ABSTRACT

With the ability to integrate a large number of cores on a single chip, research into on-chip networks to facilitate communication becomes increasingly important. On-chip networks seek to provide a scalable and high-bandwidth communication substrate for multi-core and many-core architectures. High bandwidth and low latency within the on-chip network must be achieved while fitting within tight area and power budgets. In this lecture, we examine various fundamental aspects of on-chip network design and provide the reader with an overview of the current state-of-the-art research in this field.

KEYWORDS

interconnection networks, topology, routing, flow control, computer architecture, multiprocessor system on chip

Acknowledgments

First we would like to thank Mark Hill for the feedback and support he has provided from the inception of this project through the final product. Additionally, we would like to thank Michael Morgan for the opportunity to contribute to this lecture series. Many thanks to Dennis Abts and Timothy Pinkston for their detailed comments that were invaluable in improving this manuscript. Thanks to Konstantinos Aisopos, Emmanouil Koukoumidis, Abhishek Bhattarcharjee of Princeton University and Danyao Wang of the University of Toronto for proofreading our early drafts. James Balfour, Amit Kumar and Partha Kundu provided helpful insight on the microarchitectural implementation section.

Natalie Enright Jerger and Li-Shiuan Peh
June 2009

CHAPTER 1

Introduction

Since the introduction of research into multi-core chips more than a decade ago [23, 165, 203], on-chip networks have emerged as an important and growing field of research. As core counts increase, there is a corresponding increase in bandwidth demand to facilitate high core utilization and a critical need for scalable interconnection fabrics such as on-chip networks. On-chip networks will be prevalent in computing domains ranging from high-end servers to embedded system-on-chip (SoC) devices. This diversity of application platforms has led to research in on-chip networks spanning a variety of disciplines from computer architecture to computer-aided design, embedded systems, VLSI and more. Here, we provide a synthesis of critical concepts in on-chip networks to quickly bootstrap students and designers into this exciting field.

1.1 THE ADVENT OF THE MULTI-CORE ERA

The combined pressures from ever-increasing power consumption and the diminishing returns in performance of uniprocessor architectures have led to the advent of multi-core chips. With a growing number of transistors available at each new technology generation, coupled with a reduction in design complexity enabled by the modular design of multi-core chips, this multi-core wave looks set to stay, in both general-purpose computing chips as well as application-specific SoCs. Recent years have seen every industry chip vendor releasing multi-core products with increasing core counts. This multi-core wave may lead to hundreds and even thousands of cores integrated on a single chip. In addition to the integration of many general-purpose cores on a single chip, increasing transistor counts will lead to greater system integration for multiprocessor systems-on-chip (MPSoCs). MPSoCs will leverage a wide array of components, including processing cores, embedded memory and accelerators such as DSP modules and video processors.

MPSoC: Multiprocessor systems-on-chip

1.1.1 COMMUNICATION DEMANDS OF MULTI-CORE ARCHITECTURES

As the number of on-chip cores increases, a scalable and high-bandwidth communication fabric to connect them becomes critically important [167, 57, 59]. As a result, packet-switched on-chip networks are fast replacing buses and crossbars to emerge as the pervasive communication fabric in many-core chips. Such on-chip networks have routers at every node, connected to neighbors via short local on-chip wiring, while multiplexing multiple communication flows over these interconnects to provide scalability and high bandwidth. This evolution of interconnection networks as core count increases is clearly illustrated in the choice of a flat crossbar interconnect connecting all eight cores in the Sun Niagara (2005) [120], four packet-switched rings in the 9-core IBM Cell (2005) [108], and five packet-switched meshes in the 64-core Tilera TILE64 (2007) [212].

Multi-core and many-core architectures will be commonplace in a variety of computing domains. These architectures will enable increased levels of server consolidation in data centers [68, 143, 13]. Desktop applications, particularly graphics can leverage the multi-core wave [190, 137]. High-bandwidth communication will be required for these throughput-oriented applications. Communication latency can have a significant impact on the performance of multi-threaded workloads; synchronization between threads will require low-overhead communication in order to scale to a large number of cores. In MPSoCs, leveraging an on-chip network can help enable design isolation: MPSoCs utilize heterogeneous IP blocks[1] from a variety of vendors; with standard interfaces, these blocks can communicate through an on-chip network in a plug-and-play fashion.

