for use in automatically (without manual intervention) reducing the average-case memory and I/O power consumption in routers. The power reduction capabilities of caches have considerable engineering, economic, and environmental benefits.

They have also paved the way for the use of complementary high-speed interconnect and memory serialization technologies [32].[29] Unforeseen by us, new markets (*e.g.*,

[28] "An idea that spreads" — Richard Dawkins coined the term meme in 1976, in his popular book, *The Selfish Gene*.

[29] Caches hide memory latency (*i.e.*, the time it takes to fetch data) of these beneficial serial interconnect technologies, enabling them to be used for networking.

high-performance computing and storage networks) can also benefit from caching. These techniques have made high-speed routers more affordable by reducing yearly memory costs at Cisco Systems by $>$ \$149M [33] ($\sim$50% of memory cost), reducing ASIC and I/O pin counts, and halving the physical area required to build high-speed routers.

### 1.9.2 Current Usage

We estimate that more than 6 M instances of the technology (on over 25 unique product instances) will be made available annually, as Cisco Systems proliferates its next generation of 40 Gb/s high-speed Ethernet switches and Enterprise routers; and up to $\sim$80%[30] of all high-speed Ethernet switches and Enterprise routers in the Internet will use one or more instances of these technologies. They are currently also being designed into the next generation of 100 Gb/s line cards (for high-volume Ethernet, storage, and data center applications). In the next phase of deployment, we also expect to cater to Internet core routers.

Finally, in the concluding chapter we describe some of the optimizations, implementation challenges, and potential caveats in scaling these techniques across different segments of the router market. We will also describe the limitations of some of our algorithms, and present some open problems that are still in need of solution.

## 1.10 Primary Consequences

The primary consequences of our work are —

1. Routers are no longer dependent on memory speeds to achieve high performance.
2. Routers can better provide strict performance guarantees for critical future applications (*e.g.*, remote surgery, supercomputing, distributed orchestras).
3. The router applications that we analyze are safe from malicious memory performance attacks. Their memory performance can never be compromised, either now, and provably, *ever* in future.

[30]Based on Cisco's current proliferation in the networking industry.

## 1.11 Summary of Results

This thesis lays down a foundation for solving the memory performance problem in high-speed routers. It describes robust load balancing algorithms (which can emulate the performance of an ideal router) and robust caching algorithms (which can achieve 100% hit rates). It brings under a common umbrella many well-known high-speed router architectures, and introduces a general principle called "constraint sets", which unifies several results on router architectures. It also helps us derive a number of new results, and more important, vastly simplifies our understanding of high-speed routers.

The following are the 14 main results of this thesis. In what follows, $N$ and $R$ denote, as usual, the number of ports and the line rate of a router.

**Robust Load Balancing Algorithms:** We derive load balancing algorithms that use memories in parallel, for a number of router architectures. We derive upper bounds on the total memory bandwidth (as well as the switching bandwidth where appropriate), which is within a constant, $S = \Theta(1)$ of the theoretical minimum for the given architecture.[31] These include: (1) $S = 3$ for the FCFS parallel shared memory (PSM) router and $S = 4$ for a PSM router that supports qualities of service, (2) Two variants of load balancing algorithms with $S = 3, 4$ for the FCFS distributed shared memory (DSM) router, and two variants with $S = 4, 6$ for a DSM router which supports qualities of service, (3) $S = 6, 8$ for the FCFS parallel distributed shared memory (PDSM) router and $S = 12, 18$ for a PDSM router which supports multiple qualities of service (both with a number of variations), (4) $S = 6 + \epsilon$ for the FCFS combined input output queued (CIOQ) router, and (5) $S = 4$ for the FCFS parallel packet switch (PPS), and $S = 6$ for a PPS that supports qualities of service (with both centralized and distributed implementations). We also develop load balancing algorithms for two very generic applications that are widely used in systems including routers. These include: (6) An algorithm that speeds up the random cycle time to perform memory updates by a factor $\Theta(\sqrt{h})$ (where $h$ is the number of parallel memory

[31] Their total memory bandwidth is given by $SNR$. See Table 2.1 for details.

banks, and used for state maintenance by routers), and (7) An algorithm that can create $m$ copies of data (used for supporting multicasting by routers) by using memories that run only at access rate $\Theta(\sqrt{m})$.

