ISIS: The Next Generation Router Architecture, Design and Simulation

Shahzad Ali, Xia Chen, Yu Zhong, Brendan Howell

***Abstract*—High-speed routers have become the new bottleneck in the Internet with the growth of the bandwidth through fiber technology. Switching technology has not grown at the same speed as transmission technology. With the introduction of Dense Wavelength Division Multiplexed networks, the demands on the switch have increased. Also, there is a need for high port density on these routers with a consideration of cost and performance. With all attention being paid to high speed and high port density, we have almost forgotten about the rich variety of services that are to be supported on this network. This constraint requires Quality of Service (QoS) guarantees that make the design much more difficult. In this paper, we look at the design, simulation and evaluation of a high speed router switch ISIS and show that it is possible to make routers with high port density while still meeting the diverse Quality of Service (QoS) requirements that are demanded of routers today.**

***Index Terms*—high-speed router, iSlip, Quality of Service, simulator.**

I. Introduction

HIGH speed routers have been a bottleneck for high speed networks. While transmission strategies have progressed from using copper wires to fibers that carry orders of magnitude more bandwidth. The technology has not stopped at that and there are new mechanisms being developed that allows for more bandwidth to be packed into a single fiber. With the advent of multi-mode fibers, the bandwidth has almost kept up with the demands that today's Internet imposes on the network. Starting from the standard optical transmission strategies and moving towards Optical Time-division multiplexing (OTDM) and then to Dense wavelength-division multiplexing (DWDM), the transmission speeds have reached terabits on a single fiber. However, with all of these advancements, there has been a serious concern as to the switching capability within the network. The switching technology has fallen behind in the bandwidth race and we are still trying to achieve gigabits per second speeds with routers. However, it will not be fair to say that there have not been any improvements in this area. It is more appropriate to say that the technologies being used have not changed drastically to meet the challenges of high bandwidth.

Despite the various technologies that have been incrementally developed and deployed, the Internet Protocol (IP) has been the standard that has remained almost constant in about two decades. Its wide-spread deployment and high usage in the form of the World Wide Web has made IP the most prevalent form of network layer protocol deployed in the World. This does not restrict the services that can be deployed over the IP network. Most of these services are quite different from the best effort services that are provided by the generic IP protocol. Thus, the IP networks of today have to keep up with the demands imposed by new services that require the network to provide certain Quality of Service guarantees.

In this paper, we addressed the problem of making high-speed routers without losing the advantages of providing QoS support for new applications. We designed our IP-router, called ISIS with these constraints in mind and simulated and evaluated its performance against other designs that do not provide QoS guarantees. We observed that it is possible, with a little bit of extra effort, to produce routers that are as fast as other designs while also providing significant QoS support to applications and services.

We will start by listing down the goals that led us through the entire design process. We will then go on to explain the various components in the system as well as some key issues that we tackled in the design.

II. Goals

There are many equally important characteristics that a router should have if it has to be successful. We decided the important aspects that we wanted to concentrate on by deciding on where the router will be placed. As we are trying to meet a challenge of high bandwidth, we naturally assumed that the router would be placed as close to the core of the network as possible. This gave us the following placement strategies.

1. Aggregation point for large data centers
2. Backbone router at a major peering point
3. Backbone router for carrier facilities
4. Core router for IP transit providers

The characteristics that a router needs to have to be placed

Submitted on May 12, 2000 as a course project report for Broadband Networks (18-757) course.

Shahzad Ali is a graduate student in the Electrical and Computer Engineering Department , email: cheeko@cs.cmu.edu

Xia Chen is a graduate student in the Physics Department, email: xiachen@andrew.cmu.edu

Yu Zhong is a graduate student in the Information Networking Institute, email: yuzhong@andrew.cmu.edu

Brendan Howell is an undergraduate student in the Electrical and Computer Engineering Department, email: bch@andrew.cmu.edu

in such an environment are very demanding. We have listed down some of these properties that we would like our design to have.

A. *High bandwidth*

The first challenge in this environment is the high bandwidth that the router needs to support. These include speeds up to 10Gb/s that are not commonly available at the moment, but the transmission technology for them has been developed for quite some time. The line speeds that we aimed for were OC-48 that can have data rates of up to 2.48Gb/s. We also wanted our design to be useful for OC-192 speeds with slight modifications although that was not one of our initial goals.

B. *High Port Density*

Another challenge that switching technology faces is the port density that has to be provided on these routers. With the number of pipes that ISP's need to provide to their customers increasing, the routers have to have a huge number of ports to accommodate this demand. Switching is usually not scalable with the number of ports and also suffers from the physical constraints of Printed Circuit Boards (PCBs) that are used to build this technology. Port Density together with standard router sizes that will fit in TELCO (telecommunication companies) racks mean that the overall capacity of the router should be high, in the range of hundreds of OC-48 ports. As a base design goal we went with 64 ports on the router that gave the router a based capacity of 160Gb/s. This is more than the current capacity that is available on routers used by ISPs. Cisco [4], for example, has the GSR-12000 series routers that can have up to 80Gb/s in one router. They do however, provide configurations with multiple boxes that can have more than that aggregate capacity. Juniper [5], the other major player in the market, has a 40Gb/s router, the M-40 and recently announced plans for a 160Gb/s router named M-160 [6]. This shows that our design is at worst comparable to the current industry leaders.

C. *Performance*

Performance is also a key issue during the design of any such router. The performance has direct implications for the usability of the router in a real network. It has to have minimal delays incurred due to packet processing and switching so as to let the flows through it as seamlessly as possible. The throughput of such a router has to remain high with different traffic conditions and under various load characteristics. The packet drop rate on the router is also an important measure of performance. Recent technologies have allowed routers to provide support during congestion in the network and this was one of the goals of our design as well.

D. *Cost*

Cost of the router has to be controlled so that a solution can be provided that does not cost an arm and a leg for the ISP or carrier. Cost can be controlled by using generic off-the-shelf components that have reduced cost and by reducing the complexity of the system so that the effort that goes into the development process can be controlled.

E. *Robustness*

A router that is being placed so close to the core of the network has to have tremendous robustness. This means that the router should be able to operate under all conditions. This includes external environment plus internal problems. The robustness features are built into the design of ISIS so that redundancy is provided at all possible levels and the router can work in critical component failure scenarios as well.

F. *Major Routing Protocol Support*

A router is only useful if it has excellent software running on it. Although this design project did not consider software issues at all, we include the support for major protocols here for completeness sake. The router has to support unicast routing protocols such as OSPF, RIP, IS-IS and BGP and multicast routing protocols such as IGMP, DVMRP, PIM-DM, PIM-SM, MBGP and MSDP. This is in addition to debugging utilities, management interfaces and configuration support that is provided on the router.

G. *Quality of Service*

As mentioned earlier, we decided to include QoS support from the router into our design. Strict QoS support implies hard guarantees for packets belonging to a particular flow on delay and loss. This involves very strict priorities and can lead to degradation of overall system performance. So we included as a design goal, the provision of maximum QoS support without forgoing the capacity of the system. As it turns out, it is possible to provide such a support in ISIS.

H. *Scalability*

And last but not the least, we want the design on ISIS to be scalable. The design of ISIS was done to make the capacity high without overhauling the entire system. By adding new components, the system can be made incrementally scalable and the overall capacity can be increased in simple steps. This makes the design very suitable for the Internet where bandwidth needs double every eight months or so and thus providers can keep increasing the capacity of ISIS until the limit is reached. Our base design provides 160Gb/s capacity but it can be scaled up in steps for terabits per second capacity as well.

I. *Realistic Design*

Another important goal that is worth mentioning here is that we decided to make the design of ISIS as realistic as possible. As this design will only be tested with a simulator, it is very easy to forget the physical constraints that are present and the standards that need to be followed and end up designing a box that is almost impossible to produce. We made sure we kept in line with all industry standards and used components that are available so that the final product can be produced, if needed.

III. Overview

With the goals that are listed above, we set out to design a router. We used either off-the-shelf components or simple ASICs for the design. We also simulated and evaluated the performance of the various schemes that we designed.

The rest of the paper is laid out as follows. We will talk about the overall router design in terms of the hardware components needed. We will, then talk about the key issues in the design that need special attention and discuss these in more detail. We also designed and implemented a simulator to simulate the router. We will discuss the structure of the simulator. We will present results of our evaluation and give an explanation of the results. We will also give out an outline of the physical specification of ISIS and talk about the layout of the various components in the router. Finally, we will try to tackle the issue of scalability and present our architecture for ISIS-A, the scalable version of ISIS. In the end, we will conclude with some recommendations and ideas for future work in this area.

IV. Overall design

A. *Specification*

As mentioned earlier, we used a base design of 64 OC-48 ports on the router, giving an overall capacity of 160Gb/s. This was our base design and towards the end we will talk about how we made the design flexible. Apart from that, we also allowed for a Weight Fair Queuing mechanism to be implemented for the switch to provide for some QoS support. However, we soon figured out that this support is not enough for our design goals and decided to look deeper into this issue.

Apart from these standard concerns, we also tried to make the design as realistic as possible. We looked at the physical constraints that our design had and worked with those in mind.

B. *Components*

We designed our minimum specification box with 64 ports as made up of various components. These included

1. **Line cards**
2. **Chassis**
3. **Switching Fabric**
4. **Scheduler**

We have 4 ports per line card and thus have 16 line cards in the entire box. Given that our line speed is OC-48 that implies a total traffic rate of 10Gb/s per line card. The line cards each have the necessary components to handle the processing of packets as well as scheduling decisions through the switch. The line cards also have buffer space that is used to store packets as they are being routed through the switch. The chassis is a standard 19-inch rack mountable chassis with redundancy features built into it. . The switching fabric that we used is a 320Gb/s chipset by PMC Sierra [3] that fits our design specifications perfectly. This chip set provides us with all the features that we needed for our design while being suitable for the fast configuration times that we required at OC-48 speeds. During our design process, we looked at a number of other options in terms of the switching fabric but were always met with a situation where the chip set was not going to fulfill all our needs. In this process, we looked at Vitesse [8], AMCC[7] and TriQuint [9].