1.2 ON-CHIP VS. OFF-CHIP NETWORKS

While on-chip networks can leverage ideas from prior multi-chassis interconnection networks[2] used in supercomputers [72, 189, 4, 76], clusters of workstations [18] and Internet routers [44], the design requirements facing on-chip networks differ starkly in magnitude; hence novel designs are critically needed. Fortunately, by moving on-chip, the I/O bottlenecks that faced prior multi-chassis interconnection networks are alleviated substantially: The abundant on-chip wiring supplies bandwidth that is orders of magnitude higher than off-chip I/Os while obviating the inherent delay overheads associated with off-chip I/O transmission.

On the other hand, a number of stringent technology constraints present challenges for on-chip network designs. Specifically, on-chip networks targeting high-performance multi-core processors must supply high bandwidth at ultra-low latencies, with a tight power envelope and area budget. With multi-core and many-core chips, caches and interconnects compete with the cores for the same chip real estate. Integrating a large number of components under tight area and power constraints poses a significant challenge for architects to create a balance between these components. For instance, the Sun Niagara 2's flat 8×9 crossbar interconnecting all cores and the memory controller has an area footprint close to that of a core. For the 16 cores in Sun's Rock architecture, if the same flat crossbar architecture is used, it will require a 17×17 crossbar that will take up at least $8x$ more area than the final hierarchical crossbar design chosen: A 5x5 crossbar connecting clusters of four cores each [204].

In order for widespread adoption of on-chip networks, the communication latency of a network implementation must be competitive with crossbars. Furthermore, although on-chip networks require much less power than buses and crossbars, they need to be carefully designed as on-chip network power consumption can be high [211, 27]. For example, up to $\sim$ 30% of chip power is consumed by Intel's 80-core TeraFLOPS network [103, 207] and 36% by the RAW on-chip network [202]. Therefore, it is essential that power constraints are evaluated as this field matures.

[1] IP blocks are intellectual property in the form of soft macros of reusable logic.
[2] Also referred to as off-chip interconnection networks.

1.3 NETWORK BASICS: A QUICK PRIMER

In the next few sections, we lay a foundation for terminology and topics covered within this book. Subsequent chapters will explore many of these areas in more depth as well as state-of-the-art research for different components of on-chip network design. Many fundamental concepts are applicable to off-chip networks as well with different sets of design trade-offs and opportunities for innovation in each domain.

Several acronyms have emerged as on-chip network research has gained momentum. Some examples are NoC (network-on-chip), OCIN (on-chip interconnection network) and OCN (on-chip network). As there is no widely-agreed upon difference between them, throughout this book we avoid the use of acronyms and generically refer to all kinds of on-chip networks.

NoC
OCIN
OCN

1.3.1 EVOLUTION TO ON-CHIP NETWORKS

An on-chip network, as a subset of a broader class of interconnection networks, can be viewed as a programmable system that facilitates the transporting of data between nodes[3]. An on-chip network can be viewed as a system because it integrates many components including channels, buffers, switches and control.

With a small number of nodes, dedicated ad hoc wiring can be used to interconnect them. However, the use of dedicated wires is problematic as we increase the number of components on-chip: The amount of wiring required to directly connect every component will become prohibitive.

Designs with low core counts can leverage buses and crossbars, which are considered the simplest variants of on-chip networks. In both traditional multiprocessor systems and newer multi-core architectures, bus-based systems scale only to a modest number of processors. This limited scalability is because bus traffic quickly reaches saturation as more cores are added to the bus, so it is hard to attain high bandwidth. The power required to drive a long bus with many cores tapping onto it is also exorbitant. In addition, a centralized arbiter adds arbitration latency as core counts increase. To address these problems, sophisticated bus designs incorporate segmentation, distributed arbitration, split transactions and increasingly resemble switched on-chip networks.

Crossbars address the bandwidth problem of buses, and have been used for on-chip interconnects for a small number of nodes. However, crossbars scale poorly for a large number of cores; requiring a large area footprint and consuming high power. In response, hierarchical crossbars, where cores are clustered into nodes and several levels of smaller crossbars provide the interconnection, are used. These sophisticated crossbars resemble multi-hop on-chip networks where each hop comprises small crossbars.

