

# System Requirements for Super Terabit Routing

Mehrdad Nourani  
Dept. of EE  
The Univ. of Texas at Dallas  
Richardson, TX 75083  
nourani@utdallas.edu

Gautam Kavipurapu and Raju Gadiraju  
R&D Division  
IRIS Technologies, Inc.  
Dallas, TX 75229  
gautam,raju@iris-technologies.net

## Abstract

Rapid growth of Internet and huge demand for higher data transfer rate created a challenge for network, processor and communication engineers. In this paper, we first discuss the deficiencies of the conventional routing switch architectures. Then, we present a novel switch strategy based on space and time division multiplexing and a system level architecture by which achieving several hundred of Terabits per second throughput for networking equipments becomes possible.

**Keywords:** Crossbar, Internet Protocol, Network Processor, Router, Routing Lookup Table, Shared Memory, Switch Fabric.

## 1. INTRODUCTION

In recent years the phenomenal growth of the Internet has created an unprecedented demand for faster, more flexible and yet shorter time-to-market systems to process the IP (Internet Protocol) packets. Figure 1, chosen from [1], shows the rapid growth that almost doubles the data traffic every three to six months.

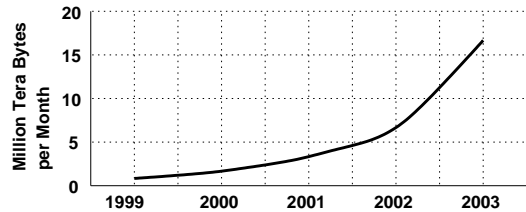


Figure 1: Internet traffic growth forecast.

The trend of such exponential growth is not expected to slow down, mainly because the data-centric businesses and consumer networking applications continue to drive global demand for broadband access solutions. Figure 2, presented in [2], demonstrates some of the main carriers of the IP packets and the need to accommodate their backbone speed and capacity requirements.

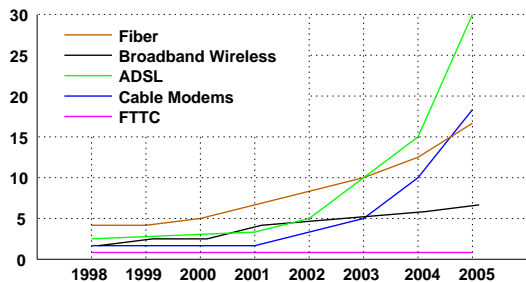


Figure 2: The growth in Internet access lines.

From architecture point of view, the huge increasing demands of bandwidth has rekindled the need for scalable switching-routing architectures that satisfy various cost and performance constraints and more

importantly scale effectively to satisfy the demands of tomorrow. In today's market terminology, these architectures are called *network processors*. There are around 50 companies competing in this field and according to some independent research and consulting companies [1] [3] the sales are expected to grow by more than 200% by the end of 2001 and surpass ASIC sales by 2003. Although there is no universal agreement on the definition of network processors (also called Network Instruction Set Computer or NISC), they are intended to provide at least three important features for their main job of data transfer: 1) programmability, 2) flexibility; and 3) high speed [3].

### • Programmability:

The network processor evolved because the industry gradually realized that three conventional CPU design philosophies, i.e. RISC, CISC and DSP, were not designed to perform the network functions. For example, fast multiplication is crucial in DSP processors, important in CISC and customary in RISC. However, multiplication imposes an unnecessary overhead to NISC. As another example, the lookup table search and score boarding [4] [5] [6] is quite crucial in NISC while it has no general purpose use in RISC, CISC or DSP counterparts. The programmable nature of NISC gives the products longer life time due to its potential for software upgrading by user.

### • Flexibility:

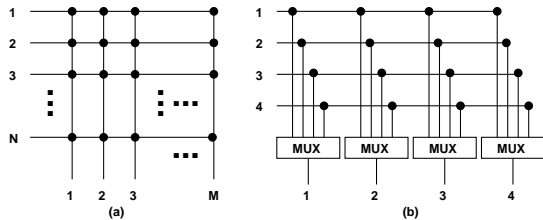
The nature of tasks required in the IP packet processing is very much different compared to scientific or database processings. Among the tasks a NISC will be called upon to perform are data extraction, filtering, classification, modification, lookup search and queue/policy management. Other new challenges include encryption/decryption of IP packets, handling Quality of Service (QoS) priorities among packets and on-demand switching patterns (e.g. multicast).

