The original version of this paper appears in Business Communication Review.

WHITE PAPER:

A Fast Switched Backplane for a Gigabit Switched Router

by Nick McKeown (tel: 650/725-3641; fax: 650/725-6949; email: nickm@stanford.edu) a professor of electrical engineering and computer science at Stanford University. He received his PhD from the University of California at Berkeley in 1995. From 1986-1989 he worked for Hewlett-Packard Labs, in their network and communications research group in Bristol, England. In spring 1995, he worked for Cisco Systems as an architect of the Cisco 12000 GSR router. Nick serves as an editor for the IEEE Transactions on Communications. He is the Robert Noyce Faculty Fellow at Stanford and recipient of a fellowship from the Alfred P. Sloan Foundation. Nick researches techniques for high-speed networks, including high-speed Internet routing and architectures for high-speed switches. More recently, he has worked on the analysis and design of cell-scheduling algorithms, memory architectures and the economics of the Internet. His research group is currently building the Tiny Tera; an all-CMOS Terabit-per-second network switch.

Table of Contents

1 Abstract

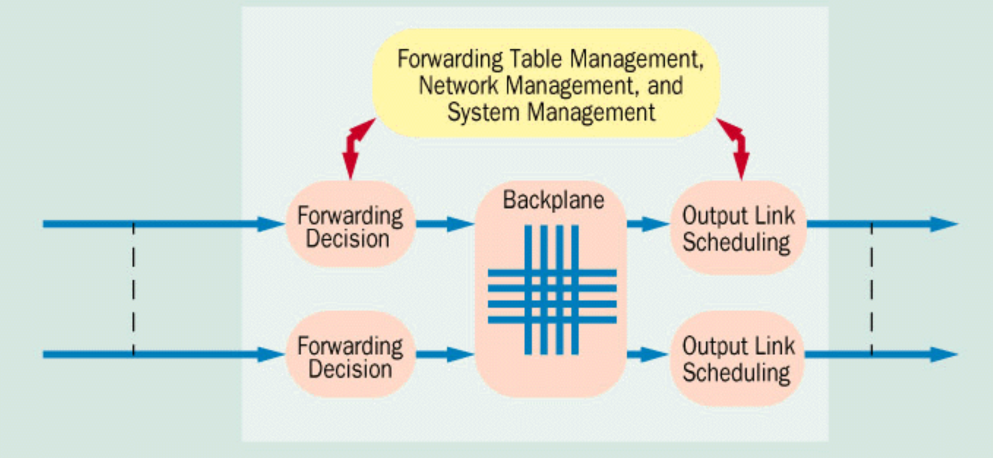
There is a new trend in the architecture of high-performance Internet routers: congested, shared backplanes are being replaced by much faster switched backplanes that allow multiple packets to be transferred simultaneously. This paper explains why switched backplanes are needed now, and the technical problems that must be solved in their design. Although others are taking the same approach, we will focus, as an example, on the switched backplane developed for the Cisco 12000 GSR.

This router's backplane can switch 16 ports simultaneously, each with a line rate of 2.4 Gbps. It uses a number of new technologies that enable a parallel, compact design, providing extremely high throughput for both unicast and multicast traffic. Integrated support for priorities on the backplane allows the router to provide distinguished qualities of service for multimedia applications.

Top of Page

2 The Architecture of Internet Routers

FIGURE 1 The Key Common Architectural Components of a Router



In high-performance systems, the forwarding decision, backplane and output link scheduling must be performed in hardware, while the less timely management and maintenance functions are performed in software.

We begin by revisiting the general architecture of an IP router, as shown in a simplified form in Figure 1. Router functions can be separated into two types:

1. *Datapath Functions:* operations that are performed on every datagram that passes through the router. These are most often implemented in special purpose hardware, and include the forwarding decision, the backplane and output link scheduling.

2. *Control Functions:* operations that are performed relatively infrequently. These are invariably implemented in software and include the exchange of routing table information with neighboring routers, as well as system configuration and management.

Therefore, when trying to improve the per-packet performance of a router, we naturally focus on the datapath functions. Let's take a closer look at the datapath functions, by tracing the typical path of a packet through an IP router:

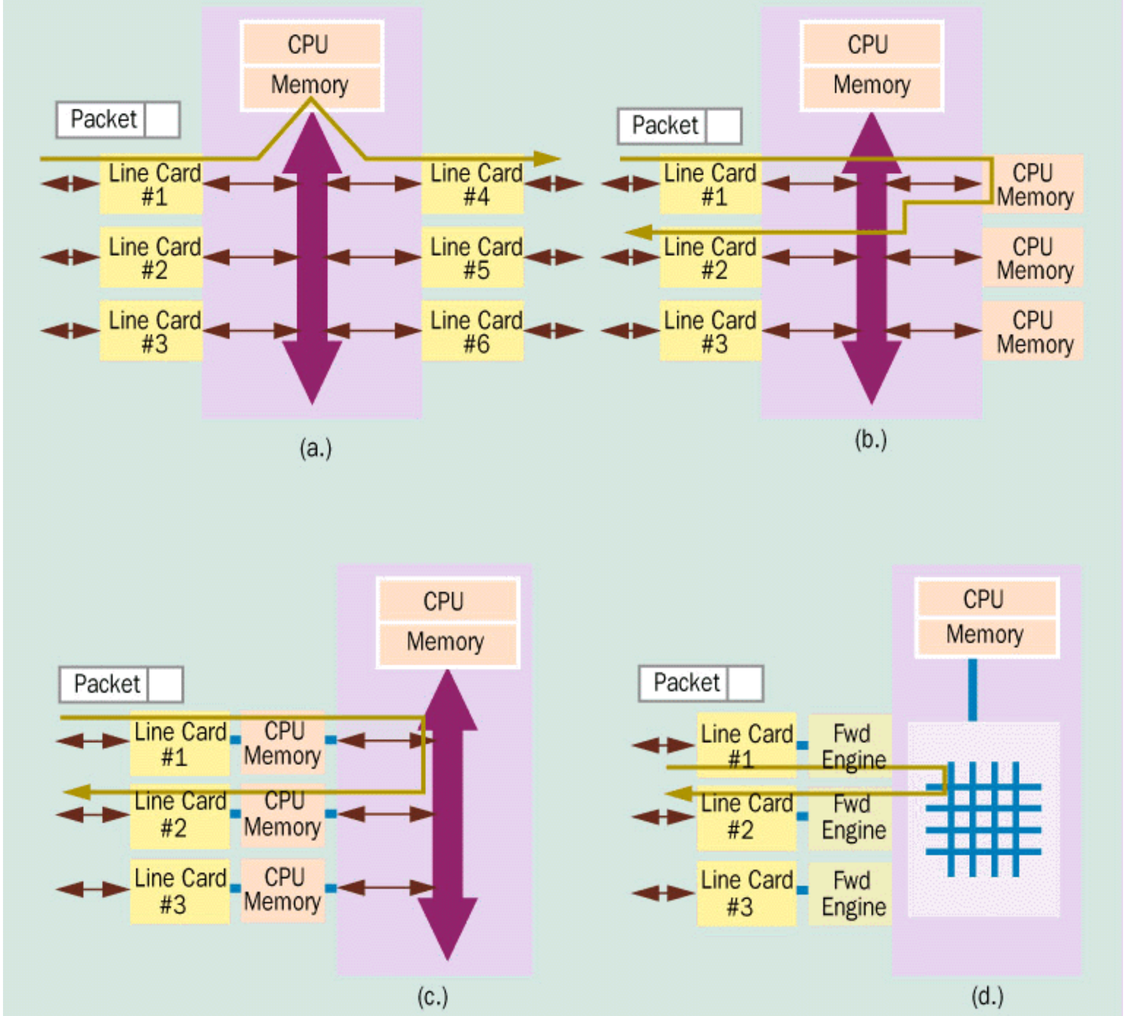
- *The Forwarding Decision:* When a datagram arrives, its destination address is looked up in a forwarding table. If the address is found, a next-hop MAC address is appended to the front of the datagram, the time-to-live (TTL) field of the IP datagram header is decremented, and a new header checksum is calculated.
- *The Backplane:* The datagram is then forwarded across the backplane to its outgoing port (or ports). While waiting its turn to be transferred across the backplane, the datagram may need to be queued: if insufficient queue space exists, the datagram may need to be dropped, or it may be used to replace other datagrams.
- *The Output-Link Scheduler:* When it reaches the outgoing port, the datagram waits for its turn to be transmitted on the output link. In most routers today, the outgoing port maintains a single first-come-first-served (FIFO) queue and transmits datagrams in the order they arrive. More advanced routers distinguish datagrams into different flows, or priority classes, and carefully schedule the departure time of each datagram in order to meet specific quality of service guarantees.

