Localized Message Passing Structure for High Speed Ethernet Packet Switching

Morteza Fayyazi[1]
David Kaeli

Department of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115
mfayyazi,kaeli@ece.neu.edu

Abstract
In this paper, we will introduce two different physical structures and topologies to realize an ethernet layer-2 switch. To evaluate these two structures and produce throughput projections we have utilized MPI. This simulation environment has helped us identify performance bottlenecks and to give an overall estimation of the switch throughput. Our results show that for a hardware implementation of this switch in .18um technology, assuming a 200 MHz clock rate for the processing elements and 100 MHz clock rate for the memory blocks, our design can support multiple gigabit channel ports.

1. Introduction

Over the years there have been a variety of architectures used for packet switching. While the detailed implementations of different vendor commercial packet switches have remained proprietary, the resulting designs have evolved in similar ways. There has been a growing trend towards more parallelism to achieve higher performance. Parallelism is exploited in two ways: first, components that were once shared (such as centralized CPUs and shared memory) are now commonly placed on physically distributed modules [5,6]; second, parallelism is commonly used to process a stream of packets using multiple, identical elements (i.e., using multithreading concepts [8]).

In spite of the rate of acceleration of switch speed (which has increased by a order of over the past few years), there has been a growing gap between switching speed and channel data-rate speed. To cope with the current and future applications, a switch needs to be designed considering scalability and programmability to enable high performance [1]. To address programmability we use programmable processing elements and run-time reconfigurable routing connections between the processing elements and memory blocks. Users can program processing elements to handle different packet formats with different switching algorithms (i.e., store-and-forward or cut-through algorithms.) Reconfigurability of routing connections will result in a traffic-shapeable switch. To address high performance and gigabit-level throughput we use separate memory blocks for different ports with local lookup tables. Uniform structure of the switch and uniform distribution of packet traffic between ports contributes to the scalability of the system.

[1] Corresponding author and presenter

In the next section we describe our two switch structures. In section 3 we describe our MPI/C simulation model for our Ring and Mesh structures. Section 4 provides simulation results. Section 5 provides conclusions and describes future work.

2. Switch structures

In this paper we describe two switch structures: ES-Ring and ES-Mesh. When comparing the two designs, our ES-Ring structure is more efficient when considering area, power and implementation cost, but also results in lower throughput. Next, we explain the functionality of both structures.

2.1 ES-Ring structure

Our ES-Ring model for a five-port switch is shown in Figure 1.

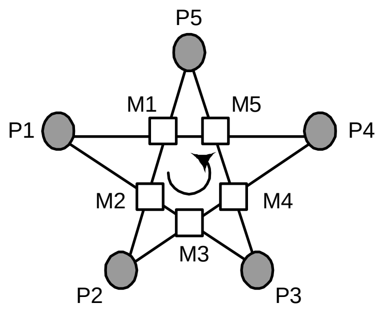

Figure 1. ES-Ring structure for 5-port switch

Pi is a processing element and Mi is a memory block. Each processing element is connected to two memory blocks. Each memory block is shared between two processing elements. Memory blocks are dual-port 16KB SSRAM. There is a hardware-based memory manager residing inside each memory block, which effectively and efficiently handles *malloc* and *free* operations. Each processing element (Pi) has two processing thread elements, the Rx and Tx engines which run in parallel. A local lookup table (i.e. a 128-entry CAM structure with address-hashing update algorithm) is also provided [7]. The Rx thread is responsible for receiving incoming packets, requesting a buffer (128 bytes for cut-through operations, or the size of packet for store-and-forward operations) from the memory block in its left side and storing one buffer of the packet into the memory. The Tx engine reads the destination address of buffers from its right memory block and transmits buffers (if any) with the destination address available in the lookup table. All processing elements are operating (i.e., issuing sends and receives) in parallel. These operations are done in one X-clock. During each X-clock, ports receive and/or transmit packets. At the end of each X-clock, memory blocks rotate one step in a counterclockwise direction. For example, if M1 is on the left side of P1 and M2 is on the right side, after one X-clock, M5 will be on the left side and M1 on the right side. (One implementation for the memory rotation is to use a crossbar switch between processing elements and memory blocks.) In the case of store-and-forward, each processing port has a 2KB buffer. When an entire packet is received, Rx then stores the packet in the left memory block of the processing element. When we use cut-through, Rx stores the packet in separate 128-byte buffers as soon as the partial part of the packet is received. In addition to the packet data, Rx stores the packet information (i.e. packet destination address, source address, size, etc.) to the dedicated part of the memory block. Tx then reads the packet information to decide if the packet needs to be sent out on this port or not.

Assuming that processing elements have 200 MHz clock (Pclk) and the memory blocks use 100MHz clock (Mclk) to connect OC-48 (about 2.5 Gbps) to each port, in the worst case (minimum size packets) we need to handle 4Mpps per port. Therefore each X-clock has to be implemented in 50 Pclk.