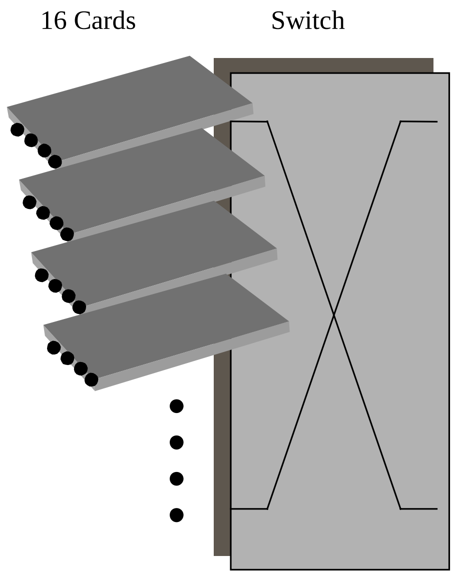


Figure 1: The entire layout of the router. This includes the 16 line cards each with four ports of OC-48 lines and the switching fabric on a back plane

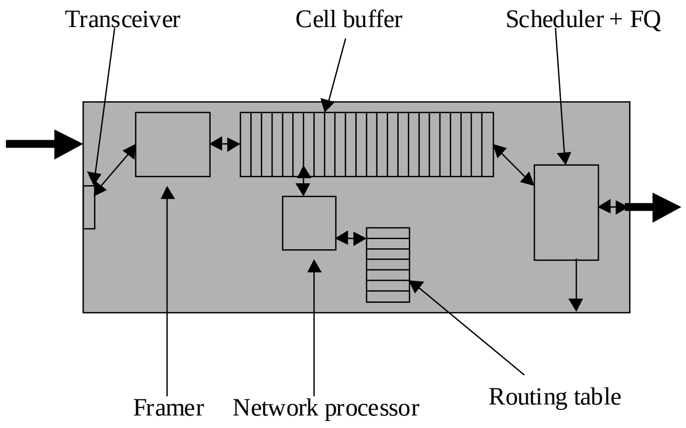


Figure 2: Components and functions that need to be performed per port on each line card. There will be four such modules in one line card. This picture only shows modules for one port

Figures 1 and 2 show the overall layout of our switch. There are 16 line card modules that plug into the switching fabric, each having a total traffic rate of 10Gb/s. The back plane that we used has 16 10Gb/s channels and can thus be connected to each one of the line card modules easily. Later, we will talk about how to make this design extensible by using these 10Gb/s channels.

Each port needs a transceiver that performs the physical layer processing. This includes synchronization of the SONET clock and other physical level features.

Once the physical level processing is finished and an IP packet is received, the packet is stored in a cell buffer. This buffer is a fast DRAM so that packets can be written and read at the speeds that they are being received by the transceiver. The destination of the packet has to be looked up in a table to figure out where to send the packet. This process is called IP lookup and happens at every packet time interval. Whenever a packet is received, its address needs to be looked at and its destination checked in the routing table. Therefore, we need a processor for this purpose. There are many choices available in the market for these processors, including generic off-the-shelf processors and sophisticated network processors. We selected the Intel IXP 1200 [10] network processor that is capable of doing IP lookup for about 4Gb/s of data. This is greater than our line speed of 2.48Gb/s and thus is a good choice for our implementation. However, we need four such processors on a single line card. Given the low cost of these processors, we think the cost effectiveness of using the IXP 1200 against using a generic processor is much higher.

ISIS is an IP router that means that the packet sizes are variable. However, the back plane works best when the sizes of the packets that it is serving at fixed. Therefore, we decided to fragment the packet into fixed size cells. We chose a cell size of 64 bytes, with an overhead of 6 bytes. These parameters were chosen due to the fact that the minimum size IP packets in the network at around 40 bytes and thus would fit into one cell. However, this does incur an extra overhead whenever the packets are broken into a cell. These cells are then collected at the output and recombined to make the packet before sending the packet out.

Other packet processing functions include the update of the TTL value and discarding the packet if the TTL goes to zero. The checksum on the IP packets need also be checked and bad packets are dropped. This processing is done with specialized instructions in the IXP-1200 instruction set and can thus be performed at very high speeds.

After the IP processing, the packet needs to be switched to the output port it is destined for. This is done through a scheduler that is a distributed algorithm implemented as an ASIC on each line card. This turns out to be the bottleneck in the capacity of the entire switch. The many constraints that need to be placed on the scheduler make its implementation very hard and reduce the entire capacity of the switch. Another problem that is observed in our design, with the scheduler is that the distributed nature of the scheduler makes it impossible to make QoS decisions at the input. As we will see later, this is a crucial component of our system and we had to spend considerable amount of time on it. However, in the end we have a solution that meets all our goals and is realistic as well.

C. *Other details*

There are some other very important details that we have not addressed in this project. These include the control processor support and the routing protocol support that is to be provided in any successful IP router. The switching fabric has buffers that are usually managed as FIFO. However, there have been results to show that a FIFO discipline is not really fair. We have looked at the implementation of Random Early Drop (RED) [11] for the buffers, but have not evaluated this discipline any further. We believe that such a discipline is necessary for ISIS.

We also realize that routing table look up has traditionally been a barrier to high speed routing. Thus it would make sense to have a scheme based on Multi-protocol Label Switching (MPLS) as such a scheme can be implemented in hardware at a very high speed.

We have also ignored multicast traffic for this project. However, it is clear that such traffic needs to be handled. The effect of multicast traffic will make our results somewhat different, however, not significantly.

V. Queuing Decisions

It is evident from the overall design in the last section, that packets that have been fragmented into fixed size cells can not be sent through the switch as soon as they are available and thus they need to be stored in a buffer. We will look at the buffering schemes as well as some basics about why they need to be buffered in the first place.

A. *Contention*

Packets (cells) destined to a particular output port can sometimes not get there in one time slot. To see why this can be the case, lets look at an extreme example, where all input ports are sending packets to one output port. The output port and the output line can only handle the traffic that is being generated by one input port and thus the switch needs to buffer all the traffic that is being sent by other input ports. It is a crucial decision as to where to buffer these packets that can not go through the switch.

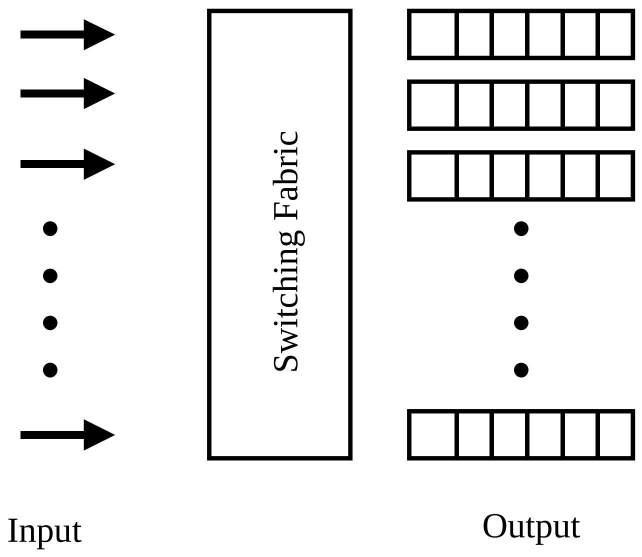


Figure 3: Output queuing switch. Fabric requires to be N times faster than the line speed to achieve this.

B. *Output Queuing*

Packets that can not be sent out from the output link in the time slot they are received can be buffered at the output. However, it is easy to see that with a fabric that is operating at the speed at which the line is operating, it won’t be possible for the fabric to send that much traffic through. In the worst case, all the input ports can be sending traffic to the same output port and thus the fabric needs to operate at N times the speed of the line where N is the number of ports on the switch. It is not hard to see that with line speeds increasing and high port densities, it is almost impossible to build a fabric that has N times the speed. This factor, N, is referred to as the speedup of the fabric. Nominal speedups such as 1.25 or 1.5 are normal. However, a speedup of 64 or 128 is absurd. This scheme also requires the output port buffers to be able to write in to them at N times the speed of the line. This makes the entire design very expensive and resistant to scalability. As one of our major concerns is scalability, this design violates that principle. The basic design of an output queued switch is shown in Figure 3.

This design however, is not all useless. If all the packets get to the output port as soon as they are received, a scheduler placed at the output link can chose whichever packets to get out on the link in arbitrary orders. This is an ideal situation for providing QoS in terms of delay bounds or priorities to packets belonging to certain types of flows. Thus output queuing helps our final design goal of providing QoS. With an output queued switch, various output scheduling decisions such as Weighted Fair Queuing (WFQ) [12], Worst-case weighted fair queuing (WF^2Q) and WF^2Q+[13], can be implemented quite easily and QoS guaranteed for certain kinds of flows. There is an overhead associated with any FQ mechanism but there are techniques present that can reduce that overhead. Most FQ techniques incur the overhead of at least one sorting operation that is an O (N) operation. Recent work by Huang and Su shows methods of reducing this overhead by clever use of a data structure called calendar queues and with some reasonable assumptions. Thus, the Output queued switch can be made ideal in terms of providing QoS for flows through the switch.