Over the years, particular router architectures have been selected on the basis of a number of factors, including cost, number of ports, required performance and currently available technology. While the detailed implementation of individual commercial routers have remained proprietary, all routers have evolved along similar lines. Moreover, at any one time, higher-performance routers have differed from lower-performance devices in similar ways.

The first trend in router evolution has been to implement more of the datapath functions in hardware. Recent improvements in CMOS technology integration have made it possible to implement a larger number of functions in ASIC components, moving datapath functions that were once performed in software to special-purpose hardware. Increased integration doesn't merely enable the same functions to be performed at similar speed for a lower cost; with sufficient hardware, system performance can also be significantly improved.

The second trend has been toward parallelism, either to achieve higher performance or to allow the use of lower-cost components. Third, and most important to our discussion here, there has been a trend away from the use of shared buses (also referred to as shared-memory backplanes). Buses that are shared between multiple functions can become congested, especially if the bus bandwidth doesn't match the aggregate data rate of the ports and CPU I/O, thus limiting the performance of the system.

FIGURE 2 The Basic Architectures of Packet Processors



Note: Over time, or to enhance performance, more hardware is used on the main datapath and more parallelism is employed.

As shown in Figure 2(a), the earliest routers were built around a conventional computer architecture: a shared central bus, with a central CPU, memory and peripheral Line Cards. Each Line Card performs the MAC-layer function, connecting the system to each of the external links. Packets arriving from a link are transferred across the shared bus to the CPU, where a forwarding decision is made. The packet is then transferred across the bus again to its outgoing Line card, and onto the external link.

The main limitation of this architecture is that the central CPU must process every packet, ultimately limiting the throughput of the system. This limitation prompted the architecture in Figure 2(b), in which multiple CPUs process packets in parallel. Incoming packets are forwarded to a CPU as before, but this time there is a choice. For example, the packet could be sent to the first available CPU, or packets with the same destination address could always be sent to the same CPU. The advantage is clear: Parallelism can increase system throughput or allow the use of lower-cost CPUs.

The architecture in Figure 2(c) takes parallelism one stage further, placing a separate CPU at each interface. Local forwarding decisions are made in these dedicated CPUs, and the packet is immediately forwarded to its outgoing interface. Since each packet traverses the bus only once, system throughput is again increased. The central CPU maintains the forwarding tables in the line card CPUs, and handles system management functions.

The performance of the architecture in Figure 2(c) is ultimately limited by two factors. First, forwarding decisions are made in software, and so are limited by the speed of a general-purpose CPU. Carefully designed, special-purpose ASICs can readily outperform a CPU when making forwarding decisions, managing queues and arbitrating access to the bus. Hence, CPUs are increasingly being replaced by specialized ASICs.

The second limiting factor is the use of a shared bus—only one packet may traverse the bus at a time between two line cards. Performance can be increased if multiple packets can be transferred across the bus simultaneously. This is why crossbar switches are replacing shared buses in the latest router architectures.

In a crossbar switch, multiple line cards can communicate with each other simultaneously, greatly increasing the system throughput. So, by introducing hardware forwarding engines on the line cards and replacing the bus with a crossbar switch, we reach the architecture shown in Figure 2(d). Today, the very-highest-performance routers are designed according to this architecture.

Because our work is motivated by the following configuration, we will use it as an example throughput this paper: a 16-port IP router, with each port operating at 2.4 Gbps. Let's consider how fast each of the datapath functions must operate in our example router.

First, a new forwarding decision must be made for every arriving datagram. With today's average Internet datagram length of 250 bytes, each port on our router must make a forwarding decision every 800ns. This corresponds to approximately 1.2 million decisions per second. Performing a router lookup in such a short time may sound like a difficult goal, but in practice it is quite straightforward. In fact, it is a common misconception that routers today—operating at much lower data rates than our example—are limited by their ability to forward packets. This is not so. Techniques for making over a million forwarding decisions per second have been known for some time. And more recently, a variety of techniques operating at 5- to 20-million forwarding decisions per second have been described [2] [1] [18] [19] [20].

Now let's take a look at the performance we need for our router's backplane. In the worst case, all of the line cards may wish to transfer datagrams simultaneously, requiring a backplane with an aggregate bandwidth of 38.4 Gbps. If the backplane cannot keep up, the shared bus becomes congested, buffers overflow and packets are dropped. As we will see, it is impractical today to build shared backplanes operating at 38.4 Gbps. But fortunately, switched backplanes provide a simple and efficient alternative, and can easily operate at these rates. The Cisco 12000 router uses the architecture in Figure 2(d) allowing multiple packets to be transferred simultaneously.

Top of Page

3 Why We Need Switched Backplanes

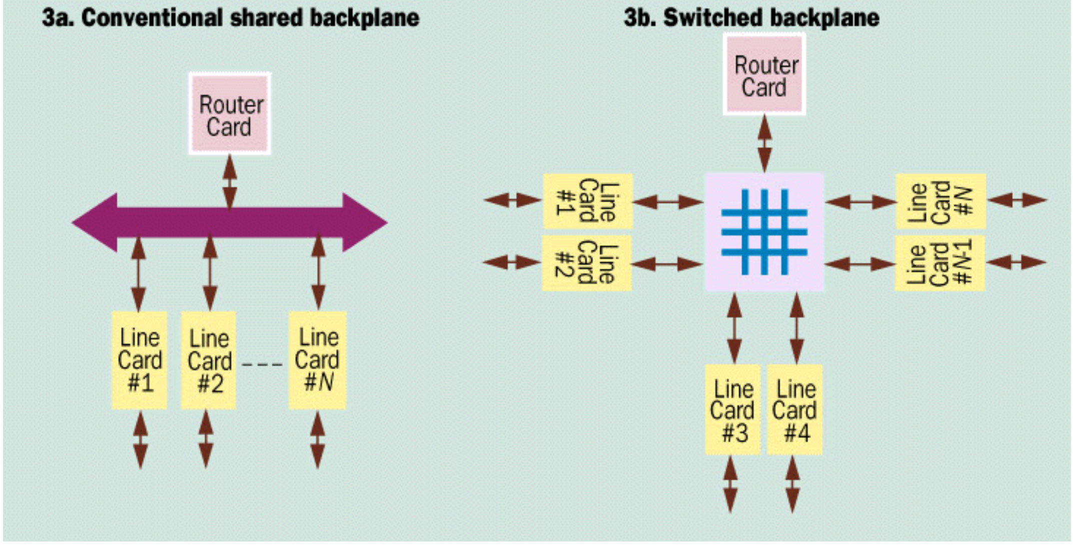
If backplanes are congested, why don't we just make them faster? After all, the computer industry has introduced faster and faster buses over the past few years: from ISA to EISA and now PCI. Unfortunately, while the need for routing bandwidth is growing exponentially, bus bandwidth is growing much more slowly.