On-chip networks are an attractive alternative to buses and crossbars for several reasons. First and foremost, networks represent a scalable solution to on-chip communication, due to their ability to supply scalable bandwidth at low area and power overheads that correlate sub-linearly with the number of nodes. Second, on-chip networks are very efficient in their use of wiring, multiplexing different communication flows on the same links allowing for high bandwidth. Finally, on-chip

[3] A node is any component that connects to the network: e.g core, cache, memory controller, etc.

networks with regular topologies have local, short interconnects that are fixed in length and can be optimized and built modularly using regular repetitive structures, easing the burden of verification.

1.3.2 ON-CHIP NETWORK BUILDING BLOCKS

The design of an on-chip network can be broken down into its various building blocks: its topology, routing, flow control, router microarchitecture and design, and link architecture. The rest of this lecture is organized along these building blocks and we will briefly explain each in turn here.

Topology. An on-chip network is composed of channels and router nodes. The network topology determines the physical layout and connections between nodes and channels in the network.

Routing. For a given topology, the routing algorithm determines the path through the network that a message will take to reach its destination. A routing algorithm's ability to balance traffic (or load) has a direct impact on the throughput and performance of the network.

Flow control. Flow control determines how resources are allocated to messages as they travel through the network. The flow control mechanism is responsible for allocating (and de-allocating) buffers and channel bandwidth to waiting packets[4]. Resources can be allocated to packets in their entirety (done in store-and-forward and virtual cut-through flow control); however, this requires very large buffer resources making it impractical on chip. Most commonly, on-chip networks handle flow control at the flit level[5]. Buffers and channel bandwidth are allocated on the smaller granularity of flits rather than whole packets; as a result, routers can be designed with smaller buffers.

Router microarchitecture. A generic router microarchitecture is comprised of the following components: input buffers, router state, routing logic, allocators, and a crossbar (or switch). Router functionality is often pipelined to improve throughput. Delay through each router in the on-chip network is the primary contributor to communication latency. As a result, significant research effort has been spent reducing router pipeline stages and improving throughput.

Link architecture. Thus far, all on-chip network prototypes have used conventional full-swing logic and pipelined wires. Pipelined wires uses repeaters to improve signal reach. While full-swing repeatered links dominate current designs, there are exciting ongoing research and opportunities for alternative link architectures. We discuss this research in Chapter 7.

1.3.3 PERFORMANCE AND COST

As we discuss different on-chip design points and relevant research, it is important to consider both the performance and the cost of the network. Performance is generally measured in terms of network latency or accepted traffic. For back-of-the-envelope performance calculations, zero-load latency is often used, i.e. the latency experienced by a packet when there are no other packets in the network. Zero-load latency provides a lower bound on average message latency. Zero-load latency is found by taking the average distance (given in terms of network hops) a message will travel times the latency to traverse a single hop.

zero-load latency

[4] A packet is a subdivision of a message.
[5] A flit is a flow control unit, a subdivision of a packet.

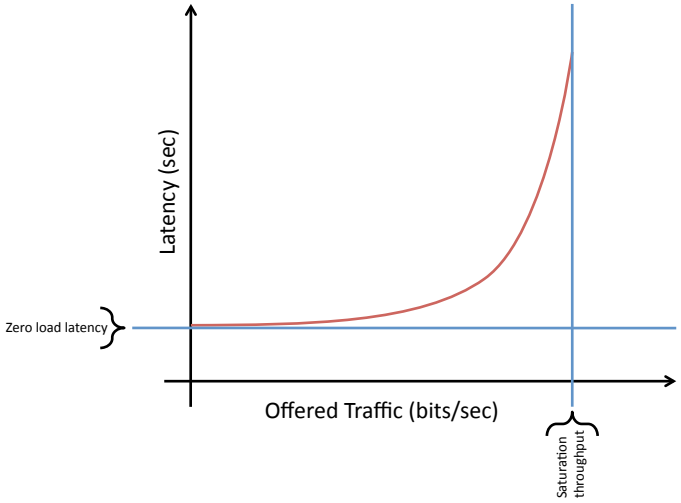


Figure 1.1: Latency vs Throughput for an on-chip network.