The conflicting properties of the output queuing switch make our decision difficult. We decided to look at other possibilities that may provide a better tradeoff between these design goals than an output queued switch.

C. *Input Queuing*

The other possibility that we have is to let the packets be buffered at the input side. This means that the fabric runs without a speedup and the packets that can not get through the fabric in one time slot are buffered in the input buffer. This design has certain desirable properties.

One of these properties is the simplicity in terms of hardware that is needed for building such a switch. As the fabric and the memory do not need any speedup, it is cheaper and easier to build this switch as compared to the output queuing switch. There are scheduling algorithms that have been around for switching packets through a fabric when packets are being queued at the input side. Using one of these algorithms or its modifications, it could possibly produce a switch with 100% throughput.

Another desirable property is the scalability of the design. This switch can be extended to have a higher port density without having to change the fabric. If the scheduling algorithm can handle the amount of traffic, the fabric just needs to run at the speed of the line. This is an important consideration in our design and so makes an input queuing switch a very possible choice.

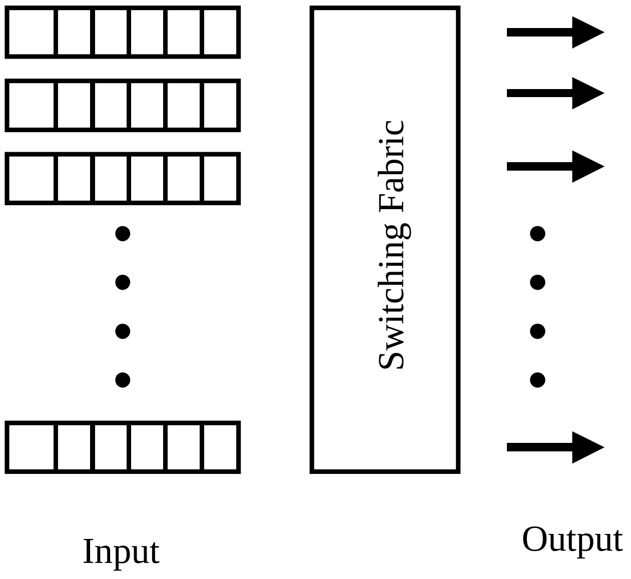


Figure 4: Input Queued Switch. There are no guarantees for QoS in this kind of a switch.

The disadvantages of a input queued switch is that it does not provide good support for QoS. When packets are queued at the input port, and scheduling is done in a distributed way, there is no way for a FQ module to decide which packets are destined for which output port for it to do some ordering between the packets. The only naïve solution is to have a central scheduler that looks at all packets at all input ports to make such a decision. It is obvious that this solution means that the switch design will not be scalable and will thus violate our initial design goal.

The design of such a switch is shown in Figure 4. Unlike the output queued switch, it is not easy to ensure that the throughput of the switch is 100%. In later sections, we will see what causes the throughput of the switch to be less than 100% and if there are any solutions to this problem.

D. *QoS with input queuing*

Having said that providing QoS with input queued switches is a tough problem, there has been some recent work done in this area and there are some mechanisms by which soft QoS guarantees can be ensured. By soft guarantees, we mean probabilistic guarantees on the delay and loss of packets from a particular flow rather than a deterministic guarantee.

Stephens present one possible solution to this problem in [14]. According to that, a switch that has input queuing can bound delay, only if it has buffers at the cross-points in a crossbar as well. The number of buffers that are needed for each cross-point in the crossbar are small, usually about 3-5 cells. This makes the design of the crossbar easy and inexpensive. The key features of the architecture are the FQ servers that are implemented at the inputs, outputs and at the crossbar buffers. With these FQ servers, it is possible to ensure a delay guarantee for flows.

This concept has also been shown to work with multiple stage switches instead of a crossbar in [15] by Chiussi and Francini who use the ATLANTA chipset for their implementation.

These show that the solution is feasible and that it does provide guarantees with input queued switches as well. However, the solution is complicated and the resulting design is expensive and has to use ASICs for a lot of the components. There is also some concern about the throughput of such a switch given that it is non-work conserving now.

E. A Third Option? Combined Input-Output Queuing (CIOQ)

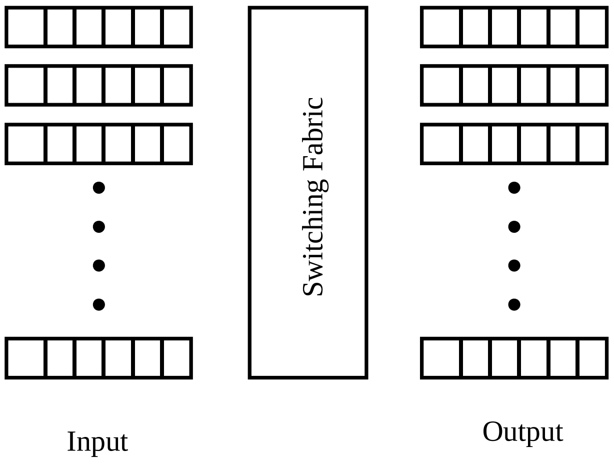


Figure 5: The buffering is needed at the inputs and the outputs of a CIOQ switch. The QoS guarantees that this switch can provide with a speedup of 2 are phenomenal.

There is a third option as far as queuing disciplines are concerned (Figure 5). Instead of queuing at the input or the output, it is possible to queue the cells at the input and the output. This means, that the fabric that is used needs to have a speedup of greater than 1 but less than N, where N is the number of ports in the switch. With a speedup of 1, we get a input queued switch and with a speedup of N we get an output queued switch. However, as we have seen that a speedup of N is not practical for high-speed switches. We have also seen that the speedup of N is the ideal situation for providing QoS support to the router. Our hope is that by using a hybrid approach, we will be able to provide some QoS support without requiring a huge speedup.

Speedup	Queuing
1	Input
N	Output
> 1 and < N	CIOQ

Table 1: Effect of speedup of the fabric on the queuing discipline used.

There has been some work done in the area of CIOQ switches to see if it is possible to provide some level of QoS with nominal speedup. [16] and [17] are two such pieces of work that show that with a speedup of 2 in the fabric, it is possible to emulate a output queued switch perfectly. This result is very promising as a speedup of 2 is very realistic and it provides QoS support that we demand. The algorithm cited in these schemes is quite complex and it is still a question whether the implementation can be done in a distributed manner without complex hardware. But if it is possible to do that, CIOQ provides a mechanism to achieve the goals for ISIS. In the next section, we will talk about all the scheduling decisions including those with various queuing mechanisms.

VI. Scheduling Decisions

There are two problems faced by the current high-speed switches simultaneously: a need for a faster switching/routing infrastructure and a need to introduce guaranteed qualities of service (QoS). Each problem can be solved independently: switches and routers can be made faster by using input-queued crossbars, instead of shared memory systems; and QoS can be provided using WFQ-based packet scheduling. However, these two solutions have been mutually exclusive until now. All of the work on WFQ-based scheduling algorithms have required that switches/routers use output queuing, or centralized shared memory. Thus we have to deal with the trade-off in our design for the switch architecture. To implement pure output queuing, we need speedup of N for the switch fabric that is not practical for our basic OC-48 channel. Thus we choose input queuing as our basic design. And we will use iSlip algorithm, which uses virtual output queuing (VOQ), to eliminate the head of line (HOL) blocking. Recent research shows that a combined input output queued (CIOQ) switch running twice as fast as an input-queued switch can provide precise emulation of a broad class of packet scheduling algorithms, including WFQ and strict priorities. Although the speedup of 2 is still a too high requirement now, but it might be possible in the near future. So we analyze and simulate the CIOQ scheduling as our alternative design to support QoS in the very high bandwidth routers.

A. iSlip

One of the major types of blocking that can limit a crossbar switch's performance is called head-of-line (HOL) blocking. HOL-blocking can significantly reduce the performance of a crossbar switch wasting nearly half of the switch's bandwidth. In the very simplest crossbar switches, all of the cells waiting at each input are stored in a single FIFO queue. When a cell reaches the head of its FIFO queue, the centralized scheduler considers it. The cell contends for its output with cells destined to the same output, but currently at the HOL of other inputs. It is the job of the centralized scheduler to decide which cell will go next. This FIFO queuing has a problem: cells ahead of them destined to a different output can hold up cells. This phenomenon is called HOL-blocking. The centralized scheduler only "sees" the cell at the head of the FIFO queue, and so the HOL cell blocks packets behind it that need to be delivered to different outputs. Even under benign traffic patterns, HOL blocking limits the throughput to just 60% of the aggregate bandwidth for fixed or variable length packets. When the traffic is bursty, or favors some output ports the throughput can be much worse.

There is a simple fix to this problem known as virtual output queuing (VOQ). At each input, a separate FIFO queue is maintained for each output, as shown in Figure 6. After a forwarding decision has been made, an arriving cell is placed in the queue corresponding to its outgoing port. At the beginning of each time slot, a centralized scheduling algorithm examines the contents of all the input queues, and finds a conflict-free match between inputs and outputs. In theory, a scheduling algorithm can increase the throughput of a crossbar switch from 60% with FIFO queuing to a full 100% if VOQs are used.