State of the art shared buses today can achieve a maximum capacity of about 20 Gbps[3]. This is more than adequate for bridges and routers with a few 100-Mbps Ethernet ports. But for our router with line rates of 2.4 Gbps, we need an aggregate bandwidth of 38.4 Gbps; a shared backplane cannot keep up.

Very-high-speed shared backplanes are impractical for two reasons. The first is congestion: the bandwidth of the bus is shared among all the ports, so contention leads to additional delay for packets. If the arrival rate of packets exceeds the bus bandwidth for a sustained period, buffers will overflow and data will be lost. Second, the transfer-capacity of a multiport shared bus is limited by electrical loading, by the number of connectors that a signal encounters and by the reflections from the end of unterminated lines.

4 Switched Backplanes: An Overview

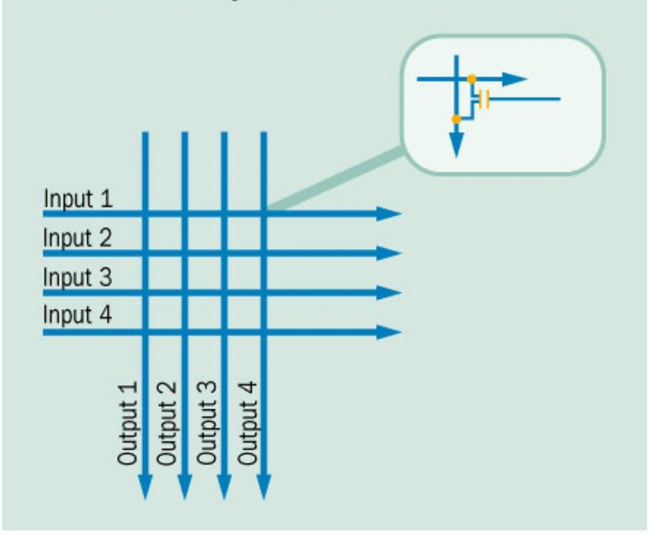
FIGURE 3 Comparison of Conventional and Switched Backplane Designs for a High-Performance Router



4.1 Crossbar Switch: The limitations of a shared backplane are eliminated with a switched backplane. Figure 3 makes a simplified comparison between a conventional shared backplane and a switched backplane comprising multiple line cards arranged around a central crossbar switch. Crossbar switches are widely used for switched backplanes because of their simplicity.

The crossbar switch enables high performance for two reasons: First, connections from line cards to the central switch are now simple point-to-point links, which means they can operate at very high speed. In the past few years, semiconductor companies have developed chip-to-chip serial links operating at over 1 Gbps using a conventional CMOS process. These serial links are now being deployed in a number of commercial switches and routers. And laboratory results suggest that links operating at 4 to 10 Gbps will be available in the next few years. With only one transmitter on each wire, reflections can be controlled, allowing signals to settle in a shorter time. Short point-to-point links also help control clock skew and signal integrity, and they reduce electromagnetic interference.

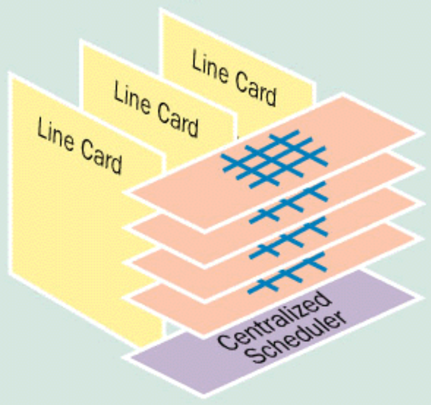
FIGURE 4 A Four-Input Crossbar Interconnection Fabric



The second reason a crossbar switch can provide higher performance is that it can support multiple bus transactions simultaneously. This greatly increases the aggregate bandwidth of the system. A crossbar switch is shown in Figure 4; by closing several crosspoints at the same time, the switch can transfer packets between multiple ports simultaneously. In fact, we say that crossbar switches are internally non-blocking because they allow all inputs and outputs to transfer packets simultaneously.

The crosspoints are controlled by a centralized scheduler. It considers all the packets waiting to be transferred across the switch fabric and selects a configuration of the crossbar, ensuring that at any one instance each input is connected to at most one output, and that each output is connected to at most one input.

FIGURE 5 A Four-Way Parallel Crossbar Switch, Interconnecting Three Line Cards



Note: A centralized scheduler connects to each line card, and determines the configuration of the crossbar switch for each time slot.

The simple structure of a crossbar switch allows it to be readily implemented in silicon. Furthermore, we can boost performance by using multiple crossbar switches, or slices, in parallel, to form a switch core. Figure 5 shows a system with multiple line cards connected to a four-way parallel crossbar. During each time slot, all the crossbar slices are set to the same configuration by the central scheduler, presenting a 4-bit-wide bus from each line card to the switch core. If the connection from each line card to each slice uses one of the fast serial links described above, we can readily build a switched backplane with a very high aggregate bandwidth.

4.2 Fixed vs. Variable Length Packets: Packets may be transferred across the switched backplane in variable or fixed-length units (or *cells*

[Note: Although we borrow the term *cells* from ATM, we use it here to mean packets of any fixed length and format]). Life is simple if we used fixed-length cells, since we can think of time progressing in fixed-length increments, or time-slots. At the end of each time slot, the scheduler examines the packets waiting to be transferred across the crossbar switch and selects the configuration that will connect the appropriate inputs and outputs. As discussed below in "4.5 The iSLIP Algorithm", this scheduling decision can be made in a way that is fair among connections, never starves input or output, and maintains highly efficient use of the crossbar switch.

Efficiency is important, because the crossbar switch is a central resource shared among all the line cards. Wasting its bandwidth would degrade the performance of the entire system, causing the same problems as an undersized shared backplane—e.g., overflowing buffers and queues and lost packets. Finally, from a hardware-design perspective, experience has proved that processing fixed-size cells is simpler and faster than handling variable-length packets.

Life is much harder if we use variable-length packets, because they may finish traversing the crossbar switch at any time. This means the scheduler must constantly track the busy and idle status of the outputs and, as soon as an output becomes idle, quickly choose which input is allowed to connect to it next. Not only might several inputs be waiting for the same output, but an idle input might have packets for several different outputs. Should the scheduler make the input wait (for a possibly long time) for a busy output to become free, or should it immediately connect to an idle output? If it does not wait, packets can be held up in the inputs and starved to the idle output, placing the switch at the mercy of the arriving traffic. This is why switches that handle variable-length packets have lower throughput, and are able to use only about half the system's aggregate bandwidth (see I. Iliadis, "Head of the line arbitration of packet switches with combined input and output queueing," *International Journal of Digital and Analog Communication Systems*, volume 4, pp. 181-190 (1991).) We choose switched backplanes with fixed-length cells because of their superior performance.

4.3 Input Queuing and Virtual Output Queuing: Although a crossbar switch is internally non-blocking, three other types of blocking can limit its performance: head-of-line (HOL) blocking, input blocking and output blocking. We must understand and avoid each of these blocking mechanisms if we want to build a very-high-performance switched backplane using a crossbar switch.