Contention for memory blocks are not the bottleneck. A 64-bit bus connection between a Pi and a Mi will provide 6.4 Gbps. Our simulation model shows that the complexity of memory accesses and computations at each X-clock is not exceeding 50 Pclk. Regarding the quality of service, each processing element has a queue of output packets to implement QoS.

2.2 ES-Mesh structure

The ES-Mesh model for a switch with four ports is shown in Figure 2.

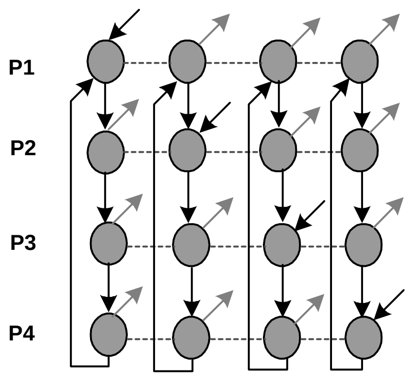

Figure 2. ES-Mesh structure for 4-port switch

In our mesh structure we use an n×n mesh for an n-port switch. Each row in the mesh represents a port and each column is a ring of the ports (similar to the ES-Ring structure.) Not shown in Figure 2 are the memory blocks connected to each node. Each port has n nodes (located in a row.) One node is responsible for receiving packets and n-1 nodes are responsible for sending packets. The nodes of a port (nodes in a row) share a common look up table and controller. Each column is similar to the ES-Ring structure. The resulting packet traffic in each column is less than that experienced in the ES-Ring structure because there is just one port that injects packets to the ring. Memory blocks are implemented using dual-ported memory, therefore when the ports are not broadcasting packets, all ports can simultaneously receive and send packets (if their channels let them to do.) For example, P1 can simultaneously receive data and send it to P2 in the first column and P2 can receive data and send it to P3 in the second column, and so on. Each Pi has several queues of output packets to implement QoS. Multi-Gigabit port switching can be supported by the ES-Mesh structure.

3. MPI/C Modeling

We elected to simulate our two architectures using MPI/C, because of the ease of programming in this environment. This allows us to quickly validate our model before implementing our design in an HDL. Our simulation environment provides MPI running on a cluster of nine Alpha machines. We have simulated both structures in MPI/C.

3.1 ES-Ring

There are two limitations to simulate the switch architecture by MPI. First, in the architecture each memory is a dual-port memory and is shared between two processing elements at each time. Shared memories cannot be implemented by MPI. Second, in the architecture a packet is physically static in a memory block and the connection between memories and the processing elements are dynamic. A dynamic connection cannot be implemented by MPI. To address these limitations, we used a ring of processors with local memories. Each processor simulates one processing element and two memories connected to the processing element. At each X-clock, processors transfer packet(s) to the next processor. The operation can be interpreted as dividing each memory of the architecture into two parts and transferring packets dynamically between the memory partitions instead of dynamically reconfiguring connections. The functionality of the simulated model will be the same as the architecture with some overhead for data communication. Processors in the ring read their packets in parallel from different input files. Each processor has a 2KB Receive and Send buffer. Each buffer can store a maximum size ethernet packet. When a processor reads a packet from its input file, it stores the packet in the receive buffer, derives the packet information and sends the buffer and the packet information to the next processor in the ring. When a processor receives a packet from another processor, it reads the packet information. If the packet has broadcast a tag, the processor sends out the packet (writes to the output file) in addition to transferring the packet to the next processor. If no tag was supplied, the processor performs a lookup of the destination address of the packet in its local table. If the destination exists in the table, the processor sends out the packet to the output file. At this point the packet leaves the system. When the destination does not exist in the table, the processor simply transfers the packet to the next processor. A case can happen such that a packet rotates once in the ring, no processor eliminates the packet and the packet returns to its source processor. In this case, the source processor inserts a broadcast tag in the packet and resends the packet to the ring. This produces a worst-case scenario when calculating delay of the system.

3.2. ES-Mesh

In the ES-Mesh MPI/C model, each node of the mesh is mapped onto one processor. Therefore, for an n-port switch, the model uses n^2 processors (or threads.) Processors in the same column perform the same functionality as in the ES-Ring model, though only one processor will receive packets while the other processors will send packets out. Processors in the same row have similar look-up tables for resolving packet addresses. When a packet is received from a new source, the receiving processor broadcasts the packet information to the all processors in the same row in order to update their look-up tables. Therefore, each sending processor can decide locally whether a packet should be sent out.

4. Simulation results

We have run our simulations with a range of different workloads and calculated the total running time (computation and communication.) Results are shown in Figure 3.