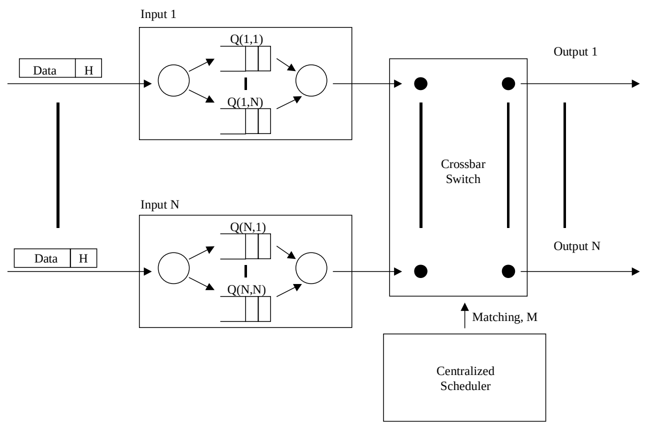


Figure 6: Shows how VOQ works to prevent HOL blocking.

iSLIP algorithm is one of the scheduling algorithms for crossbar switches that use virtual output queuing. It is designed to meet such properties as high throughput, starvation free, fast and simple to implement. It is an iterative algorithm, during each time slot, multiple iterations are performed to select a crossbar configuration, matching inputs to outputs. The iSLIP algorithm uses rotating priority (“round-robin”) arbitration to schedule each active input and output in turn. ISLIP 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. The steps of each iteration are

1. **Step 1**: Request. Each input sends a request to every output for which it has a queued cell.
2. **Step 2**: Grant. If an output receives any requests, it chooses the one that appears next in a fixed, round robin schedule starting from the highest priority element. The output notifies each input whether or not its request was granted.
3. **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 element. The pointer to the highest priority element of the round-robin schedule is incremented (modulo N) to one location beyond the accepted output. The pointer to the highest priority element at the corresponding output is incremented (modulo N) to one location beyond the granted input. The pointers are only updated after the first iteration.

By considering only unmatched inputs and outputs, each iteration matches inputs and outputs that were not matched during earlier iterations.

The performance of iSLIP is pretty good. In brief, it has following main properties:

1) Property 1

High Throughput. For uniform, and un-correlated arrivals, the algorithm enables 100% of the switch capacity to be used. In our simulation, we have proved this.

2) Property 2

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 iSLIP is in some sense fair: with one iteration and under heavy load, all queues with a common output have identical throughput. But there are two kinds of other blocking will make a packet’s delay unpredictable, input- and output blocking. Input blocking arises because VOQs at one input can blocked by other VOQs at the same input that receive preferential service. 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 only transfer 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 very difficult to predict when a cell will be delivered to its output. We can use some techniques such as priority classes and speedup to control the delay of packets. But this can only solve the problem to a limited extent and we still can’t predict the time a cell will spent within the switch fabric. Thus this makes it difficult to implement output fair queuing to support QoS. We will see a CIOQ scheduling with speedup of 2 will exactly emulate any of the output fair queuing and provide the same QoS.

3) Property 3

Fast. The algorithm is guaranteed to complete in at most N iterations. However, in practice the algorithm almost always completes in fewer than iterations. But this is still not good enough for scalability because of its centralized iteration process.

4) Property 4

Simple to implement. An iSLIP scheduler consists of 2N programmable priority encoders. It can be implemented on a single chip.

B. CIOQ

A switch with a speedup of S can remove up to S cells from each input and deliver up to S packets to each output within a time slot, where a time slot is the time between packet arrivals at input ports. Hence, an OQ switch has a speedup of N while an IQ switch has speedup of one. For values of S between 1 and N packets need to be buffered at the inputs before switching as well as at the outputs after switching. We call this architecture a combined input and output queued (CIOQ) switch.

In order to offer QoS guarantee, the CIOQ switch should behave identically to an OQ switch for all types of traffic. Here a CIOQ switch is said to behave identically to an OQ switch if, under identical inputs, the departure time of every cell from both switches is identical. Recent research [1] has shown that a speedup of 2 is sufficient and speedup of 2-1/N is necessary for this exact emulation. As a benchmark with which to compare our CIOQ switch, we will assume there exists a shadow N*N OQ switch that is fed the same input traffic pattern as our CIOQ switch. The key to solving the speedup problem is a scheduling algorithm that keeps track of the cells in the CIOQ switch. The scheduling algorithms decides the order in which cells at the input are transferred across the switch fabric to the output in such a way that the cells may depart from the switch at the same time as they do in the shadow OQ switch. Each time cells are to be transferred, the scheduling algorithm selects a matching between inputs and outputs so that each non empty input is matched with at most one output and, conversely, each output is matched with at most one input.

Before we explain the CIOQ scheduling algorithm further, we give some definitions that are crucial to the following part.

1) *Push-in-queues*

Similar to a discrete-event queue, a push-in queue is one in which arriving cells are added to an arbitrary location in the queue based on some criterion. The only property that defines a push-in queue is that once placed in the queue, cells may not switch places with other cells. In other words, their relative ordering remains unchanged. In general, we distinguish two types of push-in queues: (1) “Push-In-First-Out” (PIFO) queues, in which arriving cells are placed at an arbitrary location, and the cell at the head of the queue is always the next to depart. (2) “Push-In-Arbitrary-Out” (PIAO) queues, in which cells are removed from the queue in an arbitrary order, i.e. it is not necessarily the case that the next cell to depart is the one currently at the head of the queue. We will use PIAO queues as a buffering mechanism at the input of a CIOQ switch. Each output maintains an output priority list: an ordered list of cells at the input waiting to be transferred to this particular output. The output priority list is always arranged in the order in which the cells would depart from the OQ switch we wish to emulate (i.e. the shadow OQ switch). This priority list will depend on the queuing policy followed by the OQ switch (FIFO, WFQ, strict priorities etc).

2) *Time to Leave*

TL(c) is the time slot in which cell c would leave the shadow OQ switch. Of course, TL(c) is also the time slot in which it must leave from our CIOQ switch for the identical behavior to be achieved.

3) *Output Cushion*

OC(c) is the number of cells waiting in the output buffer at cell c’s output port which have a lower time to leave value than cell c. If a cell has a small (or zero) output cushion, then the scheduling algorithm must urgently deliver the cell to its output so that it may depart when its time to leave is reached. A cell’s output cushion can only be increased by newly arriving cells that are destined to the same output and have a more urgent time to leave.

4) *Input Thread*

IT(c) is the number of cells ahead of cell c in its input priority list. In other words, it represents the number of cells currently at the input that need to be transferred to their outputs more urgently than cell c. A cell’s input thread is decremented only when a cell ahead of it is transferred from the input, and is possibly incremented by newly arriving cells. It would be undesirable for a cell to simultaneously have a large input thread and a small output cushion, the cells ahead of it at the input may prevent it from reaching its output

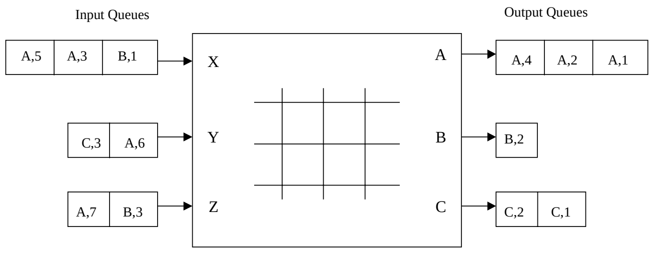


Figure 7: CIOQ at work

before its time to leave. This motivates the definition of slackness.

5) *Slackness*

L(c) equals the output cushion of cell c minus its input thread i.e. Slackness is a measure of how large a cell’s output cushion is with respect to its input thread. If a cell’s slackness is small, then it urgently needs to be transferred to its output.

6) *Stable Matching*

A matching of input ports to output ports is said to be stable if for each cell c waiting in an input queue, one of the following holds:

1. Cell c is part of the matching, i.e. c will be transferred from the input side to the output side during this phase.
2. A cell that is ahead of c in its input priority list is part of the matching.
3. A cell that is ahead of c in its output priority list is part of the matching.

To illustrate our definitions, Figure 7 shows a snapshot of the CIOQ switch with a number of cells waiting at its inputs and outputs. We define the time of the snapshot to be time slot 1. We use the notation (P,t) to represent a cell that, in the shadow switch, will depart from output port P at time t, its time to leave. Consider, for example, the cell denoted in the figure by (A,3). For the CIOQ switch to mimic the shadow OQ switch, the cell must depart from port A at time 3. Its input thread is IT(c) = 1, since (B,1) is the only cell ahead of it in the input priority list. Its output cushion is OC(c) = 2, since out of the three cells queued at A's output buffer, only two cells (A,1) and (A,2) will depart before it. Further, the slackness of cell c is given by L(c) = OC(c) - IT(c) = 1.

The conditions for a stable matching can be achieved using the so-called stable marriage problem. Solutions to the stable marriage problem are called stable matching and were first studied by Gale and Shapely [2]. They gave an algorithm that finds a stable matching in at most M iterations, where M is the sum of the lengths of all the input priority lists.

Our specification of the scheduling algorithm for a CIOQ switch is almost complete: the only thing that remains is to specify how the input queues are maintained. Different ways of maintaining the input queues result in different scheduling algorithms. On the output side, the CIOQ switch keeps track of the time to leave of each waiting cell. During each time slot the cell that departs from an output and is placed onto the outgoing line is he one with the smallest time to leave. For our CIOQ switch to successfully mimic the shadow OQ switch, we must ensure that each cell crosses over to the output side before it is time for the cell to leave. This is the key point to design the queuing policy. The following is our CIOQ scheduling algorithm.

Group By Virtual Output Queue (GBVOQ) algorithm

GBVOQ maintains a VOQ for each input-output port pair. When a new cell arrives at an input port, GBVOQ checks to see if the corresponding VOQ is empty. If it is, then the incoming cell is also placed at the head of the entire input queue, which means the corresponding VOQ will get the highest priority. If, on the other hand, the VOQ corresponding to the new arrival is non-empty, the new cell is placed into VOQ according to the specific fair queuing policy of the output queue. And the corresponding VOQ's relative priority is not changed. In our simulation, we just implement a simple FIFO scheme, thus we just place the cell at the tail of the nonempty VOQ.