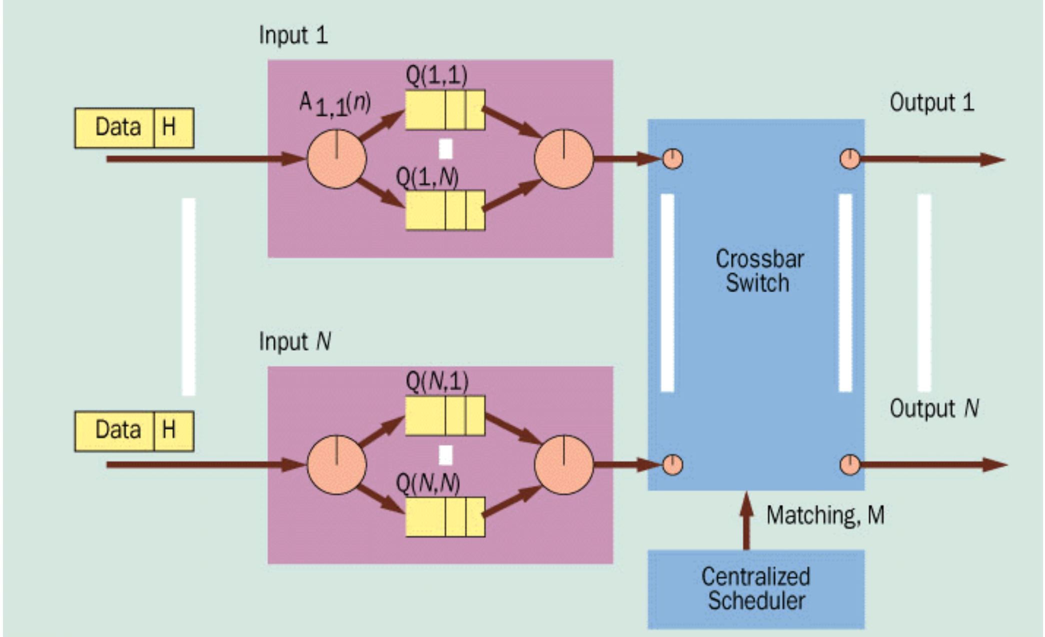
HOL blocking can waste nearly half a crossbar switch's bandwidth, if the cells waiting at each input are stored in a single first-in-first-out (FIFO) queue. Only when a cell reaches the head of its FIFO queue can it be considered by the centralized scheduler. Then the cell contends for its output with other cells that are destined to the same output and currently at the HOL of other inputs. The centralized scheduler must decide which cell will go next.

Although each cell will eventually be selected and delivered to its output across the crossbar switch, FIFO queueing invariably leads to HOL blocking. Put another way, the centralized scheduler "sees" only the cell at the head of each FIFO queue, so that cell blocks those behind it that need to be delivered to different outputs.

A good analogy is an intersection controlled by a traffic light, in which vehicles turning to the right are allowed to proceed even when the light is red. If there is only a single lane, you can be blocked from making the right-hand turn if the driver of the car ahead of you wants to go straight ahead.

Even under benign traffic patterns, HOL blocking limits the throughput to just 60 percent of the aggregate bandwidth for fixed- [4] or variable-length packets. When the traffic is bursty, or when it favors particular output ports, the throughput can be much worse.

FIGURE 6 Model of an *N*-port Input-Queued Switch with Virtual Output Queueing (VOQ)



Note: Cells arrive at input 1, and are placed into the appropriate VOQ. At the beginning of each time slot, the centralized scheduler selects a configuration for the crossbar, by matching inputs to outputs. Head of line blocking is eliminated by using a separate queue for each output at each input.

Fortunately, there is a simple fix to this problem known as virtual output queueing (VOQ), first proposed in "High Performance Multiqueue Buffers for VLSI Communication Switches" (June 1988) [5]. At each input, a separate FIFO queue is maintained for each output, as shown in Figure 6. After a forwarding decision has been made, arriving cells are placed in the queues corresponding to their outgoing ports. A centralized scheduling algorithm then examines the contents of all the input queues, and finds a conflict-free match between inputs and outputs.

In theory, an efficient scheduling algorithm can increase the throughput of a crossbar switch from 60 percent with simple FIFO queueing to a full 100 percent if VOQs are used[6]. This is because HOL blocking has been eliminated entirely: No cell is held up by a cell in front of it that is destined to a different output.

In our traffic-light analogy, this is equivalent to having a dedicated right-turn lane: the car in front of you no longer blocks you, and you are free to proceed to your free output. Our switched backplane will use virtual output queueing to boost performance, which we will examine more closely in "Unicast Traffic: Performance Comparison."

The other types of blocking, input and output blocking, are present in all crossbar switches. They arise because of contention for access to the crossbar: Each input and each output can transport or receive only one cell at a time. If multiple cells wish to access an input or output simultaneously, only one will gain access, while the others will be queued. Technically, input and output blocking do not reduce the throughput of a crossbar switch, but they do increase the delay of individual packets through the system and, perhaps more important, make the delay random and unpredictable. We can control packet delay by using a prioritization mechanism in the crossbar switch, and with another technique called speedup. Our switch uses both methods, which we will in detail in "Controlling Delay."

4.4 Crossbar Scheduling Algorithms: In recent years, there has been a significant amount of research work on scheduling algorithms for crossbar switches that use virtual output queueing. Scheduling algorithms have been developed by a number of researchers[23] [24] [25] [26], for research prototypes. [2] [10], and commercial products[7]. What makes a good crossbar scheduling algorithm for the backplane of a high-performance router? We desire algorithms with the following properties:

- High Throughput—An algorithm that keeps the backlog low in the VOQs. Ideally, the algorithm will sustain an offered load up to 100 percent on each input and output.
- Starvation Free—The algorithm should not allow any VOQ to be unserved indefinitely.
- Fast—The scheduling algorithm must not become the performance bottleneck. It should select a crossbar configuration as quickly as possible.
- Simple to implement—To be fast in practice, the algorithm must be implemented in special-purpose hardware; preferably within a single chip.

4.5 The *i*SLIP Algorithm

The *i*SLIP (an arbitrary name, pronounced "eye slip") algorithm is designed to meet our goals. The main characteristic of *i*SLIP is its simplicity; it is readily implemented in hardware and can operate at high speed. A prototype version is currently being implemented that makes a new scheduling decision every 40 ns[10].

*i*SLIP is an iterative algorithm—during each time slot, multiple iterations are performed to select a crossbar configuration, matching inputs to outputs. It uses rotating priority ("round-robin") arbitration to schedule each active input and output in turn.

*i*SLIP attempts to quickly converge on a conflict-free match in multiple iterations, where each iteration consists of three steps. All inputs and outputs are initially unmatched and only those inputs and outputs not matched at the end of one iteration are eligible for matching in the next. The three steps of each iteration operate in parallel on each output and input, as follows:

- Step 1. Request—Each input sends a request to every output for which it has a queued cell.
- Step 2. Grant—If an output receives any requests, it chooses the one that appears next in a fixed, round-robin schedule of the inputs starting from the highest-priority input. The output notifies each input whether or not its request was granted.
- Step 3. Accept—If an input receives a grant, it accepts the one that appears next in a fixed, round-robin schedule starting from the highest priority output. The pointer to the highest-priority output incremented (modulo *N*) to one location beyond the accepted output. Likewise, the pointer to the highest-priority input is incremented (modulo *N*) to one location beyond the granted input. The pointers are updated only after the first iteration; subsequent iterations match inputs and outputs that were not matched during earlier iterations.