**Robust Caching Algorithms:** We describe several robust caching algorithms that ensure, 100% of the time, that the cache never overflows or misses, along with related cache replenishment and cache replacement policies. The caches are of size (8) $\Theta(N^2)$ for an FCFS buffered crossbar, (9) two variants with cache size $\Theta(N^2)$ and $\Theta(N^3)$ for buffered crossbars that support qualities of service, (10) $\Theta(Nk)$ for an FCFS parallel packet switch (where $k$ is degree of parallelism, *i.e.*, the number of center stage switches), (11) $\Theta(Qb \log Q)$ for packet buffers with $Q$ queues (where $b$ is the width of the memory access), (12) $\Theta(Q \log \frac{Qb}{x})$ for pipelined buffers, which can tolerate a latency of $x$ time slots, (13) $\Theta(\log \log N)$ for measurement infrastructure with $N$ counters, and (14) $\Theta(Qb \log Q + RL)\frac{\log_2 P_{max}}{P_{min}}$ for packet schedulers with $Q$ linked lists operating at rate $R$, and a communication latency of $L$ time slots (where $P_{max}$ and $P_{min}$ are the maximum and minimum packet sizes respectively).

## Summary

1. High-speed networks (*e.g.*, the Internet backbone) suffer from a well-known problem: packets arrive on high-speed routers at rates much faster than commodity memory can support.
2. By 1997, this was identified as a fundamental approaching problem. As link rates increase (usually at the rate of Moore's Law), the performance gap widens and the problem becomes worse. For example, on a 100Gb/s link, packets arrive ten times faster than memory rates.
3. If we are unable to bridge this performance gap, then our routers cannot give any guarantees on performance. In this case, our routers (1) cannot reliably support links >10Gb/s, (2) cannot support the needs of real-time applications, and (3) are susceptible to hackers or viruses that can easily exploit the memory performance gap and bring down the Internet.
4. The goal of this thesis is to build routers that can provide deterministic performance guarantees. We are interested in two kinds of guarantees — (1) work conservation (which

ensures that our routers do not idle when a packet is destined for it), and (2) delay guarantees (*i.e.*, bounds on the time it takes for a packet to transit a router).

5. This thesis attempts to provide fundamental (not ephemeral) solutions to the memory performance gap in routers.
6. The approach taken to alleviate the memory performance gap is to use parallelism. We analyze many router architectures, and various data-path applications on routers. We do not attempt to be exhaustive; rather, we assume that memory performance will be a constraint, and we attempt to solve some of the fundamental memory constraints that routers might face.
7. This thesis develops two classes of solutions, both of which use parallelism — (1) load balancing algorithms that attempt to distribute the load in a "perfect" manner among slower memories, and (2) caching algorithms that use a small, high-speed memory that can be accessed at very high rates while most of the data resides in slower memories. The trick is to guarantee, with no exceptions whatever, that data is available in the cache 100% of the time.
8. We analyze the performance of the above algorithms by comparing them to an ideal (output queued) router. We call this *emulation*.
9. This thesis only pertains to memory-intensive data-path tasks in a router. It does not attempt to solve the algorithmic and computationally intensive tasks on a router.
10. Each chapter in the thesis solves a different memory-related problem. Part I of the thesis pertains to router architectures, while Part II pertains to data-path tasks and applications on a router.
11. The thesis lays down the theoretical foundations for solving the memory performance problem, presents various solutions, gives examples of practical implementations, and describes the present widespread adoption of these solutions in industry.
12. As a consequence — (1) routers are no longer dependent on memory speeds to achieve high performance, (2) routers can better provide strict performance guarantees for critical future applications (e.g., remote surgery, supercomputing, distributed orchestras), and (3) the router applications that we analyze are safe from malicious memory performance attacks. Their memory performance can never be compromised either now or, provably, *ever* in future.
13. It presents solutions to 14 different memory performance problems in high-speed routers.