Delay Till Critical (DTC) strategy:

In order to reduce the number of iterations needed to compute a stable matching from N2, we introduce the Delay Till Critical (DTC) strategy. During each scheduling phase, we mark as active all cells with a slackness of zero or less, and mark all other cells inactive. The stable matching algorithm now considers only active cells. This simple strategy reduces the number of iterations to compute a stable matching from N2 to N. It is easy to see that all cells that are in the same VOQ occupy contiguous positions in the input queue. Therefore it is sufficient to just keep track of the relative priority ordering of VOQs.

Stable Marriage Matching Algorithm:

At each scheduling phase, the centralized scheduler finds a stable matching between the input ports and the output ports. This is can be realized by the so-called stable marriage matching algorithm.

First, each input port propose to its "favorite" output port. Each output port who receives more than one proposal rejects all but its "favorite" from among those which have proposed to it. However, the output port does not accept it yet, but keeps it in a waiting list to allow for the possibility that some other input port with higher priority may request later. For the output port, the relative priority of proposing input ports is determined by the specific output queuing policy. In our simulation, we just implement a simple FIFO scheme.

Second, those input ports that were rejected now propose to the next choices in their output priority list. Each output receiving proposals chooses its favorite from the group consisting of the new proposals and the one in the waiting list, if any. It rejects all the rest and again keeps the favorite in suspense.

This iteration process proceeds in the same manner until all the input ports have finished proposed for all its active cells. We want to point out that the CIOQ scheduling algorithm can be used to emulate any output queuing scheme. We need only to use the specific OQ policy to calculate the relative priority of the cells to insert them into the right position of the VOQs and help the output ports to determine which proposing input port to accept in the stable marriage matching algorithm.

The weakness of the CIOQ algorithm is from its requirement of speedup of 2 and its bad scalability. But we still believe that it will help to support QoS in very high bandwidth switches and routers in the near future.

VII. Physical Layout

The physical design is a crucial element in coming up with a useable product. Even the most brilliant ideas can be rendered useless by physical constraints or poor consideration of practical factors. We have devised a powerful switch but we need to ensure that it can be integrated into a modern data center environment.

A. Chassis Design

With the idea of flexibility in mind, we chose to separate the router into two basic modules: the Line card Module and the Switch Fabric/Control Module. This separation enables the user to be very flexible with rack placement and also accommodates scaling and reconfiguration without requiring much manual labor. It also reduces cost by using a small generic case for each module. What enables us to separate these normally eminent functions is the use of the LCS (Line Card Switch) protocol for backplane data. The LCS protocol connects individual 10Gbps line cards to the switching matrix over a multi-mode fiber optic connection. These links can be up to 200 ft. long so the user can put line cards in a separate rack or possibly a separate room from the switch. The new configurations that are possible with this design are many. It will give architects and facilities managers quite a lot of room to work with.

Another aspect of the chassis design that some designers overlook is the simple fact that this product must fit into an industry standard 19 inch or 24 inch rack. Any oversize protrusion in any dimension can cause a lot of frustration and unnecessary difficulty for integration. Our generic case is dimensioned 10 inches high by 17.25 inches wide (without rack mounting brackets) by 18 inches deep. This case is perfectly suited to our circuit boards and should not cause any headaches when it comes time to bolt it into the racks. No user wants to have to redesign their data center to accommodate a new product, no matter how much it improves performance. We have made sure that our switch fits those basic specifications.

The weight is also quite reasonable, topping out at around 50 pounds for a fully loaded line card module and about 35 pounds for a switch/control module with redundant fabrics. Users should not have any trouble achieving high density without buckling the raised floors.

This layout is depicted in Figure 10.

B. *Power Specifications*

An additional advantage of having a modular system is that we do not have a huge box that requires special high current power lines. Each module is equipped with a 2+1-power module configuration. While it would be ideal to have a one to one ratio for protection, it makes requirements on space that add very little statistical superiority. In addition, the power can easily be connected to separate redundant power feeds. The line card module runs of 220 VAC and peaks out at 2000 W of power. It can also be equipped with -48 VDC power modules topping out at 1700 W. The switch/control module requires slightly less power and draws 1500 W at 220 VAC and 1300 W under DC power. This is perfectly reasonable for most settings.

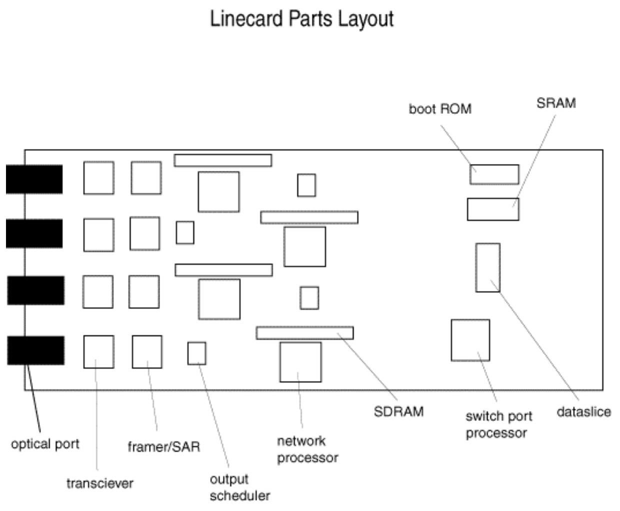


Figure 8: Physical card Layout

C. *Standards Compliance*

As a wise man once said "the wonderful thing about standards, is that there are so many to choose from" and that` certainly applies to network equipment. There are numerous well thought out international standards and we have tried to make sure that our design will comply with the most important ones.

Safety is always a fundamental concern and our device should comply with UL 1950, IEC 60950 and 60825, TS 001 as well as AS/NZS 3260. This should be adequate to ensure that this is not a dangerous device to install in a facility.

Electromagnetic emissions, while a concern for network engineers, are more significant to regulators who authorize the sale of electronic devices in different countries. Our device should pass FCC Class A, ICES-003 Class A, EN55022 Class B, VCCI Class B and AS/NZS 3548 Class B.

The last major standard is a broader standard that is crucial in order to sell the product to major US telecommunications carriers. The NEBS standards devised by Telcordia (formerly Bellcore) provides a diverse set of requirements that are suited to the more rigorous standards of voice service providers. We have tried to ensure that our product will be NEBS SR-3580 Level 3 compliant. This should be adequate for most big Bell class carriers.

VIII. Parts list

Due to a lack of time and expertise, we were unable to do a thorough and accurate board level design. However, we were able to make an estimate and approximate the physical layout of the cards.

The line card, while using mostly simpler off the shelf parts, has a higher part count but most of the parts take up a small area. The only ASIC (the output queue manager) should be a relatively simple state machine.

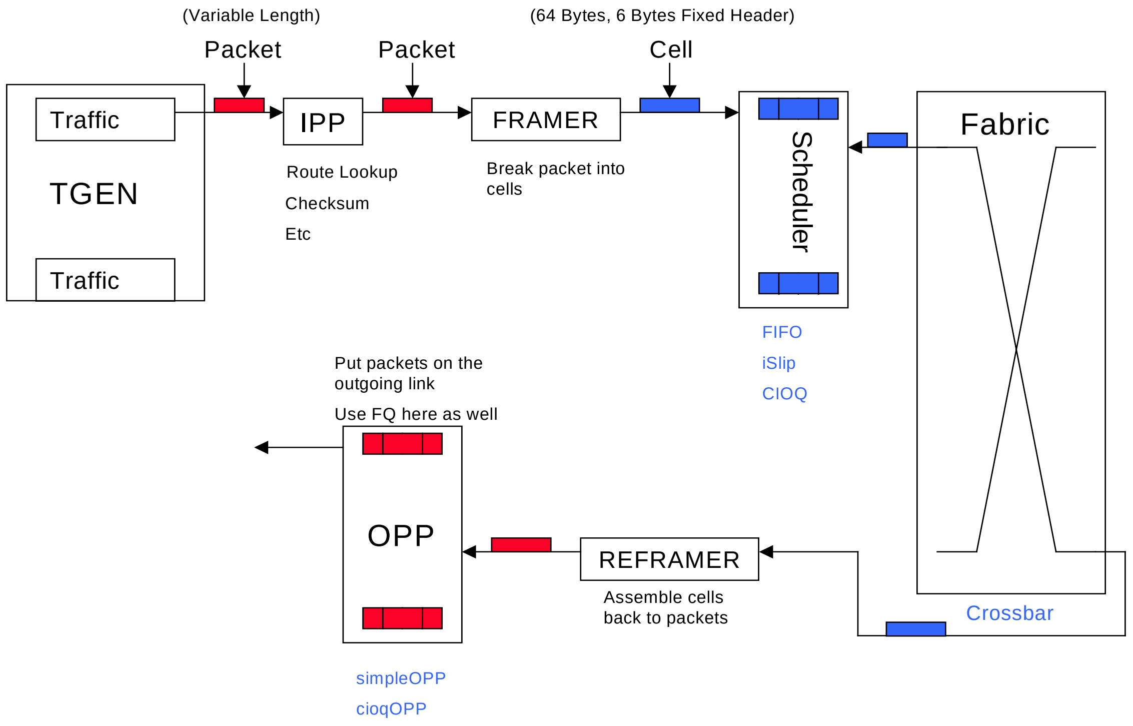


Figure 9: The overall structure of ngrSim. The various modules are represented as separate blocks. There are some modules that have multiple implementation, such as the scheduler.