In addition to providing ultra-low latency communication, network's must also deliver high throughput. Therefore, performance is also measured by its throughput. A high saturation throughput indicates that the network can accept a large amount of traffic before all packets experience very high latencies, sustaining higher bandwidth. Figure 1.1 presents a latency versus throughput curve for an on-chip network illustrating the zero-load latency and saturation throughput.

saturation throughput

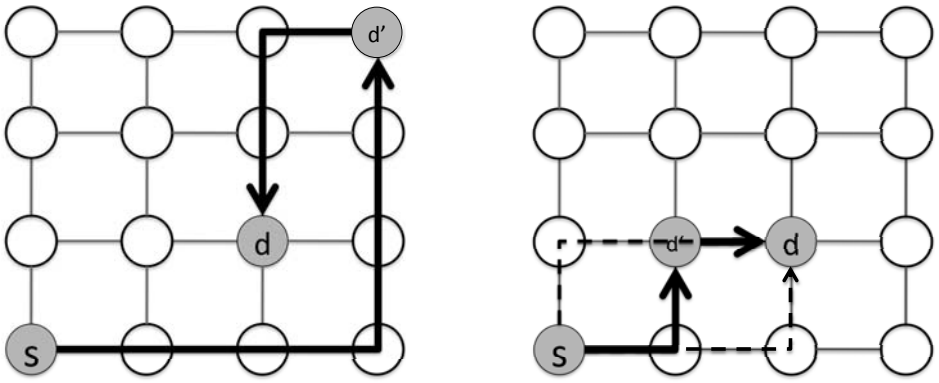
The two primary costs associated with an on-chip network are area and power. As mentioned, many-core architectures operate under very tight power budgets. The impact of different designs on power and area will be discussed throughout this book.

1.4 COMMERCIAL ON-CHIP NETWORK CHIPS

With on-chip networks being a nascent research area, there are only a handful of chips that have been designed incorporating sophisticated on-chip networks [202, 187, 94, 190, 199, 184, 130, 154, 93, 212, 208][6]. In this book, we select four commercial chips as case studies for illustrating the diversity in on-chip network chip designs. A brief summary of the chips is given here, followed by a discussion of their specific design choices in network interface design, topology, routing, flow control and router microarchitecture in subsequent chapters.

IBM Cell. The Cell architecture [94, 85] is a joint effort between IBM, Sony and Toshiba to design a power-efficient family of chips targeting game systems, but that are general enough for other domains as well. Cell is a product that is in most game consoles in the market today. It is a 90nm, $221mm^2$ chip that can run at frequencies above 4GHz. It consists of one IBM 64-

[6]This is a non-exhaustive list of academic and industrial chips, to the best of the authors' knowledge.



(a) Valiant's Routing Algorithm

(b) Minimal Oblivious Routing Algorithm

Figure 4.4: Oblivious Routing Examples.

chooses between X-Y or Y-X routes is not deadlock-free because all four turns from Figure 4.2 are possible leading to potential cycles in the link acquisition graph.

4.5 ADAPTIVE ROUTING

A more sophisticated routing algorithm can be adaptive, i.e. the path a message takes from A to B depends on the network traffic situation. For instance, a message can be going along the X-Y route, see congestion at (1,0)'s east outgoing link and instead choose to take the north outgoing link towards the destination (see Figure 4.1).

Local or global information can be leveraged to make adaptive routing decisions. Adaptive routing algorithms often rely on local router information such as queue occupancy and queuing delay to gauge congestion and select links [50]. The backpressure mechanisms used by flow control (discussed in the next chapter) allow congestion information to propagate from the congestion site back through the network.

Figure 4.5 shows all possible (minimal) routes that a message can take from Node (0,0) to Node (2,3). There are nine possible paths. An adaptive routing algorithm that leverages only minimal paths could exploit a large degree of path diversity to provide load balancing and fault tolerance.

Adaptive routing can be restricted to taking minimal routes between the source and the destination. An alternative option is to employ misrouting, which allows a packet to be routed in a non-productive direction resulting in non-minimal paths. When misrouting is permitted, livelock becomes a concern. Without mechanisms to guarantee forward progress, livelock can occur as a packet is continuously misrouted so as to never reach its destination. We can combat this problem

misrouting

livelock

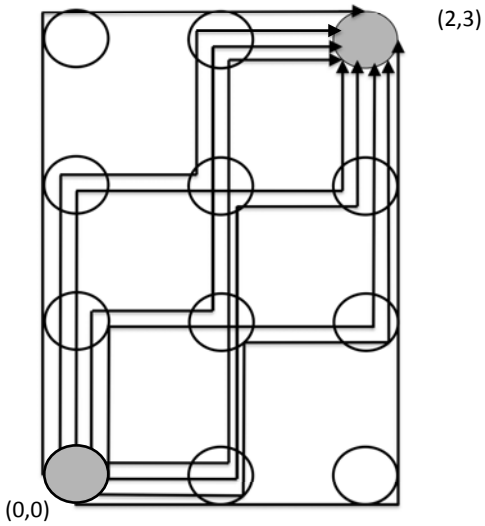


Figure 4.5: Adaptive Routing Example.

by allowing a maximum number of misroutes per packet and giving higher priority to packets than have been misrouted a large number of times. Misrouting increases the hop count but may reduce end-to-end packet latency by avoiding congestion (queueing delay).