Despite its simplicity, the performance of *i*SLIP is surprisingly good, as discussed in [8], and briefly in [9]. It matches our requirements as follows:

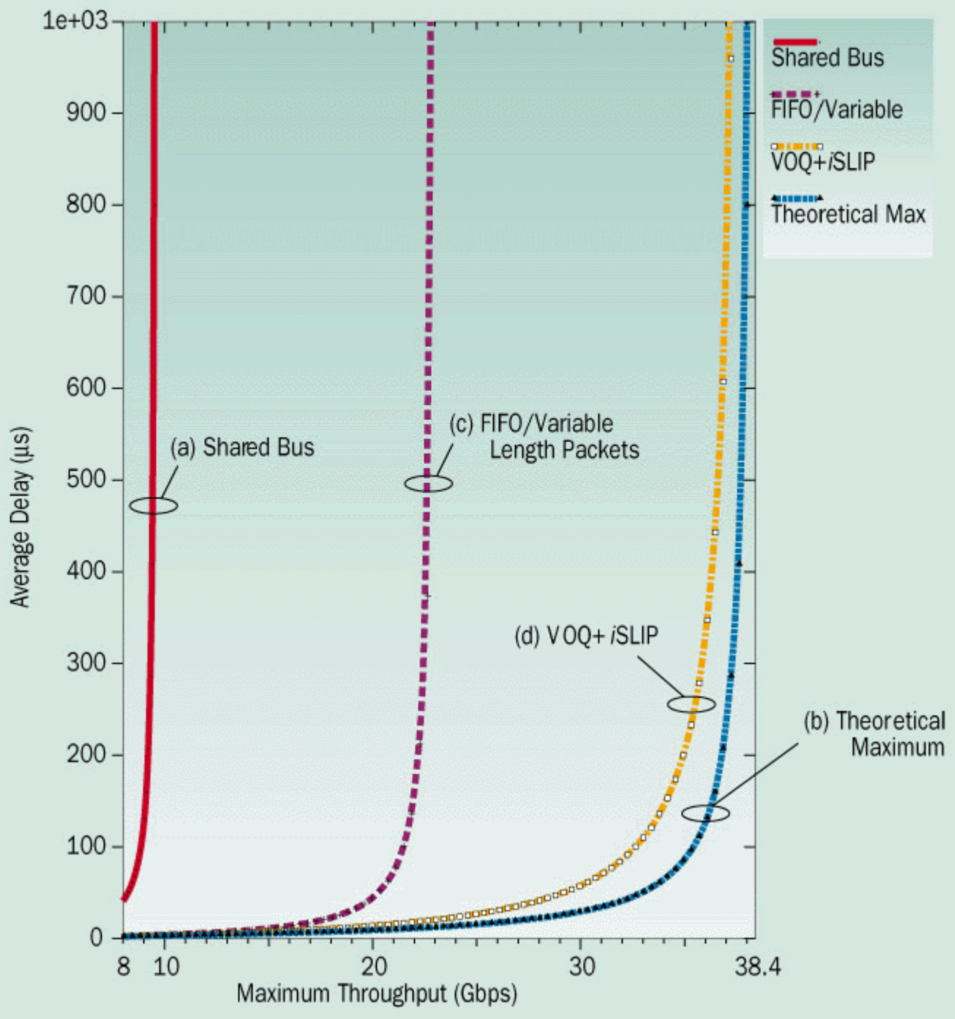
- High Throughput—For uniform and uncorrelated arrivals, the algorithm enables 100 percent of the switch capacity to be used.
- Starvation Free—No connection is starved. Because pointers are not updated after the first iteration, an output will continue to grant to the highest-priority requesting input until it is successful. Furthermore *i*SLIP is in some sense fair: with one iteration and under heavy load, all queues with a common output have identical throughput.
- Fast—The algorithm is guaranteed to complete in at most *N* iterations (where *N* is the number of ports). However, in practice the algorithm almost always completes in fewer than $\log_2 N$ iterations. For example, for a switch with 16 ports, four iterations will suffice.
- Simple to implement—An *i*SLIP scheduler consists of 2*N* programmable priority encoders. A scheduler for a 16-port switch is readily implemented on a single chip.

To benefit from these properties, our switched backplane will use VOQs and the /SLIP scheduling algorithm.

Top of Page

5 Unicast Traffic: Performance Comparison

FIGURE 7 Comparing Performance of Various Switched Backplanes for a 16 x 16 Router, with Each Port Operating at 2.4 Gbps



Note: The traffic is benign: random, but uniform traffic is applied to each input of the router. The graph plots aggregate traffic load at all the ports against packet delay, (in milliseconds). In each case, there is a traffic load that saturates the router, and the packet delay grows without bound. Lines (a) and (d) show the two extremes: line (a) is for a conventional router with a shared bus operating at four times the line rate. This router can support a total aggregate traffic load of only 9 Gbps. Line (b), on the other hand, represents the maximum theoretical throughput achievable by any router. Inbetween these two extremes lie two alternative crossbar designs. Line (c) shows what happens if we use a crossbar switch, but with variable-length packets and FIFO queues. The performance is limited to about 22 Gbps, and is well short of the theoretical maximum. But if we use fixed-length packets (cells), VOQs and the *i*SLIP scheduling algorithm, line (d) shows that the throughput comes close to the maximum achievable: 38.4 Gbps.

It is worth pausing to consider our choices so far, and their impacts on router performance. Figure 7 compares the performance of a shared bus, a crossbar switch with variable-length packets plus FIFO queuing, and a crossbar switch with cells, plus VOQ and *i*SLIP, against the theoretical maximum of any backplane. It plots the relationships between the average delay seen by cells waiting to traverse the backplane (*y* axis) and the amount of arriving traffic (*x* axis).

For each type of backplane, there is an offered load at which the system saturates: delay increases, the backplane becomes congested, and packets must be dropped. For example, the shared-bus alternative (line *a* in Figure 7) saturates at about 9.6 Gbps, when the offered load at each of the sixteen 2.4-Gbps input ports is just 600 Mbps. In contrast, the switched backplane with VOQs and the *i*SLIP scheduling algorithm (line *d* in Figure 7) saturates at 38.4 Gbps, when each 2.4 Gbps input line is fully loaded.

Note the reduced performance of the crossbar switch with variable length packets and FIFO queuing (line *c* in Figure 7). In this case, the system saturates at about 22 Gbps. Put another way, about 15 Gbps is wasted due to HOL blocking.

Top of Page

6 Controlling Delay

Until now, we have concentrated on how to overcome HOL blocking and increase the system throughput from 60 to 100 percent of the available switching bandwidth. While this is a major improvement over existing switched backplanes, we must also address the delay problems caused by input and output blocking. Controlling delay is particularly important for routers that handle multimedia (e.g., voice and video) traffic.

First, let's revisit the difficulties associated with controlling packet delay in a typical router. When a packet is delivered across the switched backplane to its outgoing port, an output-link scheduler determines the exact time it will be transmitted on the outgoing line. If, however, the time it takes for the packet to reach the outgoing port is unpredictable and uncontrollable, then the output link scheduler cannot precisely schedule the packet's departure time.

As we shall see, both input- and output-blocking make a packet's (or a cell's) delay unpredictable. To understand how input-blocking occurs, consider again the VOQ switch illustrated in Figure 6. At the end of each cell time, the *i*SLIP scheduler selects one VOQ to be served at each input. Other non-empty VOQs at the same input must wait until a later cell-time to receive service. In fact, it is very difficult to predict exactly when a non-empty VOQ will receive service. This is because it depends on the occupancy of other VOQs at the same input. The VOQ must contend for access to the crossbar switch with other VOQs that may block it for an unpredictable number of cell times.

Output-blocking occurs because each output line from the crossbar switch can transfer only one cell at a time. Consider two cells at different inputs that are both waiting to be transferred to the same output. Only one cell can be transferred at a time, while the other cell will be blocked until a later cell time. As with input-blocking, output-blocking makes it difficult to predict when a cell will be delivered to its output.