A. Line card

1. 4x Intel IXP1200 Network Processors
2. 4x AMCC S3046 Transceiver Interface
3. 4x Agilent HFCT-5402D OC-48 Multimode Transceiver
4. 1x 8MB SRAM 1x 8MB boot ROM 4x SDRAM slot
5. 4x Vitesse VSC9112 Framer
6. 1x PMC Sierra PM9313 Data Slice
7. 1x PMC Sierra PM9315 Port Processor
8. 4x Output queue manager ASIC

The switch fabric cards are each comprised of TT1

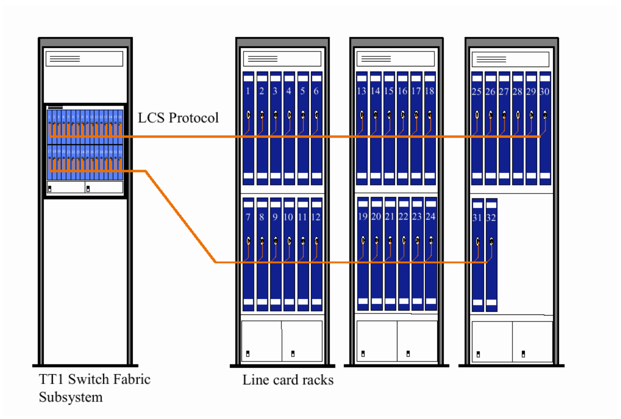


Figure 10: Possible design of the entire ISIS box without scalability extensions.

crossbar switch elements and one modified TT1 scheduler IC. Figure 8 shows the physical layout of a line card.

B. Switch Fabric Card

1x PMC Sierra PM9311 Scheduler

14x PMC Sierra PM9312 Crossbar

The control module is essentially a generic high end Intel laptop motherboard with an interface into the switching matrix as well as 2 flash card slots.

1. 1x Intel 440MX laptop motherboard
2. 2x Flash memory slots

C. Parts Selection Rationale

We tried, as much as possible to base our design around using off the shelf parts. While some efficiency and some cost gains can be made by rolling our own ASICs, there is usually no need to reinvent the wheel and custom designs can

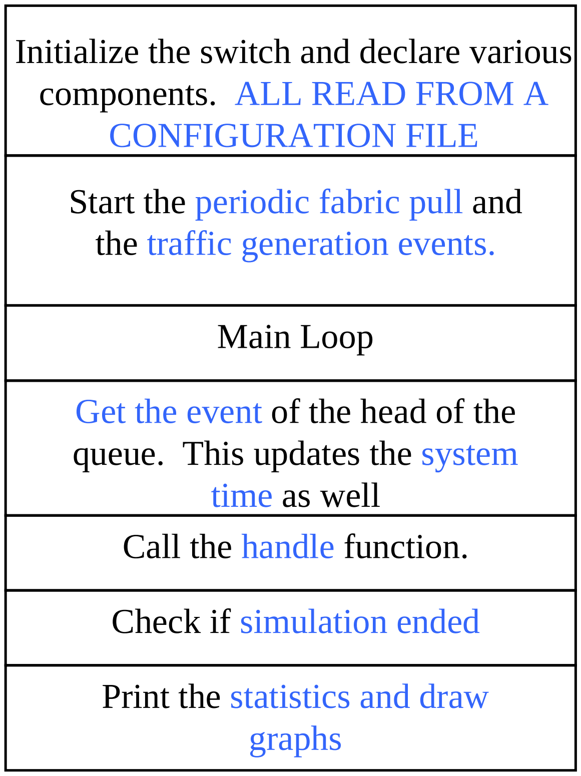
```
class handler {
public:
  handler() { };
  virtual ~handler() { };
  virtual void handle(event *e) = 0;
  void set_next_handler(handler * h)
  {
      next_handler = h;
  }
protected:
  handler * next_handler;
};
```

Figure 11: The description of the class handler. This is an abstract class that acts as a container for all of the objects in the simulation. The handle function is used every time an event needs to be run.

often cause large delays in development. As you can see, we managed to fit this hardware into rather modestly sized spaces. We feel that the trade-off and advantages will yield a feasible product.

IX. Simulator Design

Once we had designed the entire system and looked at the physical constraints that needed to be tackled, we developed a simulator to evaluate the performance of the various scheduling algorithms. The design of the simulator was event based with all actions happening with events. As the switch runs at a nanosecond simulated clock interval, we spent a lot of effort designing the event handling routines and data structure to ensure that the events are handled fast and the extra overhead of this process does not effect the entire simulation.



Initialize the switch and declare various components. ALL READ FROM A CONFIGURATION FILE

Start the periodic fabric pull and the traffic generation events.

Main Loop

Get the event of the head of the queue. This updates the system time as well

Call the handle function.

Check if simulation ended

Print the statistics and draw graphs

Figure 12: Explanation of the main look of ngrSim.

The simulator, called ngrSim (Next Generation Router Simulator) is a modular design that allows for easy mix and match of various components. This means that we could try different algorithms in different stages of the networks easily. This also means that future development on the simulator is very easy. We will explain the simulator in more detail now. Figure 9 represents the overall design of ngrSim.

A. *Events*

The simulator is event driven and thus events are an important part of the simulator. For generality purposes, we used an abstract class handler, as a container for most of the classes in the simulator. The handler class has a function called handle(). The structure of the handler class is shown in Figure 11. Using this class, we defined an event, to be a class with a handler object, some data (that could be either packets or cells) and a few other fields.

When the event at the head of the queue is removed by the main dispatcher, the handle() function of the handler object in the event is run. This function is set at the time the event was inserted into the event queue and it does the appropriate action that needs to be taken. Figure 14 depicts the event structure showing its important fields. During the handling routines for events, new events are inserted into the queue to be run at various times. When there arent any events in the event queue, the simulation has ended.

B. *Event Queue*

```
// Initialize the components tgen, ipp,
// framer, sched, fab, reframer, opp

((crossbar*)fab)->start();
((tgen*)tg)->start();

while (1){
    event* e = (eq.instance()).dequeue();

    if (e == NULL && !sim_running)
      break;

    (e->get_next_handler())->handle(e);

    if ((eq.instance()).get_time() > simTime){
      sim_running = 0;
      ((tgen*)tg)->stop();
      ((crossbar*)fab)->stop();
    }
}

// print Stats
```

Figure 13: Code for the main look of ngrSim

The most important part of the simulator is an event queue. The event queue is a priority list that is prioritized on the time at which the event is supposed to run. The events are inserted into the event queue in an arbitrary order but are removed from the front of the queue. As this is a priority queue, insertion of a new event is an O(N) event where N is the number of events in the queue. With a large switch simulation, there could be a few million events in the event queue at a time that reduces the performance of the simulator tremendously. The naïve way of making the event queue would be a linked list. However, due to the performance penalty that might be incurred, we used a structure known as Skip lists [18]. This structure is a probabilistic structure that allows O(1) inserts on average. With this list structure, the performance of the simulator was enhanced and the running time of simulation reduced. The event queue also keeps time of the current simulated time, which is the time of the last event. The event queue just takes an event from the head of the queue and runs it, updates the time and gets the next event after updating statistics. Figure 13 shows this process as it is written for ngrSim. Figure 12 shows the description of the same steps explaining what each step involves.

C. Components

We followed the same modules that we used in the design of the switch in the simulator as well. The simulator code is divided up into different modules each of that has an interface with some other modules. We made the interfaces generic so that other variants of the modules can be written without ever changing the entire structure or unrelated modules.

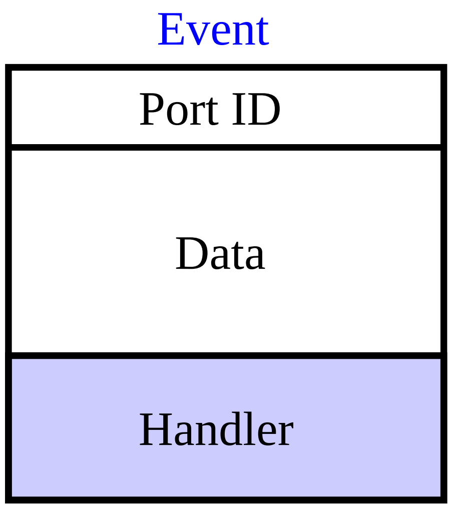


Figure 14: Event structure. The handler could be any object from the simulation.

1) Traffic Generation (TGEN)

The first module is a traffic generation module. This module consists of N traffic modules, each one responsible for generating traffic for the N input ports. Currently, the traffic generators generate traffic from an exponential distribution. An average load can be specified, which is a percentage of the line speed. A mean burst length specifies that burst size that is allowed. A hot spot share is the share of the entire traffic coming in from that input port destined to one specific port i.e. port 0. This tries to emulate the non-uniform destination addresses problem that does not come up when destinations are picked randomly from the available set. The traffic generator is started in the beginning of the simulation. This inserts the first generation events into the event queues and starts the simulation process. When these events are fired, packets are generated, packaged into an event and the handler object is set to the next module in line that is the input port processor (IPP). This event, when run, will call the handle function defined in IPP. Thus a seamless interface between the traffic generation and the input port processor is established through the handler abstract class.

2) Input Port Processor

The traffic generation module passes the packets to the IPP through the event queue. The input port processor, is a network processor that does the normal IP packet processing functions. These functions include:

1. checksum check
2. TTL update
3. Checksum recalculation
4. Packet drop if TTL < 1
5. IP route lookup

These functions have to be performed for each packet. The speed at which these lookups have to be performed has to match the speed of the incoming link. It is generally assumed that the average size of a packet on the Internet is about 1000 bytes. With that calculation and given our line speed of 2.48Gb/s, we will have on average about 0.31Million packet lookups a second. The smallest size packets on the Internet are about 20 bytes long and so in the worst case, there might be 7.75Million packet lookups a second. If we use a single processor per line card, that will mean about 1.24Million packet lookups a second average and 31 Million packet lookups in worst case. This number is too high a rate for any processor to work at and so we will have one processor per port. In the simulator, we are already given the destination address and so no lookup is required. We simulate the delay for the IPP without looking into it in any more detail.