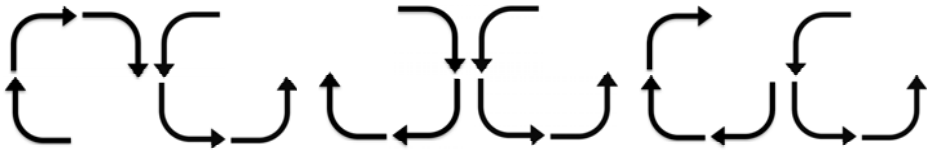
With a fully-adaptive routing algorithm, deadlock can become a problem. For example, the adaptive route shown in Figure 4.1 is a superset of oblivious routing and is subject to potential deadlock. Planar-adaptive routing [38] limits the resources needed to handle deadlock by restricting adaptivity to only two dimensions at a time. Duato has proposed flow control techniques that allow full routing adaptivity while ensuring freedom from deadlock [62]. Deadlock-free flow control will be discussed in Chapter 5.

Another challenge with adaptive routing is preserving inter-message ordering as may be needed by the coherence protocol. If messages must arrive at the destination in the same order that the source issued them, adaptive routing can be problematic. Mechanisms to re-order messages at the destination can be employed or messages of a given class can be restricted in their routing to prevent re-ordering.

4.5.1 ADAPTIVE TURN MODEL ROUTING

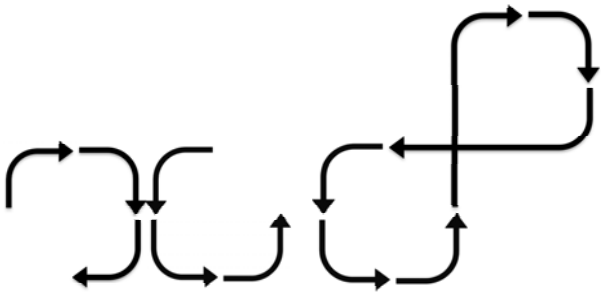
turn model routing

While we introduced turn model routing earlier in Section 4.3, discussing how dimension order X-Y routing eliminates two out of four turns (Figure 4.3), here, we explain how turn model can be more broadly applied to derive deadlock-free adaptive routing algorithms. Adaptive turn model routing eliminates the minimum set of turns needed to achieve deadlock freedom while retaining some path diversity and potential for adaptivity.



(a) West First Turns (b) North Last Turns (c) Negative First Turns

Figure 4.6: Turn Model Routing.

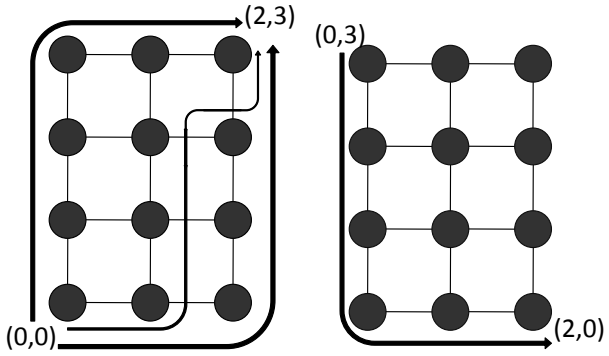


(a) Illegal Turn Model Routing (b) Resulting Deadlock Cycle

Figure 4.7: Turn Model Deadlock.

With dimension order routing only four possible turns are permitted of the eight turns available in a two dimensional mesh. Turn model routing [75] increases the flexibility of the algorithm by allowing six out of eight turns. Only one turn from each cycle is eliminated.