In our switched backplane, we use two techniques to better control the delay of cells. The first technique is to separate them into different priority classes according to their urgency, create separate VOQs for each output/priority pair, and give preferential access to the crossbar switch to the higher-priority cells. Although this does not reduce the total amount of input- or output-blocking, it prevents low-priority cells from affecting the delay of high-priority cells. In essence, high-priority cells "see" a less-congested switch—a high-priority cell can be blocked only by another high-priority cell.

This simple prioritization technique effectively meets our goal: to provide a small, controlled delay for high-priority traffic. As shown in Figure 7, a lightly loaded VOQ switch delays cells by a negligible amount. A reservation protocol such as RSVP can be used to limit the amount of high-priority traffic entering the switch, so that high-priority cells will experience negligible delay. Once they reach the output port, their departure time can be strictly controlled to meet the quality of service requirements.

Prioritization works well when the amount of high-priority traffic is kept small. However, with the fraction of multimedia traffic growing in the Internet, we would like to offer multiple priority classes and be able to control the delay in each class. This is why we use our second technique: speedup.

Speedup is a common way to reduce input and output blocking, by running the crossbar switch faster than the external line rate. For example, if we operate the crossbar switch twice as fast as the external line, we can transfer two cells from each input port, and two cells to each output port during each cell time. In this case, we say that the switch has a speedup of two.

The advantage of speedup is obvious: delivering more cells per cell time reduces the delay of each cell through the switch. In fact, with sufficient speedup, we can guarantee that every cell is immediately transferred to the output port, where its departure time can be precisely scheduled.

So why not determine how much speedup is needed, and always run the backplane at that speed? Unfortunately, if we want to ensure that no packets are ever queued at the input, it is generally believed to require a speedup of N where N is the number of ports. Moreover, it is assumed that that the queue memories and the links connecting the memories to the crossbar must also run N times faster.

Of course, this would be totally impractical for high-performance routers: their performance is invariably limited by the availability of fast memories. Significant speedup is either uneconomical, or simply impossible. For example, if the Cisco 12000 were to have a speedup of N the memory bandwidth at each output port would have to increase from 2.4 Gbps to 38.4 Gbps. Even if the memories were arranged to be one cell wide, the memory devices would have to perform a read or write operation every 5ns (possible, but very expensive).

However, recent research suggests that a crossbar switch with a speedup of two behaves almost identically to a crossbar switch with a speedup of N independently of N (see I. Iliadis & W. E. Denzel, "Performance of packet switches with input and output queueing," *Proceedings of Infocom*, 1990, pp. 747-753). This has been confirmed quite recently with the following results: (a.) If a crossbar switch maintains only a single FIFO queue, then a speedup of N is required for the cells to experience predictable delay. (b.) If VOQs are used, then a speedup of just four will suffice (see Stanford University Computer Systems Labs Technical Report CSL-TR-97-738, "On the speedup required for combined input and output queued switching," by B. Prabhakar & N. McKeown).

In fact, results suggest that it will soon be possible to precisely control delay through a switch with VOQs and a speedup of just two.

Top of Page

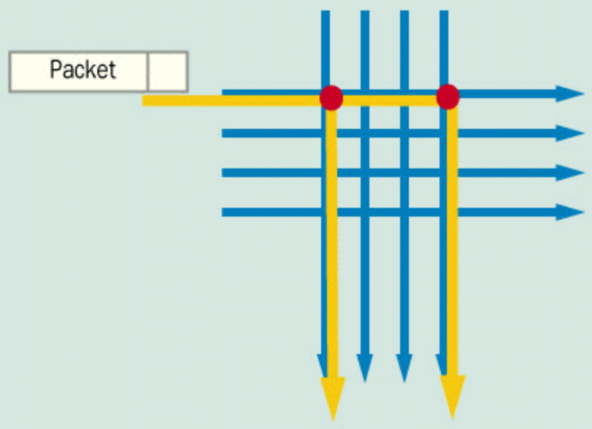
7 Supporting Multicast Traffic

While we have focused thus far on how a switched backplane can efficiently transfer unicast packets, it is becoming increasingly important for a high-performance router to efficiently support multicast traffic. As we will see, we can achieve very high performance using the same crossbar switch, and a modified version of the *i*SLIP scheduling algorithm, called ESLIP.

A simple way to handle multicast traffic is to replicate the input cell over multiple cell times, generating one output cell per time slot. However, this approach has two disadvantages. First, each input must be copied multiple times, increasing the required memory bandwidth. Second, input cells contend for access to the switch multiple times, reducing the bandwidth available to other traffic at the same input.

Higher throughput can be attained if we take advantage of the natural multicast properties of a crossbar switch. So instead, we will design our system to copy an input cell to any number of idle outputs in a single cell time.

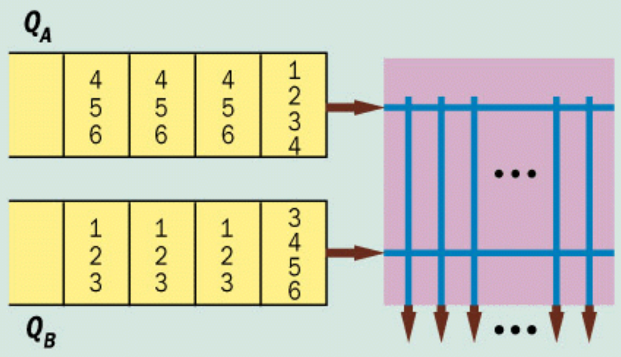
FIGURE 8 Sending Multicast Packets to Multiple Outputs Simultaneously over a Crossbar Fabric



Note: In a crossbar fabric, multicast packets can be sent to multiple outputs simultaneously by "closing" multiple crosspoints at the same time. This example shows a 4-input switch forwarding a multicast packet to two destinations.

In our combined switch, we maintain two types of queues: unicast cells are stored in VOQs, and multicast cells are stored in a separate multicast queue. By closing multiple crosspoints simultaneously, the crossbar switch essentially performs cell replication within the fabric, as shown in Figure 8. As before, at the beginning of each time slot, a centralized scheduling algorithm decides which crosspoints to close.