3) Framer

IP packets are variable length packets. Switching fabrics work best when they are handling fixed length packets, as they don't have to calculate the time it would take for a packet to go through the switch. Fragmenting the packets as they go through the fabric and then re-framing them back into packets at the output side solves this problem. This is done with a Segmentation and Reassembly chip (SAR) which can run at OC-48 speeds. The re-framer waits for the cells of a packet to arrive for certain duration of time before assuming that some cell has been dropped and so the entire packet needs to be dropped. It is important to note, that even if a cell with 1 bytes of data is lost, the entire packet has to be discarded which has effect on the overall throughput of the switch. We will address this issue more in the evaluation section.

Choosing a cell size is another interesting problem. By observation, we have seen that the distribution of packets on the Internet is bi-modal with the two values being 40 and 1500 bytes. The tradeoff in determining the cell size is the overhead incurred while making the cell and the speed at which the fabric has to switch. If the cell is too small, the fabric will have to switch at a much higher rate, making the design complex. On the other hand, if the cell size is too large, the overhead of small packets will be significant. Given these constraints, we chose a cell size of 64 bytes with 6 bytes of header. The header bytes include:

1. **2 input + output port number = 4 bytes**
2. **1 byte for cell ID**
3. **1 byte for flag + priority**

The 2 bytes for ports mean that we can have as many as 65K ports on the switch. As we will see a little later, this was an important decision as we ended up with 1024 ports in one of our designs. The overhead for some packet sizes can be high. For example, for a packet size of 59 bytes, two cells are required to transmit that packet. The overhead in this case is 59/128 that is a staggering 46%. We however, believe that very few packets of that size will exist and thus in most cases, our overhead will be significantly less than 46%.

4) *Scheduler*

The framer breaks up the packet into cell and these cells are routed through the switch. A scheduler module takes care of the order in which cells are routed through the switch. There are many different forms of scheduling algorithms that can be applied. The scheduling algorithms for various kinds of queuing disciplines mentioned before are also different. For, and output queuing switch, it does not matter which scheduling algorithm is used because the algorithm is not going to play a part in the performance at all. For input queued switches, iSlip [19] and First-come-first-serve (FCFS) are the commonly used algorithms. We tested out both of these in our implementations.

The scheduler is slightly different from other modules in that it does not add any new events to the event queue. It has a set of buffers, whose organization depends on the scheduling algorithm used, that store the cells before they are routed through the switch. As the cells are all one size, it is easy for the fabric to keep checking the scheduler at periodic intervals about the cells that the scheduling algorithm wishes to send through the fabric. This way, the fabric can pull the cells from the scheduler rather than the scheduler pushing them.

5) *Fabric*

We have currently only implemented a crossbar fabric. A fabric, in general just polls the scheduler for cells at a fixed interval. This interval depends on the speedup of the fabric. Lets say, if the fabric is running at the line speed (2.48Gb/s) and the cell size is 64 bytes, then the fabric needs to pull cells periodically every 25 nsec. At a higher speedup, of lets say, 2, the fabric needs to do a pull every 12.5 nsec. By adjusting the fabric speed, we can easily simulate different speedups.

We have not implemented other fabrics but need to point out that in ngrSim framework, it will be easy to add new fabric modules as well.

6) *Output port processing*

After passing through the fabric and re-framing the cells back into packets, the packets are sent to the OPP. This is the place where the packets are placed on to the outgoing link at the given link speed. Given that all the cells that have to depart the port are there, it is easy to do any possible arrangement of the cells to meet certain QoS guarantees. For example, if a high priority cell is behind in the OPP queue, it can just be sent out early. Many different Fair Queuing mechanisms can be deployed here to achieve this purpose.

D. *Miscellaneous Components*

Apart from these basic components, there are many other parts that have to be developed. We have developed a elaborate configuration mechanism that proved useful when different configurations needed to be tested out many times. We also employed a sophisticated statistic module in each component to gather per component statistic that helps us in evaluating the performance of each component separately. We have also written various scripts to aid in the collection and representation of data in a meaningful format. All of these are packaged with the source code of ngrSim. In the next section we will talk about some of the methodology that we used to collect results and present the actual results of our simulations.

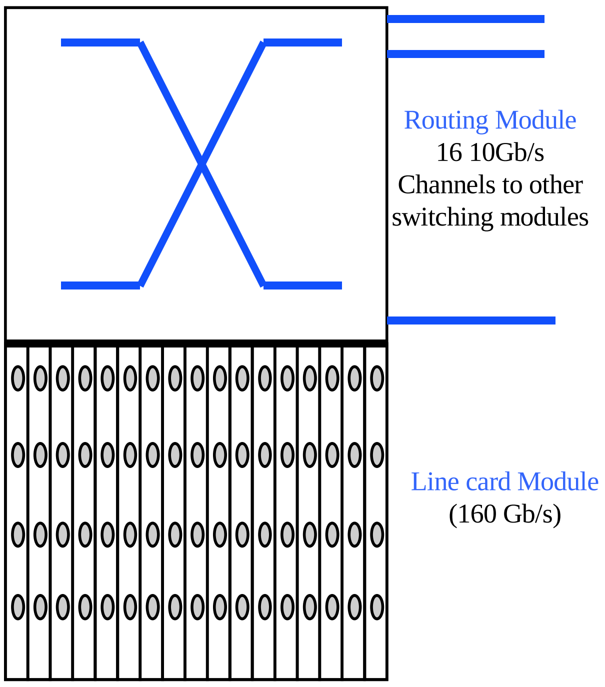


Figure 15: Details of each switching module in ISIS-A.

X. SCALABILITY: ISIS-A

We designed ISIS with the base design with 128 OC-48 ports. However, we had this important goal of making our design scalable to large number of ports and high switching capacity. We went through many iterations of this process to see if it was possible to scale this switch to a higher capacity. We looked at the chipset that we were using and tried to utilize features provided in it to achieve our goal of scalability.

A. *PMC chipset*

The chipset we are using is PMC-Sierra’s TT1 chipset that provides 320Gb/s of bandwidth. The chipset uses a proprietary protocol called line card-to-switch control (LCS) protocol that is used to produce physical separation between the line card modules and the switching fabric. The LCS protocol is run over optic fiber connections running at 10Gb/s that connect line cards with 10Gb/s channels to the switching fabric. These line cards can therefore be separated from the switching fabric by up to 200 meters. This makes the design of the box modular and allows incrementally increasing the line card modules when the need for them arises. Thus we can have 32 line cards sitting in 4 different line card modules connected to the switching fabric through optic fiber running LCS protocol. This is great but it does not produce the kind of scalability that we are interested in for ISIS.

B. *ISIS-A: ISIS with an Attitude*

We decided to take the design to the next level by using the 10Gb/s channels that the TT1 chipset provides and using

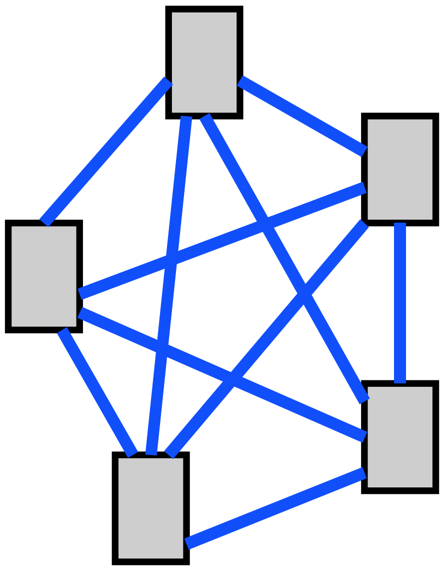


Figure 16: ISIS-A architecture showing the mesh between various switching modules.

them to connect them to different switching fabrics rather than a line card module. This way, we loose capacity on each box but we gain as we have two switching fabrics working together. We can connect up to 16 other switching fabrics to one switch fabric and utilize 160Gb/s of the switching capacity for this purpose. The rest of the capacity can be used for the two line card modules that we connect to the box. Figure 16 shows this in a little more detail.

The switching modules are connected to each other in a mesh. There are many possibilities as to how to connect these switching fabrics together, including a hyper-cube structure, but the simplest arrangement is a mesh. There can be up to 17 switching modules connected together in this arrangement. It is clear that each switching module will have a switching capacity of 160Gb/s left after reserving 16 channels for the mesh connections, which leads to a aggregate bandwidth of over 2.5Tb/s. Each module is shown in a little more detailed form in Figure 15 also. There are many issues with traffic flow that need to be resolved in this design but it still lends itself to these high capacity values and thus is the design for scalability that we have selected.