In Figure 4.6, three possible routing algorithms are illustrated. Starting with all possible turns (shown in Figure 4.6a), the north to west turn is eliminated; after this elimination is made, the three routing algorithms shown in Figure 4.6 can be derived. In Figure 4.6a, the west-first algorithm is shown; in addition to eliminating the North to West turn, the South to West turn is eliminated. In other words, a message must first travel in the West direction before traveling in any other direction. The North-Last algorithm (Figure 4.6b) eliminates both the North to West and the North to East turns. Once a message has turned North, no further turns are permitted; hence, the North turn must be made last. Finally, Figure 4.6c removes turns from North to West and East to South to create the Negative-First algorithm. A message travels in the negative directions (west and south) first before it is permitted to travel in positive directions (east and north). All three of these turn model routing algorithms are deadlock-free. Figure 4.7 illustrates a possible turn elimination that is invalid; the



(a) Exploiting Path Diversity (b) No Path Diversity

Figure 4.8: Negative-First Routing example.

elimination of North to West combined with the elimination of West to North can lead to deadlock. A deadlock cycle is depicted in Figure 4.7b that can result from a set of messages using the turns specified in Figure 4.7a.

Odd-even turn model routing [39] proposes eliminating a set of two turns depending on whether the current node is in an odd or even column. For example, when a packet is traversing a node in an even column[1], turns from East to North and from North to West are prohibited. For packets traversing an odd column node, turns from East to South and from South to West are prohibited. With this set of restrictions, the odd-even turn model is deadlock free provided 180° turns are disallowed. The odd-even turn model provides better adaptivity than other turn model algorithms such as West-First. With West-First, destinations to the West of the source, have no flexibility; with odd-even routing, there is flexibility depending on the allowable turns for a given column.

In Figure 4.8, we apply the Negative-First turn model routing to two different source destination pairs. In Figure 4.8a, three possible routes are shown between (0,0) and (2,3) (more are possible); turns from North to East and from East to North are permitted allowing for significant flexibility. However, in Figure 4.8b, there is only one path allowed by the algorithm to route from (0,3) to (2,0). The routing algorithm does not allow the message to turn from East to South. Negative routes must be completed first, resulting in no path diversity for this source-destination pair. As illustrated by this example, turn model routing provide more flexibility and adaptivity than dimension-order routing but it is still somewhat restrictive.

[1]A column is even if the dimension-0 coordinate of the column is even.

Table 4.1: Routing Algorithm and Implementation Options .

Routing Algorithm	Source Routing	Combinational	Node Table
Deterministic			
DOR	Yes	Yes	Yes
Oblivious			
Valiant's	Yes	Yes	Yes
Minimal	Yes	Yes	Yes
Adaptive	No	Yes	Yes

4.6 IMPLEMENTATION

In this section, we discuss various implementation options for routing algorithms. Routing algorithms can be implemented using look-up tables at either the source nodes or within each router. Combinational circuitry can be used as an alternative to table-based routing. Implementations have various trade-offs, and not all routing algorithms can be achieved with each implementation. Table 4.1 shows examples for how routing algorithms in each of the three different classes can be implemented.

4.6.1 SOURCE ROUTING

Routing algorithms can be implemented in several ways. First, the route can be embedded in the packet header at the source, known as source routing. For instance, the X-Y route in Figure 4.1 can be encoded as $< E, E, N, N, N, Eject >$, while the Y-X route can be encoded as $< N, N, N, E, E, Eject >$. At each hop, the router will read the leftmost direction off the route header, send the packet towards the specified outgoing link, and strip off the portion of the header corresponding to the current hop.

There are a few benefits to source routing. First, by selecting the entire route at the source, latency is saved at each hop in the network since the route does not need to be computed or looked up. The per-router routing hardware is also saved; no combinational routing logic or routing tables are needed once the packet has received its route from the source node. Second, source routing tables can be reconfigured to deal with faults and can support irregular topologies. Multiple routes per source-destination pair can be stored in the table (as shown in Table 4.2) and selected randomly for each packet to improve load balancing.

The disadvantages of source routing include the bit overheads required to store the routing table at the network interface of each source and to store the entire routing path in each packet; these paths are of arbitrary length and can grow large depending on network size. For a 5-port switch, each routing step is encoded by a 3-bit binary number. Just as the packet must be able to handle arbitrary length routing paths, the source table must also be designed to efficiently store different length paths. Additionally, by choosing the entire route at the source node, source based routing is unable to take advantage of dynamic network conditions to avoid congestion. However, as mentioned, multiple