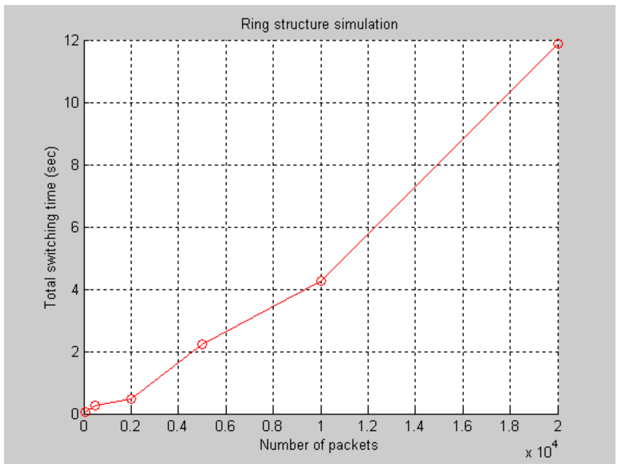

Figure 3. Switching run time on ES-Ring

The input packets into the system have the size of minimum ethernet packets. Destination addresses in the packets are such a way that they should be all broadcasted. Therefore, none of the nodes in the system finds the address in the lookup table. The packets rotate once in the ring and return to the original node. The original node changes the tag of the packets to Broadcast and sends them again into the ring. The result shows that the switching time increases linearly with increased workload. For all cases, we have simulated 10 ports and the switching time is related to the worst case, when all packets are unknown to all ports and the ports broadcast all packets.

In another scenario we examine scalability of two switch structures. In this scenario, each port receives 1000 minimum-sized packets (84-byte) and broadcasts them to all other ports. The total number of packets each port either receives or transmits is 1000 times the number of ports (i.e. if the number of ports is n, each port receives 1000 packets and transmits 1000×(n–1) packets). By calculating total switching time we can estimate overall switch throughput in addition to an average throughput of each port. Throughput of a port is equal to the number of packets received or transmitted per second (pps) by the port. Overall throughput of the switch is equal to the total number of packets received or transmitted per second by the switch. We have calculated throughput for both switch structures when the number of ports varies from 2 to 6. Due to the size of our cluster, we limit the size of our ES-Mesh model to a maximum of 6 ports. A 6-port ES-Mesh model consumes 36 threads running on eight processors. Each thread simulates one node of the ES-Mesh. For the ES-Ring model we have used two processors. Using eight processors for ES-Mesh and two processors for ES-Ring maintains a relatively similar ratio of threads to processors for both structures. For example, when we have 2-port switches, ES-Ring has two nodes and ES-Mesh has four nodes; each model can simulate each node on an individual processor. When the number of ports increases to four, ES-Ring and ES-Mesh will have 4 and 16 nodes, respectively. With 2 processors for ES-Ring and 8 processors for ES-Mesh, each processor has to run two threads to simulate switch nodes. Throughput results of ES-Ring are shown in Figure 4.

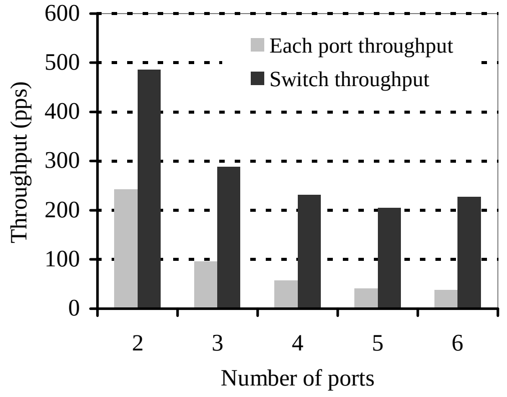

Figure 4. Throughput of ES-Ring

When we have a 2-port switch, throughput of each port is higher than when we have more ports for both structures. The reason is that each node has been simulated on a processor and we do not have multithreading. We have calculated the throughput of the ES-Ring with up to 30 ports (though we only show results for up to six ports in Figure 4) and the results show that when the number of ports increases, the switch throughput of the ES-Ring stays constant and the throughput of each port drops. Therefore, for this scenario ES-Ring is not scalable. These results are for a worst-case scenario when every port is broadcasting minimum size packets to all other ports. Throughput results of ES-Mesh are shown in Figure 5.