### • High Speed:

The ultimate effectiveness of NISC will be judged based on its capability in handling the packets at the wire speed. By year 2003, the routers are going to require at least 3 Tbps of bandwidth [1]. This speed requirement comes from the fact that the current network equipments (e.g. cable modem, DSL (Digital Subscriber Line) modems, edge/core routers and optical transport nodes) are not structured for such high volume of data traffic. As a result some level of aggregation is inefficiently passed on to the next stage without a clear solution on sight.

The main motivation of this work is the fact that the speed demands per Point of Presence (POP) will continue to increase rapidly. These demands will be dominated by variable bit rate data traffic such as real time audio and video applications. This becomes even more important when the networks carry mixed Constant Bit Rate (CBR) real-time, Variable Bit Rate (VBR) and Available Bit Rate (ABR) traffic.

To achieve high speed and highly flexible routing systems, both software and hardware aspects need to be addressed. Most of the software-related work in this field has been around faster lookup table search



**Figure 3: Space division (crossbar) switch: (a) a general  $N \times M$  switch, (b) an implementation of  $4 \times 4$  switch.**

[4] [5] [6] and most of the hardware-related work have been around designing more efficient switch fabric [7] [8] [9]. We have also considered both of these aspects in our work. The main contribution of this paper is twofold. First, we show why the two conventional switching approaches, i.e. crossbar and shared memory switches, will not be able to practically satisfy the Terabit data transfer demand. Second, we propose a mixed switch mechanism and a system-level architecture by which getting very high data transfer rate (hundreds of Tbps) is achievable and economical.

## 2. CONVENTIONAL SWITCH STRATEGIES

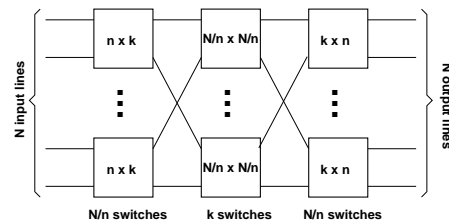
Traditional approaches to switch fabric design are implemented with either crossbar (space-division) or shared memory (time-division) switching cores. They can be implemented as single or multistage fabrics. In this section, we review these two strategies and discuss why they are not suitable for practical implementation of future routing devices.

### 2.1 Space-Division Switch

Figure 3(a) represents a general crossbar switch. It can be visualized as a rectangular matrix with  $N$  inputs and  $M$  outputs. In most cases, switches are made symmetric, i.e.  $M = N$ . A crossbar implementation of a switch fabric can be done in many ways. A simple implementation of a  $4 \times 4$  crossbar switch using multiplexors is shown in Figure 3(b).

The complexity (e.g. for control, scheduling, etc.) of the an  $N \times N$  crossbar switch is  $N^2$ . As one gets to larger and larger crossbar switches  $N^2$  becomes quite a large number and it is impossible to build polynomial computation machines that scale very well that perform  $N^2$  calculations on every clock cycle. If this control is completely done by software it introduces unnecessary latency. Additionally, crossbar implementations are limited by the IC packaging. Thus, one of the design challenge is always a tradeoff between the bandwidth and latency of the crossbar device. Note that the throughput is a function of both bandwidth and latency, that is:  $Throughput = B \cdot f = B/L$ , where  $B$ ,  $f$  and  $L = 1/f$  denote bandwidth, frequency and latency, respectively. For an  $N \times M$  switch running at frequency  $f$ , the maximum throughput (when fully non-blocking) will be  $\min\{N, M\} \cdot f$ . Thus, for a given physical link, between any two points, if we try to increase the bandwidth by creating a large parallel bus, we run into limitations (e.g. radio frequency issues, electromagnetic effects, signal integrity problems, etc.) on how fast we can run the bus or clock it. To counteract the effects, if the bus clock is slowed down, the latency increases. On the other hand, if the bus is made wider, the bandwidth increases allowing more bits to traverse in a single clock cycle.