FIGURE 9 A 2 × *N* Crossbar Switch that Supports Multicast

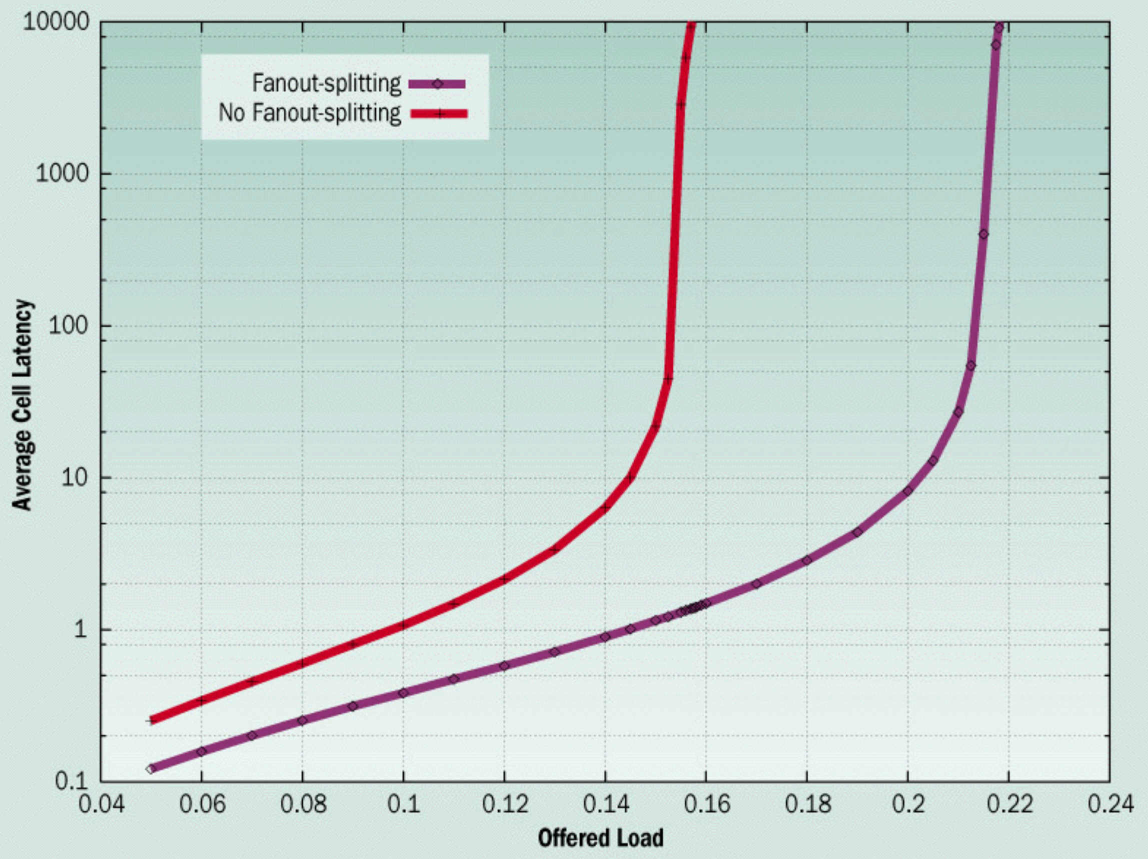


Note: Each input maintains a special FIFO queue for multicast cells.

Figure 9 shows how a crossbar switch can support multicast. The set of outputs to which an input cell wishes to be replicated, or copied, is called the fanout of that input cell. In the figure, Queue subscript *A* has a fanout of four—outputs {1,2,3,4}. For practical reasons, an input cell waits in the multicast queue until all of the cells ahead of it in that queue have departed.

7.1 Scheduling Multicast Traffic: There are two different service disciplines that we can use to schedule multicast traffic. The first is called no fanout-splitting, in which all the copies of a cell must be sent in the same cell time. If any of the output cells loses contention for an output port, none of the output cells are transmitted and the cell must try again in the next cell time. The second discipline is called fanout-splitting, in which output cells may be delivered to output ports over any number of cell times. Only those output cells that are unsuccessful in one cell time continue to contend for output ports in the next cell time.

FIGURE 10 Average Cell Latency with and without Fanout Splitting

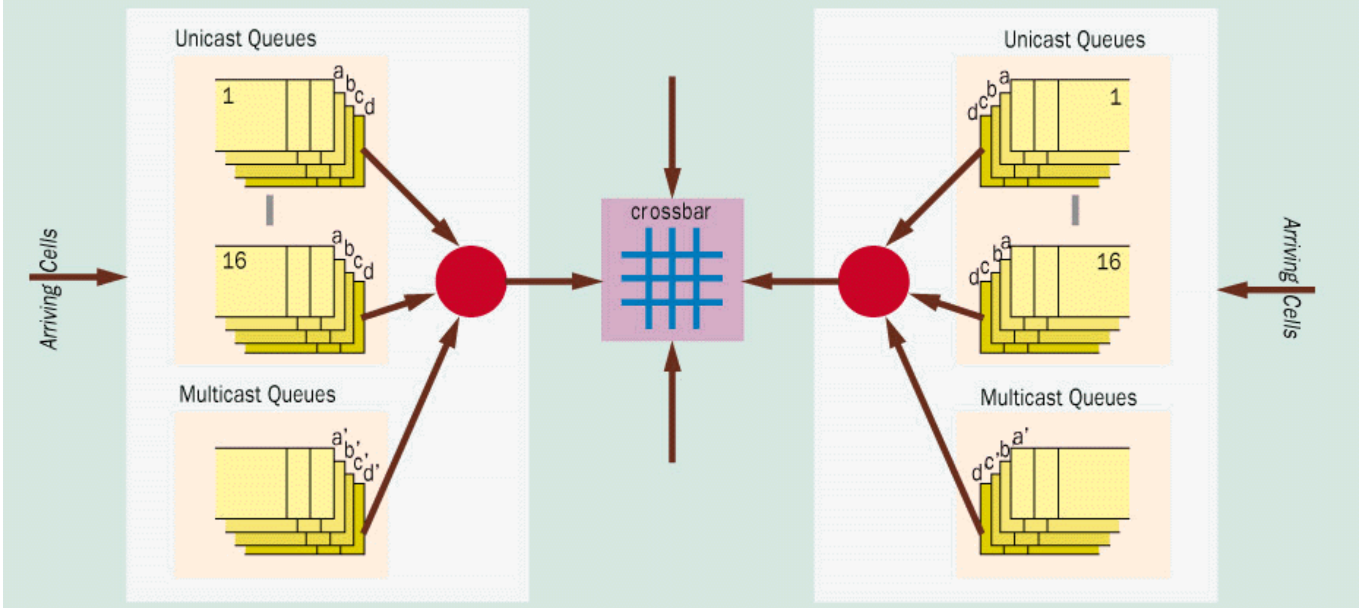


Note: Graph depicts average cell latency (in number of cell times) as a function of offered load for a 8-port switch (with uniform input traffic and average fanout of four), with and without fanout-splitting.

Research has shown that fanout-splitting enables a higher switch throughput for little increase in implementation complexity (see R. Ahuja, B. Prabhakar & N. McKeown, "Multicast scheduling for input queued switches," *IEEE Journal of Selected Areas in Communications*, June 1996.) For example, Figure 10 compares simulated average cell latencies with and without fanout-splitting, and shows that fanout-splitting can lead to approximately 40 percent higher throughput. Because of its simplicity, and performance benefits, we use fanout-splitting in our multicast scheduling algorithms.

7.2 ESLIP: Combining Unicast and Multicast: To support multicast, we need a new scheduling algorithm—one that can examine the contents of the multicast queues at each input and select which output cells will be delivered in each time slot. Furthermore, the algorithm needs to choose between competing unicast and multicast cells, each with its own priority. For this purpose, we have devised a novel algorithm, ESLIP, an enhanced version of the unicast *i*SLIP algorithm.

FIGURE 11 Queues Maintained for a 16-port Unicast-Multicast Switch



Note: The queues maintained by each input for a 16-port combined unicast-multicast switch, with four priority levels {a,b,c,d}. Each input maintains 16 VOQs for unicast traffic and 1 for multicast traffic, for each priority level. Hence, there are 68 queues per input.

ESLIP efficiently schedules unicast and multicast traffic simultaneously and is able to provide preferential service to any cells with higher priority. The queues maintained by a 16-port switch supporting four priority levels are illustrated in Figure 11. At the beginning of each time slot, the ESLIP scheduler examines the contents of all the queues. Like *i*SLIP, it is an iterative algorithm: all inputs and outputs are initially unmatched, with each successive iteration adding additional connections. To keep track of which connection to favor next, each output maintains a separate grant pointer for each priority level.