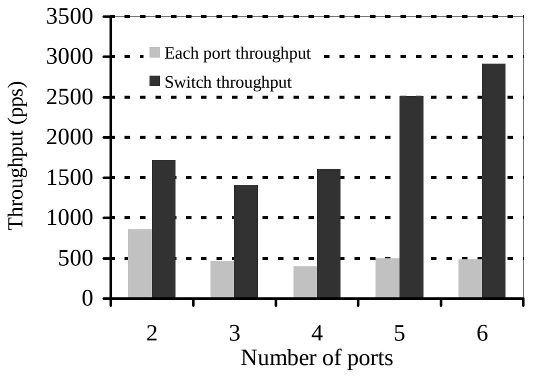

Figure 5. Throughput of ES-Mesh

Figure 5 shows that ES-Mesh is highly scalable. By increasing the number of ports, the switch throughput increases when the throughput of each port stays constant. In last scenario, we compare ES-Ring to ES-Mesh when the internal traffic of the two switches increases and we have port congestion. In this scenario we have a 5-port switch for each structure. The destinations of packets received by each port in ES-Ring and ES-Mesh are shown in Figures 6 and 7, respectively. The dash lines in the figures are used to show the destination of packets.

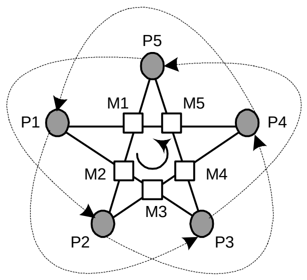

Figure 6. Packet destinations in a case-study of ES-Ring

The problem encountered with ES-Ring for this scenario is that there is congestion on each port, which increases the internal traffic. In ES-Mesh, the congestion is resolved since there are separate paths for each flow of packets.

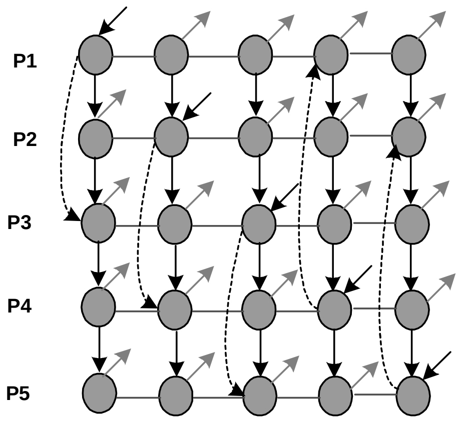

Figure 7. Packet destinations in case-study of ES-Mesh

Figure 8 compares throughput of two structures for this scenario when we inject more packets into the system and increase the internal traffic of both switch structures. The results show that for this case ES-Mesh can switch packets three times faster than ES-Ring when the number of injected packets is increased.

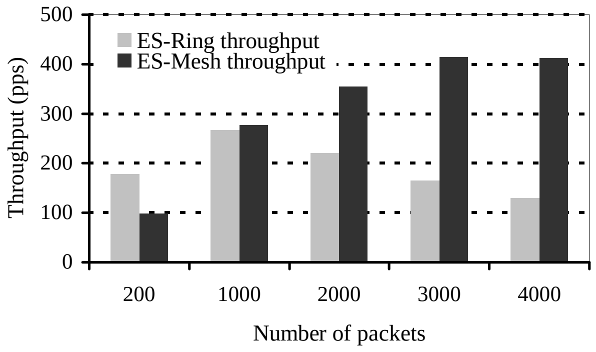

Figure 8. Comparing throughput of ES-Ring versus ES-Mesh

5. Conclusions

In this paper, we introduced two structures for an ethernet layer-2 switch. We used MPI to simulate both architectures and verify the functionality of the models. The proposed architectures can be used to implement multi-gigabit switches. In future work, we will implement the models by HDL environments to obtain a realistic estimation about the speed of the architecture, area and power consumption.

6. Acknowledgments

We would like to thank Mehdi Fakhraie from the University of Tehran for his helpful comments.

References

1. T. Spalink, S. Karlin, and L. Peterson, “Evaluating Network Processors in IP Forwarding,” *Technical Report TR-626-00*, Department of Computer Science, Princeton University, January 2001.

2. S. Walton, A. Hutton and J. Touch, "Efficient High Speed Data Paths for IP Forwarding using Host Based Routers," *Proceedings of the Ninth IEEE Workshop on Local and Metropolitan Area Networks*, 1998.
3. M. Welsh, A. Basu and T. von Eicken, "ATM and Fast Ethernet Network Interfaces for User-level Communication," *Proc. Third International Symposium on High Performance Computer Architecture*, February 1996.
4. R. Sivaram, C.B. Stunkel and D.K. Panda, "HIPIQS: A High-Performance Switch Architecture using Input Queuing," *Proc. of the 11th International Parallel Processing Symposium*, IEEE Computer Society Press, 1998.
5. "A New Architecture for Switch and Router Design," *PMC-Sierra, Inc*., December 1999.
6. "PayloadPlus^{TM} Routing Switch Processor," *Lucent Technologies*, April 2000
7. M. O'Connor and C. A. Gomes, "The IFlow Address Processor," *Silicon Access Networks*, April 2001
8. P. Alexander, "Application of Programmable Multithreaded Network Processor Architecture to OC-192 Packet Forwarding." *Lexra Corporation*, 2001