Thus, due to I/O limitations on the package and clocking speeds, most crossbar switches of today that are in a single IC are typically limited to sizes of  $64 \times 64$  bits. To switch data coming in through a port that is running at a line rate of OC-48 (2.5 Gbps) the single crossbar would have to run at 40 MHz and take the data through all 64 input pins ( $64 \times 40 \times 10^6 = 2.56 \times 10^9$ ). This is a very inefficient utilization of the switch as it only services one data stream (e.g. one OC-48). This would be a “blocking” configuration, and thus not very useful. Multiple crossbar switches are cascaded together to form multi-stage non blocking  $N \times N$  architectures such as, Banyan Network, Butterfly Net-



**Figure 4: A symmetrical three-stage Clos switch.**

work, Omega, and the more commonly used in switches and routers of today, a Clos network.

C. Clos from bell labs [10] proposed a way to implement an  $N \times N$  switch using devices that are of a smaller size than  $N$ , i.e. that are of the size  $n \times k$  where  $n \ll N$  and  $k \ll N$ . The three-stage Clos switch is shown in Figure 4. To develop this switch, Clos discovered that the optimal solution for a symmetrical switch of size  $N \times N$  is obtained when  $k = 2n$  [10]. This value of  $k$  is used to solve for the relationship between  $n$  and  $N$ . More specifically, if  $k = 2n$  to implement a non blocking three-stage, the optimal  $n$  is:  $n = (N/2)^{1/2}$ .

Let’s now consider a practical scenario. To implement a switch fabric that is  $4096 \times 4096$  lines using  $n = 64$  and  $k = 128$ , one would need 256 switches in the input stage of size  $64 \times 128$ , 128 switches of size  $64 \times 64$  in the center stage and 256 switches of size  $128 \times 64$  in the output stage. This means 640 chips and there are around 24000 signal lines running among them. Practically, that is a lot of chips and a lot wires on the printed circuit board (PCB). More importantly, all these need to be operated synchronously. Assuming the switch can run it at 50 MHz (which would be extremely difficult if not impossible), the aggregate throughput of this switch is now  $4096 \times 50MHz = 204.8Gbps$ . This looks very high throughput. However, the controller now has to worry about controlling or switching on and off 4.7 million cross-connects in three stages and has to manage it synchronously in a single clock cycle to implement a wire-speed switch matrix. That requires computing power that has not been invented yet. The number of chips, cross-connects and complexity of the controller will rise even higher if smaller switches (e.g.  $32 \times 64$  instead of  $64 \times 128$  which is more practical) are used. This shows the complexity and the limitations on scalability of the crossbar switches are poor. This has led most of the switch fabric companies to seek all optical cross-connects as a very expensive alternative for high-speed switching.

### 2.2 Time-Division Switch

The shared memory approach buys the ability to create larger bandwidth switches at the expense of latency. The shared memory technique can be used to implement a switch fabric such that it allows one to traverse between each of these  $N$  input and  $M$  output points with as many possible paths that can be formed (a fully connected graph between  $N$  inputs to  $M$  outputs). These were employed in supercomputers. It was not feasible however, to implement large switch fabrics until memory technology’s recent advances. Now, shared memory switches are very common in commercially shipped products.

Figure 5 shows how a shared memory switch can be visualized. Time division multiplexing is performed by the  $N \times 1$  MUX in front and a single data stream is sent to Time Slot Interchanger (TSI). The TSI module rearranges the data stream according to the desired switching pattern and stores it in one row of the memory. For example, if input  $i$  ( $1 \leq i \leq N$ ) has to switch to the output  $j$  ( $1 \leq j \leq M$ ), then the  $i$ th input is copied to the  $j$ th bit position in one row of memory. When this row is read and sent by the  $1 \times M$  DeMUX, the rearrangement results in the data transfer to different outputs from which they were sent. This switching is non-blocking because the presence of data in one time slot does not prevent data from being inserted into others. Moreover,

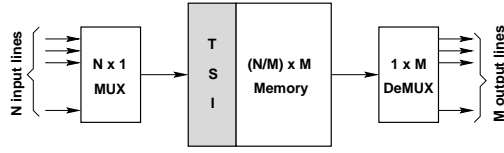


Figure 5: A visualization of a shared memory switch.

although it looks like a  $N \times M$  crossbar, it is much more flexible because it can perform almost any switching pattern on a subset of  $N$  inputs, including multicast and broadcast. Total number of switching functions equals total number of ways to select  $r$  inputs out of  $N$  and distribute them among  $M$  outputs, that is  $\sum_{r=1}^N C_r^N = 2^N$  different switching patterns. Obviously, the TSI operation requires data to be stored temporarily in the memory. The width of memory is obviously  $M$  which is the number of time slots in the data stream. The number of rows, in general, depends on the queue, forwarding and scheduling strategies used in the system. However, it can not be less than  $\lceil N/M \rceil$  to allow enough space for one sweep of all inputs.