C. Routing within ISIS-A

Now that we have designed ISIS-A, we need to address the issue of routing traffic between these modules. The problem is that the channels are only capable of carrying 10Gb/s of traffic, while one entire switch can send 160Gb/s traffic destined all for another switch in the worst case. In this scenario, the queues with the 10Gb/s channel will fill up and the performance of the entire switch will fall dramatically. That is also especially bad for our QoS goal in the beginning that will be effected totally by this problem. For the problem of getting more traffic through from one switch to the other, a solution that does not cater to QoS guarantees can do a modified form of **hot potato routing**. In this routing, modules just keep pass the data between them until one module finds bandwidth to handle all the data. Lets take the example of a switching module A that has 20Gb/s of data to send to another module B. Given that the channel between A and B is only 10Gb/s, the rest of the 10Gb/s traffic can not be sent over the channel between the two switching modules. In this scenario, the A sends the excess traffic over to another module that is least loaded, lets say module C. Module C tries to send the traffic over to module B. If it can, we have sent the 20Gb/s traffic to module B. If it does not succeed, it carries the same algorithm to send traffic to another least loaded module. This continues until the traffic gets to its desired destination. It is also important to note that there are some issues with looping in this structure. This can be avoided simply by adding TTL fields to cells so that they do not loop around in the structure.

To keep track of the utilization of a channel, a queue can be maintained and based on its occupancy, a decision to route the extra traffic can be made. These queues are already provided as part of the TT1 chipset.

Thus routing on the ISIS-A occurs in three stages.

1. ports on the card itself (through the line card)
2. Other line cards on same module (through the fabric)
3. Other switching modules (through the interconnect)

This makes the switch load distribute between three different flows and so perform better under heavy load.

D. QoS with ISIS-A

It is obvious that the routing scheme mentioned above totally violates any QoS guarantees. We can bring QoS into the picture by a few mechanisms.

We can reserve the queue with each channel for high priority traffic. We can have more than just 2 classes as well, although we believe that 2 might be enough. If a high priority cell arrives, it is sent using the reserved part of the channel. That might mean dropping low priority traffic using that channel but that is reasonable.

Another idea is a simple limit on the cell TTL. This value can bound the maximum delay that a cell has to experience. A cell with a low TTL is treated a high priority and is sent towards the destination as soon as possible. We believe that using these two schemes, efficient QoS can be provided to flows without introducing complexity into the system. The guarantees provided will not be hard but will still satisfy most service constraints.

XI. Evaluation

We ran the simulator using scripts with random seeds. The figures that are presented in this section are each plotted after running 5 experiments at each load value and then taking the average of the 5. We also made sure that we collect values for the statistics only after running the simulation for a considerable amount of time so as to let the simulation reach a steady state. Due to time constraints, we could only run simulations for 4 and 6 ports. We could only managed a few simulations with 64 ports and observed that the results were very similar for 16 ports. Therefore, we believe that running for a lesser number of ports.has not effected our simulation results greatly.

Due to the shear number of the graphs that we had to use, we have included the graphs as an appendix. We present the explanation for the results below. The appendix is labeled A-I, and each page has 3 graphs. So C-3 refers to the third graph on page C.

A. Explanation of Results

A-1 With non-uniform traffic, all three schedulers (nFIFO, iSLIP, and CIOQ) perform better with low drop rate even at high loads. FIFO experiences Head-Of-Line blocking and thus had higher drop rate at high loads.

A-2 The delay for FIFO increases sharply at about 0.4 load and then it starts dropping packets and so the delay flattens out. While other schedulers don't drop packets that frequently, FIFO causes higher delays for the packets. CIOQ performs better at high loads as compared to the other schedulers.

A-3 CIOQ performance goes down when the traffic is non-uniform. This is seen by the drop rate of CIOQ with non-uniform traffic.

B-1 B-2 FIFO under non-uniform traffic performs worse than under uniform traffic. This can be seen by the delay, throughput graphs for FIFO with non-uniform load. These graphs use a fabric speedup of 1 and so the values for drop rates and latencies are very high.

B-3 The drop-rate graph for FIFO with a speedup of 1.25 shows that with non-uniform traffic, the drops increase significantly more than with uniform traffic.

C-1 C-2 For CIOQ, the delay and throughput graphs show that under non-uniform traffic conditions, the performance of this scheduling algorithm deteriorates at about a load of 0.4.

C-3 The drop rate graph for FIFO with a speedup of 1 shows that drops increase when the traffic is non-uniform. Comparing this graph to B-3 shows that the speedup of 1.25 is necessary to bring down the drop rates for FIFO under all conditions.

D-1 D-2 The delay and throughput graphs for FIFO with speedup of 1.25 shows that the performance goes down with non-uniform traffic. Comparing these graphs to B-1 B-2, we can see that the speedup of 1.25 is necessary to get the performance higher.

D-3 E-1 The latency with FIFO increases as we increase the buffer size. Cells have a longer time to stay in the buffer before they are transferred out leading to high latency values. However, the throughput is exactly the same. This is because when the buffer fills up, then the throughput is not dependent on the buffer size but rather the latency will increase.

E-2 E-3 F-1 The delay function for smaller mean burst length for FIFO is a step function, with different load, the delay will stay constant until it reaches the next level of load. This is probably due to the traffic pattern under different load. The delay for larger mbl is a constant growing function, which is consistent with other results. The throughput under small mean burst length is smaller than that under the larger mbl. This is because the smaller packets takes more overhead than larger packets.

F-2 F-3 G-1 We see for the uniform traffic, from load = 0.8 all the ports' buffer are starting to saturate for iSlip, thus the delay is beginning to increase greatly. But for hot spot = 2/n, beginning at load = 0.4, the hot spot port's buffer is beginning to saturate, its delay is getting larger than that for uniform traffic. And because only one hot spot port is saturated, the it's saturated delay is smaller than that of the uniform traffic.

G-2 With a FIFO scheduler and a speedup of 1.25, we observe the a throughput is close to theoretical maximum of 58%. At the scheduler, the throughput is 58%, while the overall throughput is 40% due to drops after the scheduler. (One cell drop result the drop of the whole packet)

G-3 With a FIFO scheduler and a speedup of 1, the throughput is even worse because we don't consider the overhead taken by the cell headers. We get a scheduler throughput of 45% and overall throughput of 30%.

H-1 H-2 These figures illustrate the effect of speedup on FIFO scheduler. With the higher speedup, the drop rate decreases and throughput increases. At the speedup equal to the number of ports, the throughput is 100% with drop rate 0%. Also higher speedup result in lower delay because cells travel across fabrics faster.

H-3 I-1 Among the three scheduler, CIOQ gives the best throughput, follow by iSLIP. FIFO is the worst due to HOL. This is also the same trend in the delay graph.

I-2 Higher buffer size means higher delay, because with a small buffer, when buffer is full, the cell simply drops. Bigger buffer can store more cells, and when they finish transmitting, increase the overall delay.

I-3 Under uniform traffic, CIOQ has even lower delay than nFIFO with only a speedup of 2. ISILP is good until higher load of 0.8. This again illustrates CIOQ is a better choice for scheduler.

XII. Conclusion and Future Work

We believe that it is possible to provide strong QoS guarantees even in switches with very high capacity. This has normally been thought off as an impossible task. However, with the growing demands for QoS by new Internet Services, this is an unavoidable fate. We believe that using the concepts in our design, and the scheduling results that we have shown, it is possible to cater to a wide variety of applications as far as the QoS guarantees are concerned.

We also looked at the physical aspects of the router and ensured that we addressed the pressing issues of Central Offices, including space and size. The modular design of ISIS and ISIS-A using the TT1 chipset lends itself to easy configurations that can fit into any ISP Central office.

Our simulation results have shown that the throughput, delay and drop rates for the scheduling algorithms that we have chosen can meet the challenges of the new Internet. However, given the modular design of the simulator, we would like to test out more options in both scheduling and Fabric design to see if we can improve on the performance of ISIS. Due to the time constraint, we could not simulate ISIS-A and its routing algorithm, but believe that those concepts can be easily tested out in the future.

XIII. References

[1] AMCC product page. http://www.amcc.com/Products/Crosspoint/prod_190.htm
[2] TriQuint Semiconductors. http://www.triquint.com/Digital/Products/NetCom/Switching/TQ8025/sf_TQ8025.htm
[3] PMC Sierra. http://www.pmcsierra.com/tt1/index.html
[4] Cisco GSR-12000 series router documentation page. http://www.cisco.com/univercd/cc/td/doc/pcat/12000.htm
[5] Juniper Networks M-40 product page. http://www.juniper.net/products/m40-l2.htm
[6] Juniper Networks M-160 product page. http://www.juniper.net/products/m1 60-l2.html
[7] Advanced Micro circuits Corporation. http://www.amcc.com.
[8] Vitesse Corporation, http://www.vitesse.com
[9] TriQuint, http://www.triquint.com
[10] Intel IXP-1200, http://www.level1.com
[11] Floyd, S., and Jacobson, V.Random Early Detection gateways for Congestion Avoidance IEEE/ACM Transactions on Networking, 1(4):397-413 August 1993.
[12] J.C.R. Bennett and H. Zhang, ``WF2Q: Worst-case Fair Weighted Fair Queueing", INFOCOM'96, Mar, 1996.
[13] Jon C.R. Bennett and H. Zhang, Hierarchical Packet Fair Queueing Algorithms. IEEE/ACM Transactions on Networking, 5(5):675-689, Oct 1997. Also in Proceedings of SIGCOMM'96, Aug,1996
[14] Donpaul Stephens, Hui Zhang "Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture". INFOCOM'98.
[15] Fabio Chiussi, Andrea Francini, “A Distributed Scheduling Architecture for Scalable Packet Switches”.
[16] S. Chuang, A. Goel, N. Mckeown, B. Prabhakar, “Matching Output Queuing with a combined Input Output Queued Switch”.
[17] J. Chao, L. Chen, “delay-Bound Guarantee in Combined Input-Output Buffered Switches”.
[18] Skip lists, http://ww.cs.umd.edu/~pugh
[19] N. McKeown, “Fast Switched Backplane for a Gigabit Switched Router”.
[20] G. Shapely, L. Shapely, “College Admissions and the stability of marriage”.