The ESLIP algorithm has the following features:

- High Throughput—For unicast traffic, ESLIP performs as well as *i*SLIP, and hence for uniform and uncorrelated unicast arrivals, the algorithm enables 100 percent of the switch capacity to be used. Because fanout-splitting is used for multicast traffic, high throughput is possible. In addition, the global multicast pointer helps clear backlog quickly—all of the outputs tend to favor the same input, reducing the need to split fanout.
- Starvation Free—No connection is starved.
- Fast—The algorithm is guaranteed to converge in at most N iterations (where N is the number of ports). However, in practice the algorithm generally converges in fewer than $\log^2 N$ iterations—i.e. for a switch with 16 ports, four iterations will suffice. Although the algorithm will not necessarily converge to a maximum sized match, it will find a maximal match: the largest size match without removing connections made in earlier iterations.
- Simple to implement—Like *i*SLIP, an ESLIP scheduler consists of $2N$ programmable priority encoders. A scheduler for a 16-port switch can readily be implemented on a single chip. The Cisco 12000 router uses the ESLIP algorithm to schedule unicast and multicast cells across its backplane.

Top of Page

8 Concluding Remarks

Using the Cisco 12000 as an example, we have seen that a switched backplane based on a crossbar switch offers significant performance advantages over a conventional router with a centralized shared bus. Furthermore, we can expect this architecture to scale well into the future.

However, care must be taken when designing a switched backplane to achieve high throughput and predictable delay. We can summarize our design choices as follows:

- Use Fixed-Length Packets: Using fixed length packets ("cells") allows up to 100 percent of the switch bandwidth to be used for transferring cells. If variable-length packets are used, the system throughput is limited to approximately 60 percent of its theoretical maximum.
- Use Virtual Output Queuing: If VOQs are used, instead of the conventional FIFO queues, then head of line (HOL) blocking can be eliminated entirely. The combination of fixed-length packets and VOQs causes this increase.
- Use Priority Levels and Speedup: Priority levels can make packet delay more predictable, allowing the router to control delay for multimedia traffic. If VOQs and a moderate speedup are employed, then the delay can be carefully controlled.

- Use Separate Multicast Queues: If multicast traffic is queued separately, then the crossbar may be used to replicate cells, rather than wasting precious memory bandwidth at the input.
- Use Fanout-Splitting for Multicast Packets: If the crossbar implements fanout-splitting for multicast packets, then multicast traffic throughput can be increased by 40 percent over the no-fanout discipline.
- Use a Combined Unicast and Multicast Scheduler: If a single scheduler is used to schedule both unicast and multicast traffic, then: (a.) the scheduling decisions can be made at greater speed, and (b.) the relative priority of uniciast and multicast traffic can be maintained, preventing either type of traffic from starving the other.

Top of Page

9 References

[1] W. Doeringer, G. Karjoth, M. Nassehi, "Routing on Longest-Prefix Matches," *IEEE Transactions on Networking,* vol. 4 no. 1, Feb 1996, pp. 86-97.

[2] C. Partridge, P. Carvey, et al. "A Fifty-Gigabit per Second IP Router," Submitted to *IEEE/ACM Transactions on Networking,* 1996.

[3] A. Singhal, et al., "Gigaplane (TM): A High Performance Bus for Large SMPs," *Proc. of Hot Interconnects '96,* Stanford, CA..

[4] M. Karol, M. Hluchyj, S. Morgan, "Input versus output queueing on a space-division switch," *IEEE Transactions on Communications,* vol. 35, Dec 1987, pp. 1347-1356.

[5] Y. Tamir, G. Frazier, "High Performance Multiqueue Buffers for VLSI Communication Switches," *Proc. of 15th Annual Symp. on Computer Arch,* Jun 1988.

[6] N. McKeown, V. Anantharam, J. Walrand, "Achieving 100% Throughput in an input-queued switch," *Proceedings of IEEE Infocom '96*.

[7] T. Anderson, et al. "High Speed Switch Scheduling for Local Area Networks," *ACM Trans. on Computer Systems*, Nov 1993, pp. 319-352.

[8] N. McKeown, "Scheduling Cells in an input-queued switch," PhD Thesis, University of California at Berkeley, May 1995.

[9] N. McKeown, "*i*SLIP: A Scheduling Algorithm for Input-Queued Switches," Submitted to *IEEE Transactions on Networking*.

[10] N. McKeown, et al. "The Tiny Tera: A small high-bandwidth packet switch core," *IEEE Micro*, Jan-Feb 1997.

[11] S. Deering, "Multicast Routing in a Datagram Internetwork," Ph.D. Thesis, Stanford University, CA, 1991.

[12] R. Ahuja, B. Prabhakar, et al. "Multicast Scheduling for Input-Queued Switches *IEEE JSAC*, vol. 15, no. 5, Jun 1997, pp. 855-866.

[13] H. Ahmadi, W. Denzel, "A Survey of Modern High-Performance Switching Techniques," *IEEE JSAC*, vol. 7, no. 7, Sep 1989, pp. 1091-1103.

[14] A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *J Internetworking: Research and Experience*, 1990, pp. 3-26.

[15] L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks," *ACM Trans on Computer Systems*, 1990, vol. 9, no. 2, pp. 101-124.

[16] D. Clark, S.Shenker, L. Zhang, "Supporting Real-Time Applications in an Integrated Services Packet Network: Architectures and Mechanisms," *Proc. Sigcomm '92*, Aug 1992.

[17] H. Zhang, "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks," *Proc. of the IEEE*, vol. 83, no. 10, pp. 1374-1396.

[18] A. Brodnik, S. Carlsson, M. Degermark and S. Pink, "Small Forwarding Tables for Fast Routing Lookups," *Proc. Sigcomm '97*, France, Sep 1997.

[19] M. Waldvogel, G. Varghese, J. Turner and B. Plattner, "Scalable High Speed IP Routing Lookups," *Proc. Sigcomm '97*, France, Sep 1997.

[20] S. C. Lin, P. Gupta, N. McKeown, "Fast Routing Lookups in Hardware," submitted to *IEEE Infocom '98*. Preprint available on request.

[21] S. C. Lin and N. McKeown, "A Simulation Study of IP Switching," *Proc. Sigcomm '97*, France, Sep 1997.

[22] Y. Tamir and H-C. Chi, "Symmetric Crossbar Arbiters for VLSI Communication Switches," *IEEE Trans on Parallel and Dist Systems*, vol. 4, no. 1, 1993, pp. 13-27.

[23] R. O. LaMaire, D. N. Serpanos, "Two-dimensional round-robin schedulers for packet switches with multiple input queues," *IEEE/ACM Transactions on Networking*, Oct 1994, vol. 2, no. 5, pp. 471-82.

[24] C. Lund, S. Phillips, N. Reingold, "Fair prioritized scheduling in an input-buffered switch," *Proc. IFIP-IEEE Conf. on Broadband*

Communications '96, Montreal, Apr 1996. pp. 358-69.

[25] J. Hui, E. Arthurs, "A broadband packet switch for integrated transport," *IEEE J Selected Areas Communications*, vol. 5, no. 8, Oct 1987, pp. 1264-1273.

[26] M. Ali, H. Nguyen, "A neural network implementation of an input access scheme in a high-speed packet switch," *Proc Globecom 1989*, pp. 1192-1196.

[27] ATM Forum, "ATM User-Network Interface (UNI) Signalling Specification," Version 4.0, April 1996.

[28] J. E. Hopcroft, R. M. Karp, "An algorithm for maximum matching in bipartite graphs," *Soc for Industrial and Applied Mathematics J Comput*, 2 (1973), pp. 225-231.

[29] J. F. Hayes, R. Breault, and M. Mehmet-Ali, "Performance Analysis of a Multicast Switch," *IEEE Trans Commun*, vol. 39, no. 4, Apr 1991, pp. 581-587.

[30] J. Y. Hui, and T. Renner, "Queueing Analysis for Multicast Packet Switching," *IEEE Trans Commun*, vol. 42, no. 2/3/4, Feb 1994, pp. 723-731.

Top of Page