Unfortunately, in practice there are some problems with shared memory switches in terms of scalability and throughput beyond certain limits. If the memory is made of SRAM one gets zero wait state performance, but it is difficult to fabricate SRAM memories with large densities. High density SRAM memories are very expensive. If the memory is made of DRAM then one runs into latency problems.

We will walk through both scenarios. Let's say we use commercially available off the shelf DRAM, then one has two problems, they have about 16 data I/O pins that are used for both reading and writing. That rules out simultaneous reads and writes. Furthermore, the bus speed to the DRAM IC is limited to 133MHz (SDRAM). So in a clock cycle one can read or write 1.6 Gbps of data to each of these devices. That looks like a very good speed. However, when data is written into a certain row and one/several columns in that particular row (e.g. input  $i$  to time slot  $j$  only), that whole row has been rendered useless for about 70 ns (memory cell pre-charge time) to access either those same columns within that row or other columns from that row in the near future. This leads to very low utilization of each individual DRAM device. Since data is coming in at rates that are greater than 16 bits at a time at 100 MHz (from multiple input ports) and needs to go out at multiples of 16 bits at 100 MHz one needs to create a large memory array and create a complex controller to manage this array. The complexity is lower than a crossbar but it does not lend itself to scalability very well.

To design a shared memory switch that is  $1024 \times 1024$  one would need a memory bank that is 1024 bits wide and 1024 bits deep and can achieve maximum throughput of  $1024 \times 100MHz = 102.4$  Gbps. Using the 16 bit DRAM devices, one can create a SIMM or DIMM as is done in PCs and other computers to create that array that would require hundreds of devices or large amount of memory. The amount of memory required to make this non-blocking is basically a function of the number of packets one needs to service each clock cycle and the number of input and output ports. For large routers this becomes extremely inefficient in terms of control complexity as well as latency.

SRAM is a better choice in implementing the matrix or the block shown in Figure 5. SRAM requires 6 transistors to store a bit compared to 1 in DRAM, thus it costs roughly 4 to 6 times in terms of silicon area, probably 2 to 3 times in terms of actual cost. Dual ported SRAM requires 8 transistors. Higher density SRAM devices are as much as 50 times more expensive than comparable DRAM devices. At larger router sizes, the memory required to implement a non-blocking switch adds up to Gigabytes in size. Whether it is done in SRAM or DRAM, large implementations do not scale and are custom implementations. The cost and lack of scalability make designing large shared memory switches not practical.

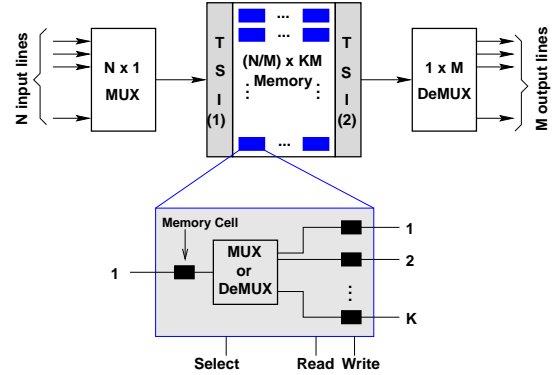


Figure 6: Our mixed space-time switch architecture.

### 3. TERABIT SYSTEM ARCHITECTURE

There are two main components in almost any routing systems which determine their performances. They are the *switch fabric* and *packet scheduler* done often by hardware and software, respectively. These two components are tightly related and improving one without the other fails to enhance the overall performance. The switch fabric determines the switching speed once the data is ready in the input of the switch and the packet scheduler delivers packets from network input lines to the fabric and from fabric to the network output lines. It is necessary for the scheduler to perform these delivery with having various factors, such as fabric speed, sampling rate, buffer size, QoS, etc. in mind. In this section we propose a switch fabric based on mixing the space and time division strategies and introduce an architecture by which packet scheduler can perform its job in a very flexible way.

#### 3.1 Space-Time Division Switch

To achieve high performance for the overall system, our approach is to combine the best qualities of both a crossbar and a shared memory switch. When combined, the switch has low latency of a crossbar switch, with the bandwidth and re-configurability of a shared memory switch. Our switch architecture is shown in Figure 6 and the fundamental building block in our space-time division switch, called *port block*, is shown in Figure 6. A port block essentially implements a  $1 \times K$  or a  $K \times 1$  crossbar switch (e.g.  $1 \times K$  DeMUX or a  $K \times 1$  MUX, respectively) with capability of storing data. This storage capability has been added to match the top level architecture which is supposed to work as a shared memory. Additionally, having small size crossbar with memory cells provides us with shorter latency and more flexibility in forwarding and scheduling the packets without worrying that they may be lost. The complete switching devices are formed using multiple port blocks within a single device. This allows for granularity in switching data as well as scalability.

Note carefully that we use two  $K \times K$  TSIs. The first one places incoming data into any outgoing time slot (i.e. any port block) on the fabric. The second TSI can locate that data from any port block (i.e. any incoming time slot) and place it in any outgoing time slot which will be distributed among the output lines. Each TSI behaves like an  $K \times K$  non-blocking crossbar switch. As discussed for shared memory, The number of rows can not be less than  $\lceil N/M \rceil$  and in general it depends on the queue/scheduling strategy. A combination of port blocks and TSIs may now be used to implement any kind of  $N \times M$  switch of any architecture that one would like to build, Clos, Banyan, Butterfly, etc. This is done by finding an optimal solution on how to map inputs to the available port blocks and eventually forwarding them to the outputs. One flexible technique is to map  $N$  inputs to  $NM$  port blocks and mapping their  $NMK$  outputs to the  $M$  outputs. The aggregate throughput now becomes  $MKf$  which is improvement by factor of  $K$  compared to conventional strategies. Multiple number of these switching devices, as shown in Figure 6, can be cascaded together to make  $N$  and  $M$  as large as the design calls for. The scalability is truly limited by

the number of devices one can accommodate on a single board given the form factor requirements for the networking system. This can be scaled to larger switches by introducing fabrics that contain multiple switching devices. For example, if a  $1024 \times 1024$  switch fabric with port blocks of size  $K = 64$  is used in a routers that run at 100 MHz, it can run at 100% efficiency and achieves the peak throughput on every clock cycle which is  $1024 \times 64 \times 100MHz = 6.5$  Tbps.

The switch fabric is non-blocking because it is still based on the shared memory fundamentals and can sustain inputs to it till all the port blocks in the switch fabric are exhausted. Since each device has upwards of hundreds of port blocks, the fabric effectively implements a switch of a size that is hundreds of input ports by hundreds of output ports with each I/O port being 128 bits wide.

### 3.2 Architecture

The advantages of our architecture are better understood with reference to the latency of the complete routing system. For crossbar architecture, as soon as the data stream comes in, it is parsed and put into a parallel buffer. Once the header has been recovered it is sent to the switch controller and the route processor for processing. When this is completed the controls to the crossbar are asserted to send the data stream in a serial format through an input port of the crossbar switch (once the route lookup, or the physical port to logical port map has been completed). This serial data stream is recovered on the output of the crossbar and stored in a buffer in a parallel format to be serialized and sent back out on the link. This process introduces several delays that contribute to the latency factor. The total latency is summed up to:  $L_{crossbar} = T_{SP} + T_{PS} + T_{lookup} + T_{SP} + T_{PS}$ , where  $T_{SP}$  and  $T_{PS}$  denote serial to parallel and parallel to serial delay of buffers, respectively.  $T_{lookup}$  is the route lookup table latency that researchers have been struggling to minimize and is beyond the scope of this paper.

In the shared memory data flow model, the latency can be summarized as:  $L_{shared} = T_{SP} + T_{PP'} + T_{lookup} + T_{PP} + T_{PS}$ , where  $T_{SP}$  and  $T_{PS}$  (buffer delays) are similar to the ones used in a crossbar architectures. The new buffers, with delay of  $T_{PP'}$  and  $T_{PP}$  are required to make bandwidth adjustments in going from the parallel register to the bandwidth sustainable by the shared memory switch core.

In our mixed (space-time) switch architecture the dataflow has enough bandwidth to eliminate the need for large I/O buffers to the switch core as seen in crossbar and shared memory switches. The latency can be summed up as:  $L_{mixed} = T_{SP} + T_{lookup} + T_{PS}$ . This is several order of magnitude lower as we have effectively eliminated the need for large input and output buffers to the switch core. The complexity of our space-time division switch is almost linear. The fabric is re-configurable and is protocol independent. It can support multiple protocols at the same time i.e. ATM switching, IP switching etc, with or without the added overhead of Multi Protocol Label Switching (MPLS), and loosing the granularity of layer 2 and layer 3 switching. In contrast with traditional architectures (crossbar or shared memory) that suffer from chip/board size explosion, we can design and manage switch of large sizes. More importantly, existing technology allows us to choose  $M$ ,  $K$  and  $f$  to conveniently achieve aggregate throughput ( $MKf$ ) of several hundred Terabit per second.

For the system architecture, we propose a two pronged approach to enabling scalable super Terabit routing. To be useful the Tbps should not indicate the lookup speed from the routing table, but it should reflect the raw throughput of data through the core switch fabric of the network systems. This makes such routing device (e.g. a router) to deliver aggregations and throughputs that are orders of magnitude faster than any comparable systems of today.

Our architecture is shown in Figure 7. There are four key components that enable it to be scalable and deliver the throughput: 1) a

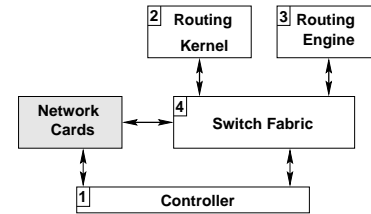


Figure 7: Overview of the system architecture.

controller that is implemented in an ASIC, 2) a routing kernel that is a hardware and firmware component that includes one ASIC and a route coprocessor, 3) a routing engine that implements an intelligent switching/scheduling algorithm in software that is scalable with the capacity of desired networking equipment such as a router; and 4) a switch fabric based on our mixed (time-space division switch strategy) that forms the foundation to the aforementioned three components. These components work together in a tightly knit fashion to deliver a very high performance and throughput that allows the utilization of the switch core to near capacity on every clock cycle.

### 4. CONCLUSION

We proposed a space-time division switch fabric and a network routing system architecture to achieve data rate up to hundreds of Terabit per second. For example, using our system a core router with 100 Tbps can aggregate and switch 10,000 OC-192 pipes in a non-blocking manner. This is an appropriate path of migration to the networks of tomorrow in a cost effective way.

### REFERENCES

- [1] Ryan Hankin Kent, Inc., www.rhk.com, 2001.
- [2] OVUM, Inc., www.ovum.com, 2001.
- [3] Cahners Inc., www.cahners.com, 2001.
- [4] W. Doeringer, G. Karjoth and M. Nassehi, "Routing on Longest-Matching Prefixes," *IEEE Trans. on Networking*, vol. 4, no. 1, pp. 86-97, Feb. 1996.
- [5] P. Newman, G. Minshall and L. Huston, "IP Switching and Gigabit Routers," *IEEE Communications Magazine*, Jan. 1997.
- [6] M. Waldwogel, G. Varghese, J. Turner and B. Plattner, "Scalable High Speed IP Routing Lookups," in Proc. *ACM SIGCOMM Conf.*, Sept. 1997.
- [7] H. Gao, X. Liao, Q. Zhou and W. Ding, "A Novel ATM Switching Fabric," in Proc. *APCC/OECC Conf.*, pp. 287-290, 1999.
- [8] C. Tsui, L. Kwan and C. Lea "VLSI Implementation of a Switch Fabric for Mixed ATM and IP Traffic," in Proc. *ASP-DAC Conf.*, pp. 5-7, 2000.
- [9] A. Lange-Pearson, S. Kumar and M. Shaaban, "Self-Similarity in a Multi-Stage Queueing ATM Switch Fabric," in Proc. *IPCCC Conf.*, pp. 143-150, 2000.
- [10] C. Clos, "A Study of Nonblocking Switching Networks," *Bell System Technical Journal*, 32(2), pp. 406-424, 1953.
- [11] J. Walrand, P. Varaiya, *High Performance Communications Networks*, Second edition, Morgan Kaufman, 1999.
- [12] M. Sexton, A. Reid, *Broadband Networking*, Artech House Publishers, 1999.
- [13] W. Stahlings, *High Speed Networks, TCP/IP and ATM design principles*, Prentice Hall, 1999.
- [14] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick and M. Horwitz, "Tiny Tera: A Packet Switch Core," *IEEE Micro*, pp. 26-33, Jan. 1997.